SC28-6483-2
File No. S370-24

Program Product

IBM OS/VS COBOL Compiler and Library Programmer's Guide

Program Number 5740-CB1
5740-LM1

IBM

SC28-6483-2
File No. S370-24

Program Product

IBM OS/VS COBOL Compiler and Library Programmer's Guide

Program Number 5740-CB1
5740-LM1

IBM

First Edition (June 1984)

Changes are made periodically to the information herein; these changes will be incorporated in new editions of this publication.

Products are not stocked at the address given below. Requests for copies of this product and for technical information about this product should be directed to your IBM marketing representative.

A Program Comment Form (for your comments about the PROFS/PC² product) and a Reader's Comment Form (for your comments about this book) are provided at the back of this publication. If the Program Comment Form has been removed, address comments to: IPS Product Support Center, IBM Corporation, P.O. Box 152560, Irving, Texas 75015-2560. If the Reader's Comment Form has been removed, address comments to: IBM Corporation, Department 6DD, 220 Las Colinas Boulevard, Irving, Texas 75062.

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

Who Should Read this Book?

If you are a business professional who has a need for the decision-making tools available on the IBM Personal Computer as well as the principal support functions available on the IBM Professional Office System (PROFS), or if you want to know how to use the PROFS Personal Computer Connection (PROFS/PC²), you should read this book.

What this Book Will Tell You

This book, *Using the PROFS Personal Computer Connection*, tells you how to install PROFS/PC², what you need to know and do before you use the product, how to use PROFS/PC², how to tailor the main menu to suit your needs, and how to respond to the messages you may get from time to time while you are using PROFS/PC².

You should read the "Introduction" to get an orientation to the whole system and to perform the tasks, especially the one-time tasks, necessary for you to use PROFS/PC². The "Introduction" tells you how to:

- Install PROFS/PC².
- Log on to the VM host where you're running PROFS. (A *host system* is the data processing system to which

The COBOL communications user must write a message control program (MCP) to handle messages transmitted between remote stations and the central computer before they can be processed by a COBOL program. General telecommunications access method (TCAM) information, as well as specific guidelines for creating an MCP, can be found in the publications:

IBM OS/VS Telecommunications Access Method (TCAM) Concepts and Facilities, Order No. GC30-2042

IBM OS/VS TCAM Programmer's Guide, Order No. GC30-2041

The list of publications that follows contains the title and IBM form number of each IBM publication referred to in this publication:

IBM VS COBOL for OS/VS, Order No. GC26-3857

VM/370 CMS User's Guide for COBOL, Order No. SC28-6469

VM/370 System Programmer's Guide, Order No. GC20-1807

OS/VS Linkage Editor and Loader, Order No. GC26-3813

OS/VS1 Planning and Use Guide, Order No. GC24-5090

OS/VS2 Planning Guide for Release 2, Order No. GC28-0667

OS/VS2 Release 2 Guide, Order No. GC28-0671

OS/VS JCL Reference publications:

OS/VS1 JCL Reference, Order No. GC24-5099

OS/VS1 JCL Services, Order No. GC24-5100

OS/VS2 JCL, Order No. GC28-0692

OS/VS Checkpoint/Restart, Order No. GT00-0304

OS/VS1 Storage Estimates, Order No. GC24-5094

OS/VS2 Storage Estimates, Order No. GC28-0604

IBM OS (TSO) COBOL Prompter Terminal User's Guide and Reference, Order No. SC28-6433

OS/VS Data Management publications:

OS/VS1 Data Management for System Programmers, Order No. GC26-3837

OS/VS2 System Programming Library: Data Management, Order No. GC26-3830

OS/VS1 Access Method Services, Order No. GC26-3840

OS/VS2 Access Method Services, Order No. GC26-3841

OS/VS2 Independent Component: Access Method Services, Order No. GT26-3843

OS/VS Tape Labels, Order No. GC26-3795

OS/VS Virtual Storage Access Method (VSAM) Programmer's Guide, Order No. GC26-3838

OS/VS Data Management Services Guide, Order No. GT00-0303

OS/VS Data Management Macro Instructions, Order No. GT00-0305

OS/VS Message Library: Linkage Editor and Loader Messages, Order No. GC38-1007

IBM OS COBOL Interactive Debug Terminal User's Guide and Reference, Order No. SC28-6465

OS/VS1 Debugging Guide, Order No. GC24-5093

OS/VS2 System Programming Library: Debugging Handbook, Order Nos. GC28-0708 and GC28-0709

OS/VS1 Utilities, Order No. GC26-3901

OS/VS2 Utilities, Order No. GC26-3902

OS Sort/Merge Programmer's Guide, Order No. SC33-4007

OS/VS Sort/Merge Programmer's Guide, Order No. SC33-4035

OS Sort/Merge Installation Reference Material, Order No. SC33-4004

OS/VS Sort/Merge Installation Reference Material, Order No. SC33-4034

Industry Standards

The OS/VS COBOL Compiler and Library, Release 2, is designed according to the specifications of the following industry standards (as understood and interpreted by IBM as of April, 1976):

- The highest level of American National Standard COBOL, X3.23-1974 (excepting the Report Writer Module). ANS COBOL X3.23-1974 is compatible with and identical to International Organization for Standardization/Draft International Standard (ISO/DIS) 1989-COBOL.
- The highest level of American National Standard COBOL, X3.23-1968 (including the Report Writer module). ANS COBOL X3.23-1968 is compatible with and identical to ISO/R 1989-1972 Programming Language-COBOL.

Quick Reference to PROFS is a handy reference card to PROFS tasks. Use it as a quick memory jogger. (Order no. GX20-2408)

IBM Personal Computer Publications

For more information about the PC, you may want to refer to the following books:

Disk Operating System, Version 2.0 or 2.1 tells you how to use DOS on the PC. (Order no. 6024001)

IBM Personal Computer 3278/79 Emulation Control Program User's Guide explains how to use your IBM PC or IBM PC XT with the IBM PC 3278/79 Emulation Control Program, that is, the program that lets you emulate, or imitate, the functions of an IBM 3278 typewriter keyboard and either the IBM 3278 Model 2 Display Station or the IBM 3279 Model 2A Color Display Station. (Order no. 1502422)

IBM 3270 Personal Computer Control Program User's Guide and Reference explains keys, functions, and procedures that are used to perform tasks on the IBM 3270 Personal Computer. The IBM 3270 Personal Computer Control Program is designed to operate on the IBM 3270 Personal Computer machine configuration 5271 standard models 2, 4, and 6, when installed with IBM Personal Computer Disk Operating System, Version 2.0 or 2.1. (Order no. 1837434)

The PROFS Personal Computer Connection (PROFS/PC²) lets you work with selected PROFS tasks on the IBM Personal Computer. By using the 3278/79 adapter, the PC 3270, or the IBM Asynchronous Communications Adapter, you can transfer DOS files from the PC to the host system and you can transfer files from the host system to the PC. (Throughout this book, the PC 3270, both hardware and software, and a PC with the 3278/79 adapter card and the 3278/79 Control Program are referred to as the "PC 327x.") Then you can work with these files either on the PC in a stand-alone mode or on the host system. You can move your incoming mail from PROFS to PROFS/PC² to open and respond to at a more convenient time or place. PROFS/PC² lets you look at, reply to, forward, erase, file, and resend notes. You can also create new notes. Actions you take during note processing can be transferred to the PROFS host system for processing or distribution to other PROFS users. If you have an optional printer attached to your Personal Computer, PROFS/PC² lets you print PROFS documents that you've transferred from the host system.

What You Need to Use PROFS/PC²

Hardware

You can perform both host-attached and stand-alone tasks by means of any one of the following products:

- IBM Personal Computer or IBM Personal Computer XT with the following:

CONTENTS

Figures

INTRODUCTION

An OS/VS COBOL program can be processed by the IBM Operating System. The operating system consists of a number of processing programs and a control program.

The processing programs include the COBOL compiler, service programs, and any user-written programs.

The control program supervises the execution or loading of the processing programs; controls the location, storage, and retrieval of data; and schedules jobs for continuous processing.

A request to the operating system for facilities and scheduling of program execution is called a job. For example, a job could consist of compiling a program by utilizing the COBOL compiler. A job consists of one or more job steps, each of which specifies execution of a program. The programmer can make requests to the operating system by using job control statements.

Each job is headed by a JOB statement that identifies the job. Each job step is headed by an EXEC statement that describes the job step and calls for execution. Included in each job step are data definition (DD) statements, which describe data sets and request allocation of input/output devices.

The data processed by execution of any processing program must be in the form of a data set. A data set is a named, organized collection of one or more records that are logically related. Information in a data set may or may not be restricted to a specific type, purpose, or storage medium. A data set may be, for example, a source program, a library of subroutines, or a group of data records that is to be processed by a COBOL program.

A data set resides in one or more volumes. A volume is a unit of external storage that is accessible to an input/output device. For example, a volume may be a reel of tape or it may be a mass storage device.

To facilitate retrieval of a data set, the serial number of the volume upon which it resides can be entered, along with the data set name, in either the system catalog of data sets (SYSCTLG) or in the VSAM catalog, or in both (if they are not the same). The catalog itself is a data set residing on one or more mass storage devices. It is organized into indexes that relate each data set name to its location--the volume in which it resides and its position within the volume. Only the data set name and DISP parameter need be specified to identify a cataloged data set to the system.

The catalog is originally created by a utility program. Once the catalog exists, any non-VSAM data set residing on either a mass storage device or a magnetic tape volume can be cataloged automatically by use of a catalog subparameter in a DD statement that refers to the data set. VSAM data sets are cataloged through Access Method Services.

Several input/output devices grouped together and given a single name when the system is generated constitute a device class. Each device class can be referred to by a collective name. For example, one device class called SYSDA could consist of all the mass storage devices in the installation; another called SYSSQ could consist of all the mass storage devices and tape devices.

EXECUTING A COBOL PROGRAM

Four basic operations are performed to execute a COBOL program:

- Compilation
- Linkage editing
- Loading
- Execution

COMPILATION

Compilation is the process of translating a COBOL source program into a series of instructions comprehensible to the computer, i.e., machine language. In operating system terminology, the input (source program) to the compiler is called the source module. The output (compiled source program) from the compiler is called the object module.

LINKAGE EDITING

The linkage editor is a service program that prepares object modules for execution. It can also be used to combine two or more separately compiled object modules into a format suitable for execution as a single program. The executable output of the linkage editor is called a load module, which must always be stored as a member of a partitioned data set.

In addition to processing object modules, the linkage editor can combine previously edited load modules, with or without multiple object modules, to form one load module.

During the process of linkage editing, external references between different modules are usually resolved.

LOADING

The Loader is a service program that processes object and load modules, resolves any references to subprograms, and executes the loaded module. All these functions are performed in one step. The Loader cannot produce load modules for a program library.

For detailed information on the Loader, see the publication OS/VS Linkage Editor and Loader, where a discussion of invoking the Loader can be found in "Using the Loader."

EXECUTION

Actual execution is under supervision of the control program, which obtains a load module from a library, loads it into main storage, and initiates execution of the machine language instructions contained in the load module.

OPERATING SYSTEM ENVIRONMENTS

The IBM OS/VS COBOL Compiler and Library operates under control of OS/VS1 or OS/VS2 (with or without TSO), and under the CMS component of VM/370. OS/VS1 and OS/VS2 can operate as independent systems or under control of VM/370.

OS/VS1

The OS/VS1 control program divides storage into a number of discrete areas called partitions. Job steps are directed to these partitions using a priority scheduling system; that is, jobs are not executed as encountered in the job stream but according to a priority code. The OS/VS1 control program provides for:

- Priority scheduling of jobs using the class code
- Concurrent scheduling and execution of up to 15 separately protected jobs
- Reading one or more input streams

For further information about the various optional features of the OS/VS1 control program, see the publication OS/VS1 Planning and Use Guide.

OS/VS2

The OS/VS2 control program divides storage into areas called regions. Like OS/VS1, the OS/VS2 control program uses a priority scheduling system and provides for concurrent execution of up to 255 tasks. In addition, the OS/VS2 control program provides for assignment of storage regions on a variable basis according to a region code. For further information about the various optional features of the OS/VS2 control program, see the publication OS/VS2 Planning and Use Guide.

CONVERSATIONAL MONITOR SYSTEM

The Conversational Monitor System (CMS) is a time-sharing system that depends upon the control program component of Virtual Machine Facility/370 (VM/370) for real computer management. CMS provides an extensive range of conversational programming capabilities at a remote terminal. The CMS command language simplifies file and data handling through the use of simple terminal commands. For a detailed description on the use of the OS/VS COBOL Compiler and Library under CMS, see the publication IBM VM/370 CMS User's Guide for COBOL. This guide contains a list of restrictions and limitations for using the OS/VS COBOL Compiler under CMS.

JOB CONTROL PROCEDURES

Communication between the COBOL programmer and the job scheduler is effected through nine job control statements (hereinafter called control statements):

1. Job Statement
2. Execute Statement
3. Data Definition Statement
4. PROC Statement
5. PEND Statement
6. Command Statement
7. Delimiter Statement
8. Null Statement
9. Comment Statement

Parameters coded in these control statements aid the job scheduler in regulating the execution of jobs and job steps, retrieving and disposing of data, allocating input/output resources, and communicating with the operator.

The job statement (hereinafter called the JOB statement) marks the beginning of a job and, when jobs are stacked in the input stream, marks the end of the control statements for the preceding job. It may contain accounting information for use by an installation's accounting routines, give conditions for early termination of the job, regulate the display of job scheduler messages, assign job priority, request a specific class for job scheduler messages, specify the amount of main storage to be allocated to the job, and hold a job for later execution.

The execute statement (or EXEC statement) marks the beginning of a job step and identifies the program to be executed or the cataloged procedure to be used. It may also provide job step accounting information, give conditions for bypassing the job step, pass control information to a processing program, assign a time limit for the execution of the job step and specify the amount of main storage to be allocated.

The data definition statement (or DD statement) describes a data set and requests the allocation of input/output resources. The DD statement parameters identify the data set, give volume and unit information and disposition, and describe the labels and physical attributes of the data set.

The PROC statement appears as the first control statement in a cataloged procedure or an in-stream procedure and is used to assign default values to symbolic parameters defined in the procedure.

The PEND statement appears as the last control statement in an in-stream procedure and marks the end of the in-stream procedure. For further information about in-stream procedures, refer to the topic "Testing a Procedure as an In-Stream Procedure" in the chapter "Using the Cataloged Procedures."

The command statement is used by the operator to enter commands through the input stream. Commands can activate or deactivate system input and output units, request printouts and displays, and perform a number of other operator functions.

The delimiter statement and the null statement are markers in an input stream. The delimiter statement is used, when data is included in the input stream, to separate the data from subsequent control statements. The null statement can be used to mark the end of the control statements for the entire job (preventing subsequent statements from being associated with this job).

The comment statement can be inserted before or after any control statement and can contain any information deemed helpful by the person who codes the control statements. Comments can be coded in columns 4 through 80. The comment cannot be continued onto another statement. If the comment statement appears on a system output listing, it can be identified by the appearance of asterisks in columns 1 through 3.

The sequence of control statements required to specify a job is called a job control procedure.

For example, the job control procedure shown in Figure 1 could be placed in the input stream to compile a COBOL source module.

```
//JOB1      JOB
//STEP1     EXEC  PGM=IKFCBL00,PARM=DECK
//SYSUT1    DD    DSNAME=&&UT1,UNIT=SYSDA,SPACE=(TRK,(40))
//SYSUT2    DD    DSNAME=&&UT2,UNIT=SYSSQ,SPACE=(TRK,(40))
//SYSUT3    DD    DSNAME=&&UT3,UNIT=SYSSQ,SPACE=(TRK,(40))
//SYSUT4    DD    DSNAME=&&UT4,UNIT=SYSSQ,SPACE=(TRK,(40))
//SYSPRINT  DD    SYSOUT=A
//SYSPUNCH  DD    SYSOUT=B
//SYSIN     DD    *
    (source deck)
/*
```

Figure 1. Job Control Procedure

In the illustration, JOB1 is the name of the job. The JOB statement indicates the beginning of a job.

STEP1 is the name of the single job step in the job. The EXEC statement specifies that the IBM OS/VS COBOL Compiler (IKFCBL00) is to execute the job. The statement also specifies that a card deck of the object module is to be produced (PARM=DECK).

The SYSUT1, SYSUT2, SYSUT3, SYSUT4, SYSUT5 (if the SYMDMP or TEST option is specified in the PARM parameter of the EXEC card), and SYSUT6 (if the LVL option is specified in the PARM parameter of the EXEC card) DD statements define utility data sets used by the compiler to process the source module. The names of the data sets defined by SYSUT1, SYSUT2, SYSUT3, SYSUT4, SYSUT5 and SYSUT6 are &&UT1, &&UT2, &&UT3, &&UT4, &&UT5 and &&UT6, respectively. SYSUT1 must be on a mass storage device (UNIT=SYSDA). The system will allocate 40 tracks of space to SYSUT1 [SPACE=(TRK,(40))]. The other four utility data sets are assigned either to any available tape, in which case the SPACE parameter is ignored, or to a mass storage unit (UNIT=SYSSQ).

The SYSPRINT DD statement defines the data set that is to be printed. SYSOUT=A is the standard designation for data sets whose destination is the system output device, usually indicating that the data set is to be listed on a printer.

The SYSPUNCH DD statement defines the data set that is to be punched. By convention, SYSOUT=B designates a card punch.

The SYSIN DD statement defines the data set (in this case, the source module) that is to be used as input to the job step. The asterisk (*) indicates that the input data set follows in the input stream.

The delimiter (/*) statement separates data from subsequent control statements in the input stream.

Output from this job step includes any diagnostic messages associated with the compilation. They are printed in the data set specified by SYSPRINT.

Note: SYSDA, SYSSQ, A, and B are IBM-specified device class names. If they are to be used, they must be incorporated at system generation time.

To avoid rewriting these statements, and the possibility of error, the programmer may place frequently used procedures on a system library called the procedure library. A procedure contained in the procedure library is called a cataloged procedure. A cataloged procedure can be called for execution by placing in the input stream a simple procedure that may require only the JOB and EXEC statements.

If slightly modified, the procedure in the previous example can be cataloged, i.e., placed in the procedure library. For example, if it were cataloged and given the name CATPROC, it could be called for execution by placing the statements shown in Figure 2 in the input stream.

```
//JOB2         JOB
//STEPA        EXEC   PROC=CATPROC
//STEP1.SYSIN DD      *
    (source deck)
/*
```

Figure 2. Using a Cataloged Procedure

In Figure 2, JOB2 is the name of the job. STEPA is the name of the single job step. The EXEC statement calls the cataloged procedure containing STEP1 to execute the job step (PROC=CATPROC).

A procedure can be tested before it is placed in the procedure library by converting it into an in-stream procedure. An in-stream procedure can be executed any number of times during a job. For further information about in-stream procedures, refer to the topic "Testing a Procedure as an In-Stream Procedure" in "Using the Cataloged Procedures."

"User File Processing" and "Appendix I: Checklist for Job Control Procedures" explain, with numerous examples, the preparation of job control procedures. "Data Set Requirements" describes required and optional data sets for compilation, linkage editing, and execution time job steps. The chapter "Using Cataloged Procedures" provides information about using and modifying cataloged procedures.

The section "Control Statements," below, shows the format and use of the parameters and subparameters that can be specified for each job control statement. Some parameters of the statements are described only briefly. For further information, see the publication OS/VS JCL Reference. The syntactic format descriptions in this chapter can be used as a reference for the exact format and for the use of each parameter.

CONTROL STATEMENTS

The COBOL programmer uses the control statements shown in Figure 3 to compile, linkage edit, and execute programs.

Statement	Function
JOB	Indicates the beginning of a new job and describes that job.
EXEC	Indicates a job step and describes that job step; indicates the load module or cataloged procedure to be executed.
DD	Describes data sets, and controls device and volume assignment.
delimiter	Separates data sets in the input stream from control statements; it may follow each data set that appears in the input stream, e.g., after a COBOL source module punched deck.
comment	Contains miscellaneous remarks and notes written by the programmer; it may appear anywhere in the job stream after the JOB statement (except within data or source).

Figure 3. Control Statements

JOB MANAGEMENT

Control statements are processed by a group of operating system routines known collectively as job management. These job management routines interpret control statements and commands, control the flow of jobs, and issue messages to both the operator and the programmer. Job management comprises two major components: a job scheduler and a master scheduler.

The job scheduler is a set of routines that reads input streams, analyzes control statements, allocates input/output resources, issues diagnostic messages to the programmer, and schedules job flow through the system.

The master scheduler is a set of routines that accepts operator commands and acts as the operator's agent within the system. It relays system messages to the operator, performs system functions at his request, and responds to his inquiries regarding the status of a job or of the system. The master scheduler also relays all communication between a processing program and the operator.

PREPARING CONTROL STATEMENTS

Except for the comment statement, control statements are identified by the initial characters // or /* in card columns 1 and 2. The comment statement is identified by the initial characters //* in columns 1 through 3. Control statements may contain four fields: name, operation, operand, and comment, as shown in Figure 4.

Statement	Columns 1	2	3	Fields 4
Job	/	/	name	JOB operand[1] comments[1]
Execute	/	/	name[1]	EXEC operand comments[1]
Data Definition	/	/	name[1]	DD operand comments[1]
Procedure	/	/	name[1]	PROC operand comments[1]
Command	/	/		operation (command) operand comments[1]
Delimiter	/	*		comments[1]
Null	/	/		
Comment	/	/	*	comments
Pend	/	/	name[1]	PEND

[1]Optional.

Figure 4. General Format of Control Statements

Name Field

The name contains from one through eight alphanumeric characters, the first of which must be alphabetic. The name begins in card column 3. It is followed by one or more blanks. The name is used, as follows:

- To identify the control statement to the operating system
- To enable other control statements in the job to refer to information contained in the named statement
- To relate DD statements to files named in a COBOL source program

Operation Field

The operation field is preceded and followed by one or more blanks. It may contain one of the following operation codes:

JOB
EXEC
DD
PROC
PEND

If the statement is a delimiter statement, there is no operation field and comments may start after one blank.

Operand Field

The operand field is preceded and followed by one or more blanks and may continue through column 71 and onto one or more continuation cards. It contains the parameters or subparameters that give required and optional information to the operating system. Parameters and subparameters are separated by commas. A blank in the operand field causes the system to treat the remaining data on the card as a comment. There are two types of parameters: positional and keyword (Figures 5, 6, and 9).

Positional Parameters: Positional parameters are the first parameters in the operand field, and they must appear in the specified sequence. If a positional parameter is omitted and other positional parameters follow, the omission must be indicated by a comma. If other positional parameters do not follow, no comma is needed.

Keyword Parameters: A keyword parameter may be placed anywhere in the operand field following the positional parameters. A keyword parameter consists of a keyword, followed by an equal sign, followed by a single value or a list of subparameters. If there is a subparameter list, it must be enclosed in parentheses or single quotation marks; the subparameters in the list must be separated by commas. Keyword parameters may appear in any sequence.

Subparameters are either positional or keyword. Positional and keyword subparameters for job control statements are shown in Figures 5, 6, and 9. Positional subparameters appear first in the parameter and must be in the specified sequence. If a positional subparameter is

omitted and other positional subparameters follow, a comma must indicate the omission.

Comments Field

Optional comments must be separated from the last parameter (or the /* in a delimiter statement) by one or more blanks and may appear in the remaining columns up to and including column 71. An optional comment may be continued onto one or more continuation cards. Comments can contain blanks.

Note: Comments in the optional comments field follow different procedures from those on the comment statement.

CONVENTIONS FOR CHARACTER DELIMITERS

Commas, parentheses, and blanks are interpreted as character delimiters. If they are not intended by the programmer to be used as delimiters, the fields in which they appear must be enclosed in single quotation marks, indicating that the enclosed information is to be treated as a single field. When an apostrophe (or a single quotation mark, since the same character is used for either) is to be contained within such a field, it must be shown as two consecutive single quotation marks (5-8 punch), not as a double quotation mark (7-8 punch). For example,

Wm. O'Connor

should be shown as

'Wm. O''Connor'

This convention applies to three fields: programmer's name in the JOB statement, information in the PARM parameter of the EXEC statement, and accounting information in the JOB and EXEC statements.

RULES FOR CONTINUING CONTROL STATEMENTS

Except for the comment statement, control statements are contained in columns 1 through 71 of cards or card images. If the total length of a statement exceeds 71 columns, or if a parameter is to be placed on separate cards, the operating system continuation conventions must be used. To continue an operand field:

1. Interrupt the field at the end of a complete parameter or subparameter, including the comma that follows it, at or before column 71.

2. Include any commments desired by following the interrupted field with at least one blank.

3. Optionally, code any nonblank character in column 72.

4. Code the identifying characters // in columns 1 and 2 of the following card or card image.

5. Continue the interrupted operand beginning in any column from 4 through 16.

Comments other than those on a comment statement can be continued onto additional cards after the operand has been completed. To continue a comments field:

1. Interrupt the comment at a convenient place.

2. Code a nonblank character in column 72.

3. Code the identifying characters // in columns 1 and 2 of the following card or card image.

4. Continue the comments field beginning in any column after column 3.

Any control statements in the input stream that the job scheduler considers to contain only continued comments will print on a system output listing with a //* in columns 1 through 3. Comments written on a comment statement cannot be continued.

NOTATION FOR DESCRIBING JOB CONTROL STATEMENTS

The notation used in this publication to define the syntax of job control statements is as follows:

1. The set of symbols below define control statements, but they are never written in an actual statement.

Name	Symbol	Purpose
hyphen	-	Joins lower-case letters, words, and symbols to form a single variable
"or" symbol	|	Indicates alternatives
braces	{ }	Indicate that the enclosed is a group of related items, only one of which is required
brackets	[ ]	Indicate that the enclosed are optional items. Brackets are also used with alternatives to indicate that a default is assumed if no alternative is listed
ellipsis	...	Indicates that the preceding item or group of items can be repeated
superscript	[1] [2] [3]	Indicates a footnote reference

2. Stacked items, enclosed in either brackets or braces, represent alternative items. No more than one of the stacked items can be written by the programmer.

3. Upper-case letters and words, numbers, and the set of symbols listed below are written in an actual control statement exactly as shown in the statement definition. (Any exceptions to this rule are noted in the definition of a control statement.)

Name	Symbol
single quotation mark	'
asterisk	*
comma	,
equal sign	=
parentheses	( )
period	.
slash	/

4. An underscore indicates a default option. If an underscored alternative is selected, it need not be written in the actual statement.

   Note: Many of these defaults can be changed at system generation time.

5. Lower-case letters, words, and symbols appearing in a control statement definition represent variables for which specific information is substituted in the actual statement.

6. Blanks are used in Figures 5, 6, 8, and 9 to improve the readability of control statement definitions. In actual statements, blanks would be interpreted as delimiters.

JOB STATEMENT

The JOB statement is the first statement in the sequence of control statements that describe a job. The JOB statement can contain the following information:

1. Name of the job.
2. Accounting information relative to the job.
3. Programmer's name.
4. Indication of whether or not the job control statements are to be printed on the system output listing.
5. Conditions for terminating the execution of the job.
6. Job priority assignment, job scheduler message class, and real or virtual region size.

Figure 5 is a general format of the JOB statement.

Identifying the Job (jobname)

The jobname identifies the job to the job scheduler. It must satisfy the positional, length, and content requirements for a name field. No two jobs being handled by a priority scheduler should have the same jobname.

Name	Operation	Operand
		Positional Parameters
//jobname	JOB	[([account-number] [,accounting-information])[1] [2] [3]]
		[,programmer-name][4] [5]
		Keyword Parameters
		[MSGLEVEL=(x,y)][6]
		[TIME=(minutes,seconds)]
		[CLASS=jobclass]
		[COND=((code,operator) [,(code,operator)]...[7])[8]]
		[PRTY=job priority]
		[MSGCLASS=classname]
		[REGION=valueK]
		[RD=request]
		[RESTART=({*, stepname, stepname.procstepname} [,checkid])]
		[NOTIFY=user id][9]
		[TYPRUN={HOLD, SCAN}][10]
		[ADDRSPC={VIRT, REAL}]

[1]If the information specified (account-number and/or accounting-information) contains blanks, parentheses, or equal signs, the information must be delimited by single quotation marks instead of parentheses.
[2]If only account-number is specified, the delimiting parentheses may be omitted.
[3]The maximum number of characters allowed between the delimiting quotation marks is 142.
[4]If programmer-name contains any special characters other than the period, it must be enclosed within single quotation marks.
[5]The maximum number of characters allowed for programmer-name is 20.
[6]x = 0, 1, or 2 is the JCL message.
y = 0 or 1 is the allocation message level.
Note that the value 1 may be used in place of (1,1).
[7]The maximum number of repetitions allowed is 7.
[8]If only one test is specified, the outer pair of parentheses may be omitted.
[9]For TSO only.
[10]SCAN is for OS/VS1 only.

Figure 5. JOB Statement

JOB PARAMETERS

Supplying Job Accounting Information

For job accounting purposes, the JOB statement can be used to supply information to an installation's accounting procedures. To supply job accounting information, code the positional parameter first in the operand field.

```
(acct#,additional accounting information)
```

Replace the term "acct#" with the account number to which the job is charged; replace the term "additional accounting information" with other items required by an installation's accounting routines. The requirement can be established with a cataloged procedure for the input reader. Otherwise, the account number is considered optional.

Notes:

- Subparameters of additional accounting information must be separated by commas.
- The number of characters in the account number and additional accounting information must not exceed a total of 142.
- If the list contains only an account number, the programmer need not code the parentheses.
- If the list does not contain an account number, the programmer must indicate its absence by coding a comma preceding the additional accounting information.
- If the account number or any subparameter of additional accounting information contains any special character (except hyphens), the programmer must enclose the number or subparameter in apostrophes (5-8 punch). The apostrophes are not passed as part of the information.

Identifying the Programmer

The person responsible for a job codes his name or identification in the JOB statement, following the job accounting information. This positional parameter is also passed to an installation's routines. As a system generation option, the programmer's name can be established as a required parameter. The requirement can also be established with a cataloged procedure for the input reader. Otherwise, this parameter is considered optional.

Notes:

- The number of characters in the name cannot exceed 20.
- If the name contains special characters other than periods, it must be enclosed in apostrophes. If the special characters include apostrophes, each must be shown as two consecutive apostrophes, e.g., 'T.O''NEILL'.
- If the job accounting information is not coded, the programmer must indicate its absence by coding a comma preceding the programmer-name.
- If neither job accounting information nor programmer-name is present, the programmer need not code commas to indicate their absence.

Displaying All Control Statements, Allocation, and Termination Messages (MSGLEVEL)

The MSGLEVEL parameter indicates whether or not the programmer wants control statements and/or allocation and termination messages to appear in his output listing. To receive this output, code the keyword parameter in the operand field of the JOB statement.

```
MSGLEVEL=(x,y)
```

The letter "x" represents a job control language message code and can be assigned the value 0, 1, or 2. When x = 0 is specified, only the JOB statement, incorrect control statements, and associated diagnostic messages are displayed. When x = 1 is specified, input statements, cataloged procedure statements, and symbolic substitution of parameters are displayed. When x = 2 is specified, only input statements are displayed.

The letter "y" represents an allocation message code and can be assigned the value 0 or 1. When y = 0 is specified, no allocation, termination, or recovery messages are displayed, unless an ABEND occurs during problem program execution. If an ABEND occurs, termination messages are displayed. When y = 1 is specified, all allocation, termination, and recovery messages are displayed.

Notes:

- If the value 1 is selected for both codes, the value may be specified once without the parentheses; i.e., MSGLEVEL=1 is the same as MSGLEVEL=(1,1).
- The default values are taken from the reader procedure.
- If an error occurs on a control statement that is continued onto one or more cards, only one of the continuation cards is printed with the diagnostic messages.

Specifying Conditions for Job Termination (COND)

To eliminate unnecessary use of computing time, the programmer might want to base the continuation of a job on the successful completion of one or more of

its job steps. At the completion of each job step, the processing program passes a number to the job scheduler as a return code. The COND parameter provides the means to test each return code as many as eight times. If any one of the tests is satisfied, subsequent steps are bypassed and the job is terminated.

To specify conditions for job termination, code the keyword parameter in the operand field of the JOB statement.

```
COND=((code,operator),...,(code,operator))
```

See the COND parameter on the EXEC statement for a discussion of the operator values and the codes issued by the compiler and linkage editor at the end of a job step.

Note:

- The subparameters EVEN and ONLY cannot be specified as part of the COND parameter on the JOB statement.

Requesting Restart for a Job (RD)

The restart facilities are used in order to minimize the time lost in reprocessing a job that abnormally terminates. These facilities permit execution of jobs that abnormally terminate to be automatically restarted.

Execution of a job can be automatically restarted at the beginning of the job step that abnormally terminated (step restart) or within the step (checkpoint restart). In order for checkpoint restart to occur, the CHKPT macro instruction must have been executed in the processing program prior to abnormal termination. The CHKPT macro instruction is activated by the COBOL source language RERUN clause. The RD parameter specifies that step restart can occur or that the action of the CHKPT macro instruction is to be suppressed.

To request that step restart be permitted or to request that the action of the RERUN clause be suppressed, code the keyword parameter in the operand field of the JOB statement.

```
RD=request
```

Replace the word "request" with:

R -- to permit automatic step restart

NC -- to suppress the action of the CHKPT macro instruction and not to permit automatic restart or deferred restart

NR -- to request that the CHKPT macro instruction be allowed to establish a checkpoint, but not to permit automatic restart. Deferred restart is permitted through specification of RESTART on the resubmitted job.

RNC -- to permit step restart and to suppress the action of the CHKPT macro instruction

Each of these requests is described in greater detail in the following paragraphs.

RD=R: If the processing programs used by the job do not include any CHKPT macro instructions, RD=R allows execution to be resumed at the beginning of the step that causes abnormal termination. If any of the programs do include one or more CHKPT macro instructions, step restart can occur if a step abnormally terminates before execution of a CHKPT macro instruction; thereafter, checkpoint restart can occur.

RD=NC or RD=RNC: RD=NC or RD=RNC should be specified to suppress the action of all CHKPT macro instructions included in the programs. When RD=NC is specified, neither step restart nor checkpoint restart can occur. When RD=RNC is specified, step restart can occur.

RD=NR: RD=NR permits a CHKPT macro instruction to establish a checkpoint, but does not permit automatic restart. Instead, at a later time, the job can be resubmitted and execution can begin at a specific checkpoint. (Resubmitting a job for restart is discussed later.)

Before automatic step restart occurs, all data sets in the restart step with a status of OLD or MOD, and all data sets being passed to steps following the restart step, are kept. All data sets in the restart step with a status of NEW are deleted. Before automatic checkpoint restart occurs, all data sets currently in use by the job are kept.

If the RD parameter is omitted and no checkpoints are taken, automatic restart cannot occur. If the RD parameter is omitted but one or more checkpoints are taken, automatic checkpoint restart can occur.

Notes:

- For OS/VS1 restart can occur only if MSGLEVEL=1 is coded on the JOB statement.
- If step restart is requested, each step must be assigned a unique step name.
- If no RERUN clause is specified in the user's program, no checkpoints are written regardless of the disposition of the RD parameter.

Reference:

- For detailed information on the checkpoint/restart facilities, see the publication OS/VS Checkpoint/Restart.

Resubmitting a Job for Restart (RESTART)

The restart facilities can be used if the job is abnormally terminated and the programmer wants to resubmit the job for execution. These facilities reduce the time required to execute the job since execution of the job is resumed, not repeated.

Execution of a resubmitted job can be restarted at the beginning of a step (step restart) or within a step (checkpoint restart). In order for checkpoint restart to occur, a program must previously have had a checkpoint record written. The RESTART parameter specifies where execution is to be restarted.

If execution is to be restarted at a particular job step, code the keyword parameter in the operand field of the JOB statement before resubmitting the job.

```
RESTART=stepname
```

Replace the word "stepname" with the name of the step at which execution is to be restarted. Replace stepname with an asterisk (*) if execution is to be restarted at the first job step.

If execution is to be restarted at a particular checkpoint within a particular job step, code the keyword parameter in the operand field of the JOB statement before resubmitting the job.

```
RESTART=(stepname,checkid)
```

Replace the word stepname with the name of the step in which execution is to be restarted. Replace the term "checkid" with the 1- to 16-character name that identifies the checkpoint within the step.

If execution is to be restarted at a checkpoint, the resubmitted job must include an additional DD statement. This DD statement defines the checkpoint data set and has the ddname SYSCHK. Do not include a SYSCHK DD statement if step restart is to be performed.

If the RESTART parameter is not specified on the JOB statement of the resubmitted job, execution is repeated.

Notes:

- If execution is to be restarted at or within a cataloged procedure step, give both the name of the step that invokes the procedure and the procedure step name, as below.

```
RESTART=stepname.procstepname
```

- If step restart is performed, generation data sets that were created and cataloged in steps preceding the restarted step must not be referred to in the restart step or in steps following the restart step by means of the same relative generation numbers that were used to create them. For example, a generation data set assigned a generation number of +1, would be referred to as 0 in the restart step or steps following the restart step.
- Backward references cannot be made to steps that precede the restart step using the following keyword parameters: PGM, COND, SUBALLOC, and VOLUME=REF, unless in the last case the referenced statement includes VOLUME=SER=(ser#).

Reference:

- For detailed information on the checkpoint/restart facilities, see the publication OS/VS Checkpoint/Restart.

PRIORITY SCHEDULING JOB PARAMETERS

Setting Job Time Limits (TIME)

To assign a limit to the computing time used by a job, code the keyword parameter in the operand field.

```
TIME=(minutes,seconds)
```

Such an assignment is useful in a multiprogramming environment where more than one job has access to the computing system. The time is coded in minutes and seconds to represent the maximum time for execution of a job.

Notes:

- The number of minutes cannot exceed 1439 and the number of seconds cannot exceed 59. If the job is not completed in this time it is terminated.
- If the job requires use of the system for more than 24 hours (1439 minutes) specify TIME=1440. This number suppresses job timing.
- If the time limit is given in minutes only, the parentheses need not be coded; e.g., TIME=5.
- If the time limit is given in seconds, the comma must be coded to indicate the absence of minutes; e.g., TIME=(,45).
- If the TIME parameter is omitted, the default job time is assumed.

Assigning a Job Class (CLASS)

To assign a job class to a job, code the keyword parameter in the operand field of the JOB statement.

```
CLASS=jobclass
```

The meaning and use of the term "jobclass" is pre-defined by each installation. If the CLASS parameter is omitted, the default job class of A is assigned to the job.

Note:

- If an installation provides time-slicing facilities in an OS/VS1 system, the CLASS parameter can be used to make the job part of the group of jobs to be time-sliced. Time-slicing permits the processing of tasks of equal priority so that each is executed for its specified period of time. At system generation, a group of contiguous partitions are selected to be used for time-slicing, and each partition is assigned at least one job class. If the job is to be time-sliced, specify a class that was assigned only to the partitions selected for time-slicing.

Assigning Job Priority (PRTY)

To assign a priority other than the default job priority (as established in the input reader procedure), code the keyword parameter in the operand field of the JOB statement.

```
PRTY=nn
```

Replace the letters "nn" with a decimal number from 0 through 13 (the highest priority number is 13).

If an installation provides time-slicing facilities in a system with OS/VS2, the PRTY parameter can be used to make the job part of a group of jobs to be time-sliced. At system generation, the priority of the time-sliced group is selected. If the job priority number specified corresponds with the priority number selected for time-slicing, then the job will be time-sliced.

If the PRTY parameter is omitted, the default job priority is assigned to the job.

Note: Whenever possible, avoid using priority 13. This is used by the system to expedite processing of jobs in which certain errors were diagnosed. It is also intended for other special uses by future features of systems with priority schedulers.

Requesting a Message Class (MSGCLASS)

With the quantity and diversity of data in the output stream, an installation may want to separate different types of output data into different classes. Each class is directed to an output writer associated with a specific output unit. The MSGCLASS

parameter allows routing of all messages issued by the job scheduler to an output class other than the normal message class, A.

To choose such a class, code the keyword parameter in the operand field of the JOB statement.

```
MSGCLASS=x
```

Replace the letter "x" with an alphabetic (A-Z) or numeric (0-9) character. An output writer, which is assigned to process this class, will transfer this data to a specific device.

If the MSGCLASS parameter is omitted, job scheduler messages are routed to the standard output class, A.

Reference:

- For a more detailed discussion of output classes, see the appropriate Planning and Use Guide.

Specifying Main Storage Requirements for a Job (REGION)

For jobs that require an unusual amount of main storage, the JOB statement provides the REGION parameter. The REGION parameter specifies the maximum amount of main storage to be allocated to the job. This amount must include the size of those components required by the user's program that are not resident in main storage.

The REGION parameter is used in conjunction with the ADDRSPC parameter to determine the total amount of main storage available to a program and to either allow or disallow paging.

Note: The REGION parameter has different meanings for OS/VS1 and OS/VS2. See the publication OS/VS JCL Services for detailed information.

To specify a region size, code the keyword parameter in the operand field of the JOB statement.

```
REGION=(nnnnnxK[,nnnnnyK])
```

To request the maximum amount of main storage required by the job, the term "nnnnnx" should be replaced with the number of 1024-byte areas allocated to the job, e.g., REGION=52K. This number can range from 1 to 5 digits but cannot exceed 16383.

If the REGION parameter is omitted or if a region size smaller than the default region size is requested, it is assumed that the default value is that established by the input reader procedure.

The REGION parameter can be used with either the VIRT or REAL options of the ADDRSPC parameter.

Notes:

- Region sizes for each job step can be coded by specifying the REGION parameter in the EXEC statement for each job step. However, if a REGION parameter is present in the JOB statement, it overrides REGION parameters in EXEC statements.

- For information on storage requirements to be considered when specifying a region size, see the appropriate Storage Estimates publication.

Holding a Job for Later Execution

To temporarily prevent a job from being selected for processing, code the keyword parameter in the operand field of the JOB statement.

```
TYPRUN={HOLD}
       {SCAN}
```

The job is then held until a RELEASE command is issued by the operator. This specification is particularly useful when one job must be run after another job has terminated.

HOLD specifies that the job is to be held until the operator issues a RELEASE command. SCAN (for OS/VS1 only) specifies that the JCL for a job is to be scanned for syntax errors but that the job is not to be executed. If SCAN is specified for OS/VS2, a JCL error will occur.

Specifying Address Space (ADDRSPC)

To take advantage of the storage facilities offered by OS/VS1 and OS/VS2, always specify ADDRSPC = VIRT or omit the parameter.

```
ADDRSPC={VIRT}
        {REAL}
```

Note that the compiler and its object code, including library subroutines, can run VIRTUAL, and should be run that way unless a non-COBOL program in the partition requires the REAL option.

EXEC STATEMENT

The EXEC statement defines a job step and calls for its execution. It contains the following information:

1. The name of a load module or the name of a cataloged procedure that contains the name of a load module that is to be executed. The load module can be the COBOL compiler, the linkage editor, the loader, or any COBOL program in load module form.
2. Accounting information for this job step.
3. Conditions for bypassing the execution of this job step.
4. Computing time for a job step or cataloged procedure step, and main storage region size.
5. Compiler, linkage editor, or loader options chosen for the job step.

Figure 6 is the general format of the EXEC statement.

Note: If the information specified is normally delimited by parentheses but contains blanks, parentheses, or equal signs, it must be delimited by single quotation marks instead of parentheses.

Identifying the Step (stepname)

The stepname identifies a job step within a job. It must satisfy the positional, length, and content requirements for a name field. The programmer must specify a stepname if later control statements refer to the step.

POSITIONAL PARAMETERS

Identifying the Program (PGM) or Procedure (PROC)

The EXEC statement identifies the program to be executed in the job step with the PGM parameter. To specify the COBOL compiler, code the positional parameter in the first position of the operand field of the EXEC statement.

```
| PGM=IKFCBL00 |
```

It indicates that the COBOL compiler is the processing program to be executed in the job step.

To specify the linkage editor, code the positional parameter in the first position of the operand field of the EXEC statement.

```
| PGM=IEWL |
```

This indicates that the linkage editor is the processing program to be executed in the job step.

The PGM parameter depends upon the type of library in which the program resides. If the job step uses a cataloged procedure, the EXEC statement identifies it with the PROC parameter, in place of the PGM parameter.

Name	Operation	Operand
		Positional Parameters
//[stepname][1]	EXEC	{PGM=progname PGM=*.stepname.ddname PROC=procname procname PGM=*.stepname.procstep.ddname}
		Keyword Parameters
		[{ACCT[2] ACCT.procstep} = (accounting-information)[3 4 5]]
		[{COND[2] COND.procstep} = ((code,operator[,stepname[.procstep]])...)[6 7]]
		[{PARM[2] PARM.procstep} = (option[,option]...)[3 8 9]]
		[{TIME TIME.procstep} = (minutes,seconds,)]
		[{REGION REGION.procstep} = valueK]
		[{RD RD.procstep} = request]
		[{DPRTY DPRTY.procstep} = (value 1, value 2)]
		[{ADDRSPC ADDRSPC.procstep} = {VIRT REAL}]

[1]Stepname is required when information from this control statement is referred to in a later job step.
[2]If this format is selected, it may be repeated in the EXEC statement once for each step in the cataloged procedure.
[3]If the information specified contains any special characters except hyphens, it must be delimited by single quotation marks instead of parentheses.
[4]If accounting-information contains any special characters except hyphens, it must be delimited by single quotation marks.
[5]The maximum number of characters allowed between the delimiting quotation marks or parentheses is 142.
[6]The maximum number of repetitions allowed is 7.
[7]If only one test is specified, the outer pair of parentheses may be omitted.
[8]If the only special character contained in the value is a comma, the value may be enclosed in quotation marks.
[9]The maximum number of characters allowed between the delimiting quotation marks or parentheses is 100.

Figure 6. EXEC Statement

1. Temporary libraries are temporary partitioned data sets created to store a program until it is used in a later job step of the same job. This type of library is particularly useful for storing the program output of a linkage editor run until it is executed in a later job step. To execute a program from a temporary library, code the positional parameter in the first position of the operand field of the EXEC statement.

```
PGM=*.stepname.ddname
```

The asterisk (*) indicates the current job step. Replace the terms stepname and ddname with the names of the job step and the DD statement within the procedure step, respectively, in which the temporary library is created.

If the temporary library is created in a catalogued procedure step, in order to call it in a later job step outside the procedure, give both the name of the job step that calls the procedure and the procedure stepname by coding the positional parameter in the first position of the operand field of the EXEC statement.

```
PGM=*.stepname.procstepname.ddname
```

2. The system library is a partitioned data set named SYS1.LINKLIB that contains nonresident control program routines, and processor programs. To execute a program that resides in the system library, code the positional parameter in the first position of the operand field.

```
PGM=progname
```

Replace the term progname with the member name or alias associated with this program. This same keyword parameter can be used to execute a program that resides in a private library. Private libraries are made available to a job with a special DD statement (see "Additional DD Statement Facilities").

3. Instead of executing a particular program, a job step may use a cataloged procedure. A cataloged procedure can contain control statements for several steps, each of which executes a particular program. Cataloged procedures are members of a library named SYS1.PROCLIB. To request a cataloged procedure, code the positional parameter in the first position of the operand field of the EXEC statement.

```
PROC=procname
```

Replace the term procname with the unqualified name of the cataloged procedure (see "Using the DD Statement" for a discussion of qualified names).

Note: A procedure may be tested before it is placed in the procedure library by converting it into an in-stream procedure and placing it within the job step itself. In-stream procedures are discussed in the section, "Testing a Procedure as an In-Stream Procedure" in the chapter "Using the Cataloged Procedures."

KEYWORD PARAMETERS

Specifying Job Step Accounting Information (ACCT)

When executing a multistep job, or a job that uses cataloged procedures, the programmer can use this parameter so that jobsteps are charged to separate accounting areas. To specify items of accounting information to the installation accounting routines for this job step, code the keyword parameter in the operand field of the EXEC statement.

```
ACCT=(accounting information)
```

Replace the term "accounting information" with one or more subparameters separated by commas. If both the JOB and EXEC statements contain accounting information, the installation accounting routines decide how the accounting information shall be used for the job step.

To pass accounting information to a step within a cataloged procedure, code the keyword parameter in the operand field of the EXEC statement.

```
ACCT.procstep=(accounting information)
```

Procstep is the name of the step in the cataloged procedure. This specification overrides the ACCT parameter in the named procedure step, if one is present.

Specifying Conditions for Bypassing or Executing the Job Step (COND)

The execution of certain job steps is based on the success or failure of preceding steps. The COND parameter provides the means to:

- Make as many as eight tests on return codes issued by preceding job steps or cataloged procedure steps, which were completed normally. If any one of the tests is satisfied, the job step is bypassed.
- Specify that the job step is to be executed even if one or more of the preceding job steps abnormally terminated or only if one or more of the preceding job steps abnormally terminated.

To specify conditions for bypassing a job step, code the keyword parameter in the operand field of the EXEC statement.

```
COND=((code,operator,[stepname]),...,
                (code,operator,[stepname]))
```

The term "code" may be replaced by a decimal numeral to be compared with the job step return code. The return codes for both the compiler and the linkage editor are:

00 Normal conclusion

04 Warning messages have been listed, but program is executable.

08 Error messages have been listed; execution may fail.

12 Severe errors have occurred; execution is impossible.

16 Terminal errors have occurred; execution of the processor has been terminated.

The compiler issues a return code of 16 when any of the following are detected:

- BASIS member-name is specified and no member of that name is found
- Required device not available
- Not enough main storage is available for the tables required for compilation
- A table exceeded its maximum size
- A permanent input/output error has been encountered on an external device

The return codes have a correlation with the severity level of the error messages. With linkage editor messages, for example, the rightmost digit of the message number states the severity level; this number is multiplied by 4 to get the appropriate return code. With the COBOL compiler, 04, 08, 12, and 16 are equal to the severity flags: W, C, E, and D, respectively.

The term "operator" specifies the test to be made of the relation between the programmer-specified code and the job step return code. Replace the term operator with one of the following:

```
GT  (greater than)
GE  (greater than or equal to)
EQ  (equal to)
LT  (less than)
LE  (less than or equal to)
NE  (not equal to)
```

The term "stepname" identifies the previously executed job step that issued the return code to be tested and is replaced by the name of that preceding job step. If stepname is not specified, code is compared to the return codes issued by all preceding steps in the job.

Replace the term stepname with the name of the preceding job step that issues the return code to be tested.

If the programmer codes

```
COND=((4,GT,STEP1),(8,EQ,STEP2))
```

the statement is interpreted as: "If 4 is greater than the return code issued by STEP1, or if STEP2 issues a return code of 8, this job step bypassed."

Notes:

- If only one test is made, the programmer need not code the outer parentheses, e.g., COND=(12,EQ,STEPX).
- If each return code test is made on all preceding steps, the programmer need not code the terms stepname, e.g., COND=((4,GT),(8,EQ)).
- When the return code is issued by a cataloged procedure step, the programmer may want to test it in a later job step outside of the procedure. In order to test it, give both the name of the job step that calls the procedure and the procedure stepname, e.g., COND=((code,operator,stepname.procstep),...).

Abnormal termination of a job step normally causes subsequent steps to be bypassed and the job to be terminated. By means of the COND parameter, however, the programmer can specify execution of a job step after one or more preceding job steps have abnormally terminated. For the COND parameter, a job step is considered to terminate abnormally if a failure occurs within the user's program once it has received control. (If a job step is abnormally terminated during scheduling because of failures such as job control language errors or inability to allocate space, the remainder of the job steps are bypassed, whether or not a condition for executing a later job step was specified.)

To specify the condition for executing a job step, code the keyword parameter in the operand field of the EXEC statement.

```
                    {EVEN}
          COND=     {ONLY}
```

The EVEN or ONLY subparameters are mutually exclusive. The subparameter selected can be coded in combination with up to seven return code tests, and can appear before, between, or after return code tests, e.g.,

```
COND=(EVEN,(4,GT,STEP3))

COND=((8,GE,STEP1),(16,GE),ONLY)
```

The EVEN subparameter causes the step to be executed even when one or more of the preceding job steps have abnormally terminated. However, if any return code tests specified in this job step are satisfied, the step is bypassed. The ONLY subparameter causes the step to be executed only when one or more of the preceding job steps have abnormally terminated. However, if any return code tests specified in this job step are satisfied, the step is bypassed.

When a job step abnormally terminates, the COND parameter on the EXEC statement of the next step is scanned for the EVEN or ONLY subparameter. If neither is specified, the job step is bypassed and the EXEC statement of the next step is scanned for the EVEN or ONLY subparameter. If EVEN or ONLY is specified, return code tests, if any, are made on all previous steps specified that executed and did not abnormally terminate. If any one of these tests is satisfied, the step is bypassed. Otherwise, the job step is executed.

If the programmer codes

```
COND=EVEN
```

the statement is interpreted as: "Execute this step even if one or more of the preceding steps abnormally terminated during execution." If COND=ONLY is coded, it is interpreted as: "Execute this step only if one or more of the preceding steps abnormally terminated during execution."

If the COND parameter is omitted, no return code tests are made and the step will be bypassed when any of the preceding job steps abnormally terminate.

Notes:

- When a job step that contains the EVEN or ONLY subparameter refers to a data set that was to be created or cataloged in a preceding step, the data set will not exist if the step creating it was bypassed.
- When a jobstep that contains the EVEN or ONLY subparameter refers to a data set that was to be created or cataloged in a preceding step, the data set may be incomplete if the step creating it abnormally terminated.
- When the job step uses a cataloged procedure, the programmer can establish return code tests and the EVEN or ONLY subparameter for a procedure step by including, as part of the keyword COND, the procedure stepname, e.g., COND.procstepname. This specification overrides the COND parameter in the named procedure step if one is present. The programmer can code as many parameters of this form as there are steps in the cataloged procedure.
- To establish one set of return code tests and the EVEN or ONLY subparameter

for all steps in a procedure, code the COND parameter without a procedure stepname. This specification replaces all COND parameters in the procedure if any are present.

Job steps following a step that abnormally terminates are normally bypassed. If a job step is to be executed even if a preceding step abnormally terminates, specify this condition, along with up to seven return code tests:

```
//STEP3 EXEC PGM=CONVERT,                X
//   COND=(EVEN,(4,EQ,STEP1)),...
```

Here, the step is executed if the return code test is not satisfied, even if one or more of the preceding job steps abnormally terminated. If a job step is to execute only when one or more of the preceding steps abnormally terminate, replace EVEN in the above example with ONLY.

If the EXEC statement calls a cataloged procedure, the programmer can establish return code tests and the EVEN or ONLY subparameter for a procedure step by coding the COND parameter followed by the name of the procedure step to which it applies:

```
//STEP4 EXEC ANALYSIS,COND.              X
// REDUCE=((16,EQ,STEP4.LOOKUP),ONLY),...
```

Here, the cataloged procedure step named REDUCE will be executed only if a preceding job step has abnormally terminated and the procedure step named LOOKUP does not issue a return code of 16. The programmer can code as many COND parameters of this type as there are steps in the procedure.

Passing Information to the Processing Program (PARM)

For processing programs that require control information at the time they are executed, the EXEC statement provides the PARM parameter. To pass information to the program, code the keyword parameter in the operand field.

```
PARM=(option[,option]...)
```

This will pass options to the compiler, linkage editor, loader, or object program when any one of them is called by the PGM parameter in the EXEC statement or to the first step in a cataloged procedure.

To pass options to a compiler, the linkage editor, loader, or the execution step within the named cataloged procedure step, code the keyword parameter in the operand field.

```
PARM.procstep=(option[,option]...)
```

Any PARM parameter already appearing in the procedure step is deleted, and the PARM parameter that is passed to the procedure step is inserted.

A maximum of 100 characters may be written between the parentheses or single quotation marks that enclose the list of options. The COBOL compiler selects the valid options of the PARM field for processing by looking for significant characters (usually three) of each key option word. When the keyword is identified, it is checked for the presence or absence of the prefix NO, as appropriate. The programmer can make the most efficient use of the option field by using the significant characters instead of the entire option. Figure 7 lists the significant characters for each option (see "Options for the Compiler" for an explanation of each).

Option	Significant Characters
ADV	ADV
APOST	APO
BATCH	BAT
BUF	BUF
CDECK	CDE
CLIST	CLI
CSYNTAX	CSY
COUNT	COU
DECK	DEC
DMAP	DMA
DUMP	DUM
DYNAM	DYN
ENDJOB	END
FDECK	FDE
FLAGE(W)	LAG, LAGW
FLOW	FLO
LANGLVL	LANGLVL
LCOL1/LCOL2	OL1/OL2
LIB	LIB
LINECNT	CNT
LOAD	LOA
LSTONLY/LSTCOMP	LSTO/LSTC
LVL	LVL
L132/L120	L13/L12
NAME	NAM
NUM	NUM
OPTIMIZE	OPT
PMAP	PMA
PRINT	PRI
QUOTE	QUO
RESIDENT	RES
SEQ	SEQ
SIZE	SIZ
SOURCE	SOU
SPACE	ACE
STATE	STA
SUPMAP	SUP
SXREF	SXR
SYMDMP	SYM
SYNTAX	SYN
SYSx	SYS
TERM	TER
TEST	TES
TRUNC	TRU
VBSUM	VBS
VBREF	VBR
VERB	VER
XREF	XRE
ZWB	ZWB

Figure 7. Significant Characters for Compiler Options

Note: The compiler recognizes the significant characters to set the proper options. If an option is incorrectly spelled, the compiler default option is used.

Options for the Compiler

The IBM-supplied default options indicated by an underscore in the following discussion can be changed when the compiler is installed. The format of the PARM parameter is illustrated in Figure 8.

Notes:

- When a subparameter contains an equal sign, the entire information field of the PARM parameter must be enclosed by single quotation marks instead of parentheses, e.g.,

  PARM='SIZE=160000,PMAP'. This is the recommended (that is, most efficient) technique. Other ways of specifying special characters in the PARM parameter are described in OS/VS JCL Reference.

- When an option and its default (such as XREF and NOXREF) are both specified, the last encountered option is generally the one assumed. (Exceptions to this rule are cited in the option descriptions.) Accordingly, the programmer may change one of the many options without repunching the entire EXEC card.

SIZE=yyyyyyy
: indicates the amount of main storage, in bytes, available for compilation (see "Machine Considerations"). The COBOL default value is 131,072 bytes, or 128K.

BUF=yyyyyy
: indicates the amount of main storage to be allocated to buffers. If both SIZE and BUF are specified, the amount allocated to buffers is included in the amount of main storage available for compilation (see "Appendix D: Compiler Optimization" for information about how buffer size is determined).

  Note: The SIZE and BUF compile-time parameters can be given in multiples of K, where K=1024 decimal bytes. For example, 131,072 decimal bytes can be specified as 128K.

LANGLVL(1/2)
: specifies whether the 1968 or the 1974 American National Standard COBOL definition (as understood and interpreted by IBM) is to be used when compiling those source elements whose meaning has changed. LANGLVL(1) tells the compiler to use the 1968 ANS standard (X3.23-1968) if the compiler encounters any of those source elements whose definition has changed; this interpretation would be the one that was used by Release 1 of the compiler. LANGLVL(2) tells the

compiler to use the 1974 ANS standard (X3.23-1974) when encountering any of those redefined elements. LANGLVL(2) is the default.

Generally speaking, the language level supported by the Release 2 compiler includes all of that supported by Release 1. The Release 2 compiler will accept not only source programs written in the new (1974) language, but also source programs that were or are written in the older (1968) language. However, the superset relationship between the new and the older languages is not absolute; there are a few exceptions--elements whose meaning has changed because of ANS redefinition. It is only these few elements that are controlled by the LANGLVL option.

(These elements are identified in Section III of Appendix A of IBM VS COBOL for OS/VS.) Language elements whose meanings did not change from 1968 ANS to 1974 ANS (the vast majority of the language) are unaffected by the LANGLVL option, as are all IBM extensions and those language elements new in the 1974 ANS definition. The compiler accepts all such elements regardless of the LANGLVL specification. LANGLVL affects only that small percentage of elements whose definition changed from 1968 to 1974.

The LANGLVL specification does affect the operation of the FIPS flagger. When the FIPS LVL option has been specified, selecting the 1968 ANS standard will cause FIPS flagging to be done to those corresponding specifications; selecting the 1974 ANS standard causes FIPS flagging to be done according to the newer specifications.

SOURCE
NOSOURCE
indicates whether or not the source module is to be listed.

CLIST
NOCLIST
indicates whether or not a condensed listing is to be produced. If specified, the procedure portion of the listing will contain generated card numbers (unless the NUM option is in effect), verb references, and the location of the first instruction generated for each verb. Global tables, literal pools, register assignments, and information about the Working-Storage Section are also provided. CLIST and PMAP are mutually exclusive options. (If both are specified, COBOL rejects the one specified first.)

Note: In nonsegmented programs, verbs are listed in source order. In segmented programs, the root segment is first, followed by the individual segments in order of ascending priority.

DMAP
NODMAP
indicates whether or not a glossary is to be listed. Global tables, literal pools, register assignments, and information about the Working-Storage Section are also provided.

PMAP
NOPMAP
indicates whether or not register assignments, global tables, literal pools, information about the Working-Storage Section, and an assembler-language expansion of the source modules are to be listed. CLIST and PMAP are mutually exclusive options. (If both are specified, COBOL rejects the one specified first.)

Note: If any one of the options CLIST, DMAP, and PMAP is specified, the compiler will produce a message giving the hexadecimal length and starting address of the Working Storage Section. For an illustration of the use of these options, see the "Output" section.

VERB
NOVERB
indicates whether procedure-names and verb-names are to be listed with the associated code on the object-program listing. VERB has meaning only if PMAP or CLIST is in effect. VERB is automatically in effect if SYMDMP, STATE, or TEST is in effect. NOVERB yields more efficient compilation.

Note: If READY TRACE debugging statements are used in the program, VERB will cause paragraph-names rather than card numbers to be displayed at execution time.

LOAD
NOLOAD
indicates whether or not the object module is to be placed on a mass storage device or a tape volume so that the module can be used as input to the linkage editor. If the LOAD option is used, a SYSLIN DD statement must be specified.

DECK
NODECK
indicates whether or not the object module is to be punched. If the DECK option is used, a SYSPUNCH DD statement must be specified.

SEQ
NOSEQ
indicates whether or not the compiler is to check the sequence of the source module statements. If the statements are not in sequence, a message is printed.

Note: For examples of what the SOURCE, DMAP, PMAP, and SEQ options produce, see "Output."

LINECNT=nn
indicates the number of lines to be printed on each page of the compilation source card listing. The number specified by nn must be a 2-digit integer from 01 to 99. If the LINECNT option is omitted, 60 lines are printed on each page of the output listing.

Note: The compiler requires three lines of what the user has specified for its headings. (For example, if nn=55 is specified, then 52 lines are printed on each page of the output listing.)

ZWB
NOZWB

indicates whether or not the compiler generates code to strip the sign from a signed external decimal field when comparing this field to an alphanumeric field. If ZWB is specified, the signed external decimal field is moved to an intermediate field, in which its sign is removed, before it is compared to the alphanumeric field. ZWB complies with the ANS standard; NOZWB should be used when, for example, input numeric fields are to be compared with SPACES.

LVL=A/B/C/D
NOLVL

specifies what level of FIPS (Federal Information Processing Standard) flagging is to be used. A, B, C, and D correspond to the levels Low, Low-Intermediate, High-Intermediate, and Full, respectively. If flagging is specified, source clauses and statements that do not conform to the specified level of FIPS are identified. See the publication IBM VS COBOL for OS/VS for a complete list of the statements flagged at each level.

Notes:

1. If LVL is the SYSGEN default, its assigned value can be overridden at compile time with any level except NOLVL. If NOLVL is the SYSGEN default, it can be overridden at compile time with any level.

2. If the LVL option is in effect, the SYSUT6 data set must be specified.

3. If both LVL=A/B/C/D and TERM are specified, the compiler listing output to SYSPRINT for options such as SOURCE, PMAP, and XREF are not produced.

4. The option SOURCE is automatically turned on if LVL is specified. If TERM is off, a listing is produced. If TERM is on, there is no listing. (See note 3.)

5. If both LVL and BATCH are specified, LVL is rejected.

6. The LANGLVL option controls the version of FIPS that will be used. Requesting LANGLVL(1)--the 1968 ANS standard--causes flagging to be done according to the 1972 FIPS specifications. Requesting LANGLVL(2)--the 1974 ANS standard--causes flagging to be done according to the 1975 FIPS specifications.

7. Generally, FIPS flagging will not be done if the compiler has detected any source errors (that is, if the return code is not zero). The only exceptions concern a few W-level messages (such as IKF1100-W) that are purely informational; detecting these will not prevent FIPS flagging. The FLAGE option should not be used, since W-level messages will not be listed and results are unpredictable.

8. If the compiler detects a D-level error, FIPS will not execute, nor will any of the compiler's usual messages or listings be produced. A return code of 16 will be the only indication that this has occurred. To overcome this difficulty, recompile with a specification of NOLVL, then remove the source errors as indicated by the compiler, and finally recompile again with LVL.

FLAGW
FLAGE

indicates the type of messages that are to be listed for the compilation. FLAGW indicates that all warning and diagnostic messages are to be listed. FLAGE indicates that all error messages are to be listed, but that the warning messages are not to be listed. Note. With the use of FLAGE, and only W-level errors, the return code will be zero.

SUPMAP
NOSUPMAP

indicates whether or not the object code listing, and object module and link edit decks are to be suppressed if an E-level or D-level message is generated by the compiler.

SPACE1
SPACE2
SPACE3

indicates the type of spacing that is to be used on the source card listing generated when SOURCE is specified. SPACE1 specifies single spacing, SPACE2 specifies double spacing, and SPACE3 specifies triple spacing.

TRUNC
NOTRUNC
applies to movement of COMPUTATIONAL arithmetic fields. If TRUNC (standard truncation) is specified and the number of digits in the sending field is greater than the number of digits in the receiving field, the arithmetic item is truncated to the number of digits specified in the PICTURE clause of the receiving field when moved. If NOTRUNC is specified, movement of the item is dependent on the size of the field (halfword, fullword).

QUOTE
APOST
indicates to the compiler that either the double quote (") or the apostrophe (') is acceptable as the character to delineate literals and to use that character in the generation of figurative constants.

STATE
NOSTATE
indicates whether or not the number of the COBOL statement being executed at the time of an abnormal termination is desired. STATE identifies the number of the statement and the number of the verb being executed. If the STATE option is used, a SYSDBOUT DD statement must be specified at execution time for the output data set on which the statement number message can be written. For more information, see "Debugging Facilities" in the chapter "Program Checkout."

FLOW[=nn]
NOFLOW
indicates whether or not a formatted trace is desired for a variable number of procedures executed before an abnormal termination. The number of procedures traced is specified by nn, where nn may be any integer value from one to 99. FLOW[=nn] must be specified at compile time to generate the necessary trace linkage; however, specifying nn may be deferred until execution time. If nn is omitted, the default value is employed. This value is either 99 or that specified at program product installation. Specifying NOFLOW at compile time precludes specification of the Flow Trace option at execution time. A SYSDBOUT DD statement must be included for the output data set on which the trace can be written. See "Options for Execution" for more information.

SYMDMP
NOSYMDMP
requests a formatted dump of the data area of the object program at abnormal termination. With this option, the programmer may also request dynamic dumps of specified data-names at strategic points during program execution. See "Symbolic Debugging Features" for more information.

Notes:

1. If the SYMDMP option is in effect, the SYSUT5 data set must be specified, and the NULLFILE or DUMMY parameter should not be specified on its DD statement.

2. If the BATCH option is requested, SYMDMP is rejected.

3. If WITH DEBUGGING MODE is specified and one or more USE FOR DEBUGGING statements are in the program, the SYMDMP option is rejected.

4. Specification of the SYMDMP option automatically yields the OPTIMIZE feature, discussed below, and rejects the STATE option because SYMDMP output includes STATE output at abnormal termination.

For a discussion of the FLOW, STATE, and SYMDMP options, and their value to the COBOL programmer, see the chapter entitled "Symbolic Debugging Features." A SYSDBOUT, SYSDBG, and debug file DD codes are required at execution time.

OPTIMIZE
NOOPTIMIZE
causes optimized object code to be generated by the compiler, considerably reducing the use of object program main storage. In general, the greater the number of COBOL Procedure Division source statements, the greater the percentage of reduction in the amount of main storage required.

Note: The optimizer feature is automatically in effect when the SYMDMP or TEST feature is specified.

SYNTAX
CSYNTAX
NOSYNTAX
NOCSYNTAX

indicates whether object code generation is to be suppressed--that is, whether the compiler will only scan the source text for syntax errors (with appropriate error messages being produced).

SYNTAX causes syntax checking only, with absolute suppression of object code generation.

CSYNTAX causes syntax checking with conditional object code generation: a full compilation is produced as long as no errors exceed the W or C level; if one or more E-level or higher severity errors are discovered, the compiler does not generate the object code.

NOSYNTAX causes normal compilation, with both syntax checking and object code generation.

Notes:

1. When the SYNTAX option is in effect, all of the following compile-time options are suppressed:

LOAD	DECK	NAME
XREF	SYMDMP	COUNT
SXREF	TRUNC	VBSUM
CLIST	OPTIMIZE	VBREF
NOSUPMAP	FLOW	DMAP
PMAP	STATE	

2. Unconditional syntax checking is assumed if all of the following compile-time options are specified:

NOLOAD	NOCLIST	SUPMAP
NOXREF	NOPMAP	NODECK
NOSXREF		

3. CSYNTAX and SYNTAX are mutually exclusive. CSYNTAX will override SYNTAX.

4. If CSYNTAX is specified and an E or D-level diagnostic message is encountered before or during the operation of phase 21 or during the operation of phases 30, 40, 45, 50, or 51, the SYNTAX option replaces CSYNTAX, and the options listed in Note 1 above are suppressed. However, certain types of output may be produced (for example, a DMAP), depending on how much work the compiler had completed before the error was discovered.

5. When CSYNTAX has been specified and an E-level or higher error is detected, the final parts of the compilation procedure execution of phases 60, 62, 63, or 64 (which deal with code generation) are not executed. Because of this, any existing syntax errors that could only be detected by these latter phases of compilation will not be discovered and made known to the user during the CSYNTAX run; they will only come to light during a later full compilation.

NUM
NONUM

indicates whether or not line numbers have been recorded in the input and, rather than compiler-generated source numbers, should be used in error messages, as well as in PMAP, CLIST, STATE, XREF, SXREF, and FLOW. NONUM indicates that the compiler-generated numbers should be used in error messages as well as in PMAP, CLIST, STATE, XREF, SXREF, and FLOW.

Note: If when the NUM option is in effect the compiler discovers a non-numeric character in a line number or if ascending numeric sequence is broken, the compiler invalidates the number. The compiler then takes the last valid card number in sequence, adds a 1 to that number and begins generating card numbers from that point. The increment is 1. Six digits is the maximum sequence number. The card that follows 999999 will be flagged and NUM, SYMDMP, and TEST canceled. STATE and FLOW will not be canceled.

If LSTCOMP is in effect, the statement number generated by the lister feature is used regardless of NUM or NONUM specification.

XREF
NOXREF

indicates whether or not a cross-reference listing is produced. If XREF is specified, an unsorted listing is produced with data-names and procedure-names appearing in two parts in source order.

SXREF
NOSXREF
indicates whether or not a sorted cross-reference listing is produced. If SXREF is specified, a sorted listing is produced with data-names and procedure-names in alphanumeric order.

Notes:

- XREF and SXREF are mutually exclusive. If both are specified, COBOL rejects the one specified first.
- Some data names used in STRING, UNSTRING, SEARCH, and USE FOR DEBUGGING statements are not part of the compiler-generated code, and therefore will not appear in an XREF or SXREF listing.
- Group names in a MOVE CORRESPONDING statement will not be listed in an XREF or SXREF listing; however, the elementary names within those groups will be listed.
- Because most of the Report Writer code is generated before the compiler creates the dictionary, most of the Report Writer data names do not appear in an XREF or SXREF listing.

LIB
NOLIB
indicates whether or not a COPY and/or a BASIS request will be part of the COBOL source input stream. If no library facilities are to be used, the specification of NOLIB will save compilation time.

BATCH
NOBATCH
indicates whether or not multiple programs and/or subprograms are to be compiled with a single invocation of the compiler. In the BATCH environment all compiler options specified on the EXEC card, plus all default options, will apply to every program in the batch unless specific options are overridden on the CBL card, which must be included for each program. See "Batch Compilation" for more information on batch compilations and the CBL card.

When BATCH is specified, the LVL and SYMDMP options will be rejected. In addition, the BUF, L120/L132, and SIZE options are precluded from use on a

CBL card and will be rejected if specified.

NAME
NONAME
indicates whether or not programs in a batch compilation environment will be link-edited into one or more load modules. If NAME is specified, each succeeding program in the batch will be link-edited into a separate load module. This option will remain in effect for the entire compilation unless NONAME is specified on the CBL card for an individual program. If NONAME is specified on the CBL card, no name will be generated for this compilation. Names for the load modules will be formed according to the rules for forming module names from the PROGRAM-ID. See "Batch Compilation" for more details on batch compilation and the CBL card.

Note: If the BATCH option is not specified, NONAME will be in effect.

RESIDENT
NORESIDENT
requests the COBOL Library Management feature. When one program in a given region/partition requests the RESIDENT option, the main program and all subprograms in that region/partition should also request it.

Note: The RESIDENT option is automatically in effect when the DYNAM option is invoked.

DYNAM
NODYNAM
causes subprograms invoked through the CALL literal statement to be dynamically loaded and through the CANCEL statement to be dynamically deleted at object time (instead of link-edited with the calling program into a single load module).

Note: When both NORESIDENT and NODYNAM are either specified or implied by default, and a CALL identifier statement occurs in the source statement being compiled, the COBOL Library Management Facility option (RESIDENT) is automatically in effect. A printed statement of this is given in the compiler output. (For a discussion of the COBOL Library Management Facility, see the section "Sharing COBOL Library Subroutines" in the "Libraries" chapter.)

SYST
SYSx
indicates whether SYSOUT or SYSOUx, where x must be alphanumeric (that is, 0-9 or A-Z except for T), is the ddname of the file to be used for debug output (READY TRACE, EXHIBIT) or DISPLAY statement. The specification in the program that is first to access the file is chosen.

ENDJOB
NOENDJOB
indicates whether or not, at the end of each run-unit (which is assumed to begin with the highest-level COBOL program called), the COBOL library subroutine ILBOSTT0 is to be called to delete modules and free main storage acquired through GETMAINs issued by COBOL library subroutines. ENDJOB takes effect either at a STOP RUN in any program, or at a GOBACK statement in a main program only. Violation of the restriction against mixing RES and NORES modules within a run-unit may cause an unpredictable execution-time abend when ENDJOB is in effect, even in programs which ran successfully without ENDJOB.

Note: When a non-COBOL program, such as IMS or an installation-defined assembler program, links to COBOL load modules many times within a job step, the resulting accumulation of GETMAIN-acquired areas and loaded modules may result in execution-time abends. In order to prevent fragmentation and overload of storage in such an environment, the ENDJOB compiler option must be specified. This will cause ILBOSTT0 to be loaded at the normal termination of the run-unit to free all GETMAIN areas and, in a RES environment, to delete any loaded subroutines and dynamically-invoked subprograms. The only GETMAIN areas not freed by ILBOSTT0 are those obtained when opening a random indexed BISAM file for which the options APPLY CORE-INDEX and/or TRACK-AREA IS integer CHARACTERS have been specified; before terminating, the user should close such files within the COBOL program.

Since ILBOSTT0 is always loaded, it must be made available at execution time (by placing it in the link pack area or by specifying the COBOL library on a STEPLIB DD statement for the GO step). If ILBOSTT0 is not placed in the link pack area, it should be explicitly deleted by the invoker of the COBOL run-unit; otherwise, one copy of this subroutine will remain in the user region after the run-unit has completed.

ADV
NOADV

indicates whether or not records for files with WRITE...ADVANCING need reserve the first byte for the control character. ADV specifies that the first byte need not be reserved, but that the compiler will add one byte to the LRECL for the control character.

COUNT
NOCOUNT

indicates whether or not code is to be generated to produce verb execution summaries at the end of problem program execution. Each verb is identified by procedure-name and by statement number, and the number of times it was used is indicated. In addition, the percentage of verb execution for each verb with respect to the execution of all verbs is given. A summary of all executable verbs used in a program and the number of times they are executed is provided. COUNT implies VERB. COUNT requires both SYSDBOUT and SYSCOUNT DD statements at execution time. For a more detailed discussion on the use of the COUNT option, see the chapter "Program Checkout."

DUMP
NODUMP

specifies whether the compiler should produce a dump or an informative message in the event it encounters a D-level ("disaster") error condition during its processing. The compiler will abnormally terminate after producing the dump/message.

DUMP specifies that a dump (but no message) is to be produced. This dump will contain a four-digit user completion code. See Appendix K for more information on these codes. (Note: The most frequent cause of compiler abend is insufficient SIZE value--a user completion code of 0003. In this case, rerun the program specifying a larger value.)

NODUMP specifies that the compiler is to produce an informative message (but no dump).

If analysis of the message or dump does not solve the problem, see Appendix L for the procedure to follow in calling IBM for assistance.

VBSUM
NOVBSUM

provides a brief summary of verbs used in the source program and a count of how often each verb appeared. This option provides the user with a quick search for specific types of statements. VBSUM implies VERB.

VBREF
NOVBREF

provides a cross-reference of all verbs used in the program. This option provides the programmer with a quick index to any verb used in the program. VBREF implies VERB and VBSUM.

Options for the Lister Feature

There are five compiler options for using the lister feature of the compiler. Note that either LSTONLY or LSTCOMP must be selected for the other four options to have meaning unless the BATCH option is specified. In a batch compilation, if some or all of the programs are to be compiled using the lister feature, L120 or L132 must be specified in the PARM field of the EXEC card, even if LSTCOMP or LSTONLY are to be specified on the CBL card.

For detailed information on the use of the lister, see the chapter "The Lister Feature." The options are listed in Figure 6, where:

LSTONLY
LSTCOMP
NOLST

indicates whether the lister feature is to be used. LSTONLY specifies that a reformatted listing is to be produced but that no compilation is to occur. LSTCOMP specifies that both a reformatted listing is to be produced and compilation is to occur in the same job step.

FDECK
NOFDECK

indicates whether a copy of the reformatted source program is to be written on the SYSPUNCH data set. Since FDECK has meaning only with either LSTONLY or LSTCOMP, the lister output will be both a reformatted listing and a reformatted deck.

CDECK
NOCDECK

indicates whether or not COPY statements are to be expanded into COPY members in the SYSPUNCH output.

The COPY members are to be expanded in the reformatted deck requested through FDECK. If CDECK is specified with NOFDECK, only the expanded COPY statements are produced.

LCOL1
LCOL2

indicates whether the Procedure Division part of the listing is to be in single or double column format.

L120
L132
: indicates whether the length of each line of the reformatted listing is to be 120 or 132 characters long.

Options for Use Under TSO Only

In addition to the preceding compiler options, the following options are designed for use with the Time Sharing Option (TSO). Time Sharing provides the COBOL programmer with facilities for entering, compiling, and testing programs at his terminal. (For further information on the Time Sharing Option, see the Program Product publication IBM OS (TSO): COBOL Prompter Terminal User's Guide and Reference.) These options are listed in Figure 8, where:

```
Compiler:

{PARM          }
{PARM.procstep} =([ SIZE=yyyyyyy ][ ,BUF=yyyyyy ][ ,LANGLVL(1/2) ] [,SOURCE  ] [,DMAP  ]
                                                                   [,NOSOURCE] [,NODMAP]

[,PMAP  ] [,SUPMAP  ] [,LOAD  ] [,DECK  ] [,SEQ  ]
[,NOPMAP] [,NOSUPMAP] [,NOLOAD] [,NODECK] [,NOSEQ]

[,LINECNT=nn ] [,TRUNC  ] [,CLIST  ] [,FLAGW] [,QUOTE]
               [,NOTRUNC] [,NOCLIST] [,FLAGE] [,APOST]

[,SPACE1] [,STATE  ] [,XREF  ] [,SXREF  ] [,NAME  ]
[,SPACE2] [,NOSTATE] [,NOXREF] [,NOSXREF] [,NONAME]
[,SPACE3]

[,BATCH  ] [,FLOW[=nn]] [,TERM  [4]] [,PRINT {(*)[5]} [4]]
[,NOBATCH] [,NOFLOW   ] [,NOTERM  ]  [,NOPRINT {(dsname)}  ]

[,SYMDMP  ] [,OPTIMIZE  ] [,SYNTAX  ] [,LVL=A/B/C/D]
[,NOSYMDMP] [,NOOPTIMIZE] [,NOSYNTAX]

[,TEST  [4]] [,ENDJOB  ] [,CSYNTAX  ] [,LIB  ] [,NUM  ]
[,NOTEST   ] [,NOENDJOB] [,NOCSYNTAX] [,NOLIB] [,NONUM]

[,RESIDENT  ] [,DYNAM  ] [,VERB  ] [,ZWB  ] [,SYST]
[,NORESIDENT] [,NODYNAM] [,NOVERB] [,NOZWB] [,SYSx]

[,ADV  ] [,COUNT  ] [,DUMP  ] [,LSTONLY [6]] [,LCOL1 [6]]
[,NOADV] [,NOCOUNT] [,NODUMP] [,LSTCOMP   ] [,LCOL2   ]
                              [,NOLST     ]

[,FDECK  [6]] [,CDECK  [6]] [,L132  [6]] [,VBSUM  ] [,VBREF  ] ) [123]
[,NOFDECK   ] [,NOCDECK   ] [,L120     ] [,NOVBSUM] [,NOVBREF]

Linkage Editor:

{PARM          }
{PARM.procstep} = ( [{MAP }]          [,LIST] [,OVLY])
                    {XREF}

Loader:

{PARM          }
{PARM.procstep} = ( [MAP  ] [,RES  ] [,CALL  ] [,LET  ] [,SIZE=100K]
                  [NOMAP] [,NORES] [,NOCALL] [,NOLET] [,SIZE=size]

                    [,EP=name ] [,PRINT  ]          ) [12]
                                [,NOPRINT]

Execution:

{PARM          }
{PARM.procstep} =([user parameters]/ [FLOW[=nn]] [,DEBUG  ] [,UPSI(nnnnnnnn)]
                                     [NOFLOW  ] [,NODEBUG]

                 [,AIXBLD  ]
                 [,NOAIXBLD]  [,QUEUE(value-list) ]) [123]

[1]If the information specified contains any special characters, it must be delimited by
 single quotation marks instead of parentheses.
[2]If the only special character contained in the value is a comma, the value may be
 enclosed in parentheses or quotation marks.
[3]The maximum number of characters allowed between the delimiting quotation marks or
 parentheses is 100.
[4]These options should be used in the Time Sharing environment only.
[5]TSO-only format.
[6]These options are used to request the lister feature.
```

Figure 8. Compiler, Linkage Editor, and Loader PARM Options

PRINT $\left\{\begin{matrix}(*)\\(dsname)\end{matrix}\right\}$

NOPRINT

indicates whether or not the program listing is to be suppressed, placed on the output data set specified by dsname, or displayed at the terminal. If PRINT is specified, the listing will include page headings, line numbers of the statements in error, message identification numbers, severity levels, and message texts (as well as any other output requested by SOURCE, CLIST, DMAP, PMAP, XREF, or SXREF). If (*) is specified instead of data-set name, the printed output is sent to the terminal. If PRINT alone is specified, a listing data set is created on secondary storage and named according to standard data set naming conventions. NOPRINT specifies that no listing is to be printed. If neither PRINT nor NOPRINT is specifiedand any one or more of the options SOURCE, CLIST, DMAP, XREF, or PMAP are specified, PRINT is the default. Otherwise, NOPRINT is the default. If PRINT is specified in a non-TSO environment, it is ignored.

TERM

NOTERM

indicates whether or not progress and diagnostic messages are to be printed on the SYSTERM terminal data set. The severity level of the messages may be controlled by the FLAG option. If PRINT (*) is specified, then NOTERM is the default, to ensure that messages appear only once. If TERM is specified in a non-TSO environment, the output that normally goes to the SYSTERM DD data set is written on the SYSTERM file if a SYSTERM DD card has been included. If there is no SYSTERM DD card, a warning message is issued.

TEST

NOTEST

indicates whether or not the program can be debugged at the terminal using the program product IBM OS COBOL Interactive Debug (Program Number 5734-CB4). A program that is compiled without the TEST option is unacceptable to the Interactive Debug command processor. Complete information on COBOL Interactive Debug is contained in IBM OS COBOL Interactive Debug Terminal User's Guide and Reference.

When TEST is in effect, the COBOL compiler produces optimized object code. When you specify TEST, you cannot also specify FLOW, STATE, COUNT, SYMDMP, or BATCH.

If the TEST option is in effect, the SYSUT5 data set must be specified.

The TEST option is rejected if both WITH DEBUGGING MODE is specified and one or more USE FOR DEBUGGING statements are in the program.

Options for the Linkage Editor

MAP

indicates that a map of the load module is to be listed. If MAP is specified, XREF cannot be specified, but both can be omitted.

XREF

indicates that a cross-reference list and a module map are to be listed. If XREF is specified, MAP cannot be specified.

LIST

indicates that any linkage editor control statements associated with the job step are to be listed.

OVLY

indicates that the load module is to be in the format of an overlay structure. This option is required when the COBOL Segmentation feature is used.

The format of the PARM parameter is illustrated in Figure 8. For examples of what the MAP, XREF, and LIST options produce, see "Output." Linkage editor control statements and overlay structures are explained in "Calling and Called Programs." There are other PARM options for linkage editor processing that describe additional processing options and special attributes of the load module (see the publication OS/VS Linkage Editor and Loader).

Options for the Loader

MAP

NOMAP

indicates whether or not a map of the loaded module is to be produced that lists external names and their absolute addresses on the SYSLOUT data set. If the SYSLOUT DD statement is not used in the input deck, this option is ignored. An example of a module map is shown in "Output."

RES
NORES
indicates whether or not an automatic search of the link pack area queue is to be made. This search is always made after processing the primary input (SYSLIN), and before searching the SYLIB data set. When the RES option is specified, the CALL option is automatically set.

CALL
NOCALL (NCAL)
indicates whether or not an automatic search of the SYSLIB data set is to be made. If the SYSLIB DD statement is not used in the input deck, this option is ignored. The NOCALL option causes an automatic NORES.

LET
NOLET
indicates whether or not the loader will try to execute the object program when a severity level 2 error condition is found.

SIZE=100K
SIZE=size
specifies the size, in bytes, of dynamic main storage that can be used by the loader. This storage must be large enough to accommodate the object program.

EP=name
specifies the external name to be assigned as the entry point of the loaded program.

PRINT
NOPRINT
indicates whether or not diagnostic messages are to be produced on the SYSLOUT data set.

The format of the PARM parameter is illustrated in Figure 8. The default options, indicated by an underscore, can be changed at system generation with the LOADER macro instruction.

Options for Execution

These options are specified through the PARM parameter, as illustrated in Figure 8. Note that a slash must immediately precede the first COBOL-defined option coded. (If user parameters themselves are to include a slash or slashes, then an additional, trailing slash must be added to demark the user parameters' end. All data following the last slash is considered to be COBOL system parameter information, and will not be passed to the program.)

If an execution-time parameter field is passed to a program, a load of the library subroutine ILBOPRM will be issued. If this subroutine is not available to the loader, an 806 abend may occur. One way to circumvent this problem is to place an INCLUDE SYSLIB(ILBOPRM) statement in the link edit SYSLIN data stream, which would link this subroutine into the load data set.

User Parameters
The programmer can code any parameters he wishes to pass to a main COBOL program. For information on how to access such parameters, see the "USING option" as described in IBM VS COBOL for OS/VS.

FLOW[=nn]
NOFLOW
If the FLOW option is specified at compile time for a trace of procedure names, at execution time a value for nn may be specified that overrides any value set at compile time. If FLOW is requested at compile time with no value for nn, a value should be specified at execution time. A default of 99 is assumed for nn if it is not specified at either step and FLOW is in effect; otherwise, nn is as previously specified.

The FLOW trace may be suppressed at execution time by specifying NOFLOW. FLOW cannot be specified as an option for execution if it is not specified at compile time or if NOFLOW is in effect by default. See the sections "Debugging Facilities" and "Options for the Compiler" for additional information.

DEBUG
NODEBUG
DEBUG indicates that USE FOR DEBUGGING declarative procedures are in the program and should in fact be activated during this execution. NODEBUG indicates that even though such declarative procedures were included in the program, they are not desired and their execution is to be suppressed. (Note that this DEBUG switch has meaning only if the source program was compiled with the WITH DEBUGGING MODE clause.)

UPSI(nnnnnnnn)
assigns values (either zero or one) to the eight switches UPSI-0, UPSI-1, UPSI-2, ... UPSI-7. (The default values are zeros.) These values are then available in the COBOL program through the condition names associated with them in the SPECIAL-NAMES paragraph.

AIXBLD
NOAIXBLD
: for VSAM KSDS and RRDS data sets, AIXBLD indicates that COBOL should invoke Access Method Services to complete the file and index definition procedures. NOAIXBLD indicates that the user has already performed such definitions himself ahead of time, and has no need for this service.

In general, better performance is obtained if the user provides such definitions himself (NOAIXBLD). If AIXBLD is specified, substantial amounts of additional storage are required for COBOL execution (the exact amount depending on specific

system configuration). Also, a SYSPRINT DD card is necessary for any Access Method Services messages that may be produced.

For more detail, see "Dynamic Invocation of Access Method Services for KSDS and RRDS Output Data Sets" in the section "VSAM File Processing."

QUEUE(value-list)
specifies a queue-name structure (in value-list) which is the structure that will cause this program to be scheduled for execution. For more detail, see "Communications Job Scheduling" in the section "Using the Communications Feature."

Requesting Restart for a Job Step (RD)

The restart facilities can be used in order to minimize the time lost in reprocessing a job that abnormally terminates. These facilities permit the automatic restart of jobs that were abnormally terminated during execution.

The programmer uses this parameter to tell the operating system: (1) whether or not to take checkpoints during execution of a program, and (2) whether or not to restart a program that has been interrupted.

A checkpoint is taken by periodically recording the contents of storage and registers during execution of a program. The RERUN clause in the COBOL language facilitates taking checkpoint readings. Checkpoints are recorded onto a checkpoint data set.

Execution of a job can be automatically restarted at the beginning of a job step that abnormally terminated (step restart) or within the step (checkpoint restart). In order for checkpoint restart to occur, a checkpoint must have been taken in the processing program prior to abnormal termination. The RD parameter specifies that step restart can occur or that the action of the CHKPT macro instruction is to be suppressed.

To request that step restart be permitted or to request that the action of the CHKPT macro instruction be suppressed in a particular step, code the keyword parameter in the operand field of the EXEC statement.

```
RD=request
```

Replace the word "request" with:

R -- to permit automatic step restart. The programmer must specify at least one RERUN clause in order to take checkpoints.

NC -- to suppress the action of the CHKPT macro instruction and to prevent automatic restart. No checkpoints are taken; no RERUN clause in the COBOL program is necessary.

NR -- to request that the CHKPT macro instruction be allowed to establish a checkpoint, but to prevent automatic restart. The programmer must specify at least one RERUN clause in order to take checkpoints.

RNC -- to permit step restart and to suppress the action of the CHKPT macro instruction. No checkpoints are taken; no RERUN clause in the COBOL program is necessary.

Each request is described in greater detail in the following paragraphs.

RD=R: If the processing programs used by this step do not include a RERUN statement, RD=R allows execution to be resumed at the beginning of this step if it abnormally terminates. If any of these programs do include one or more CHKPT macro instructions (through the use of the RERUN clause), step restart can occur if this step abnormally terminates before execution of a CHKPT macro instruction; thereafter, checkpoint restart can occur.

RD=NC or RD=RNC: RD=NC or RD=RNC should be specified to suppress the action of all CHKPT macro instructions included in the programs used by this step. When RD=NC is specified, neither step restart nor checkpoint restart can occur. When RD=RNC is specified, step restart can occur.

RD=NR: RD=NR permits a CHKPT macro instruction to establish a checkpoint, but does not permit automatic restarts. However, a resubmitted job could have execution start at a specific checkpoint.

Before automatic step restart occurs, all data sets in the restart step with a status of OLD or MOD, and all data sets being passed to steps following the restart step, are kept. All data sets in the restart step with a status of NEW are

deleted. Before automatic checkpoint restart occurs, all data sets currently in use by the job are kept.

If the RD parameter is omitted and no CHKPT macro instructions are executed, automatic restart cannot occur. If the RD parameter is omitted but one or more CHKPT macro instructions are executed, automatic checkpoint restart can occur.

Notes:

- If the RD parameter is specified on the JOB statement, RD parameters on the job's EXEC statements are ignored.
- Restart can occur only if MSGLEVEL=1 is coded on the JOB statement.
- If step restart is requested for this step, assign the step a unique step name.
- When this job step uses a cataloged procedure, make restart request for a single procedure step by including, as part of the RD parameter, the procedure stepname, i.e., RD.procstepname. This specification overrides the RD parameter in the named procedure step if one is present. Code as many parameters of this form as there are steps in the cataloged procedure.
- To specify a restart request for an entire cataloged procedure, code the RD parameter without a procedure stepname. This specification overrides all RD parameters in the procedure if any are present.
- If no RERUN clause is specified in the user's program, no checkpoints are written, regardless of the disposition of the RD parameter.

Reference:

- For detailed information on the checkpoint/restart facilities, see the publication OS/VS Checkpoint/Restart.

Priority Scheduling EXEC Parameters

Establishing a Dispatching Priority (DPRTY) (OS/VS2 only)

The DPRTY parameter allows the programmer to assign to a job step, a dispatching priority different from the priority of the job. The dispatching priority determines in what sequence tasks use main storage and computing time. To assign a dispatching priority to a job step, code the keyword parameter in the operand field of the EXEC statement.

```
DPRTY=(value 1,value 2)
```

Both "value 1" and "value 2" should be replaced with a number from 0 through 15. "Value 1" represents an internal priority value. "Value 2" added to "value 1" represents the dispatching priority. The higher numbers represent higher priorities. A default value of 0 is assumed if no number is assigned to "value 1." A default value of 11 is assumed if no number is assigned to "value 2."

Notes:

- Whenever possible, avoid assigning a number of 15 to "value 1." This number is used for certain system tasks.
- If "value 1" is omitted, the comma must be coded before "value 2" to indicate the absence of "value 1," e.g., DPRTY=(,14).
- If "value 2" is omitted, the parentheses need not be coded, e.g., DPRTY=12.
- When the step uses a cataloged procedure, a dispatching priority can be assigned to a single procedure step by including the procedure step name in the DPRTY parameter, i.e., DPRTY.procstepname=(value 1, value 2). This parameter may be used for each step in the cataloged procedure.
- To assign a single dispatching priority to an entire cataloged procedure, code the DPRTY parameter without a procedure step name. This specification overrides all DPRTY parameters in the procedure if there are any.

Setting Job Step Time Limits (TIME)

To assign a limit to the computing time used by a single job step, a cataloged procedure, or a cataloged procedure step, code the keyword parameter in the operand field of the EXEC statement.

```
TIME=(minutes,seconds)
```

Such an assignment is useful in a multiprogramming environment where more than one job has access to the computing

system. Minutes and seconds represent the maximum number of minutes and seconds allotted for execution of the job step.

Notes:

- If the job step requires use of the system for 24 hours (1440 minutes) or longer, the programmer should specify. TIME=1440. Using this number suppresses timing. The number of seconds cannot exceed 59.
- If the time limit is given in minutes only, the parentheses need not be coded; e.g., TIME=5.
- If the time limit is given in seconds, the comma must be coded to indicate the absence of minutes; e.g., TIME=(,45).
- When the job step uses a cataloged procedure, a time limit for a single procedure step can be set by qualifying the keyword TIME with the procedure step name; i.e., TIME.procstep=(minutes,seconds). This specification overrides the TIME parameter in the named procedure step if one is present. As many parameters of this form can be coded as there are steps in the cataloged procedure.
- To set a time limit for an entire procedure, the TIME keyword is left unqualified. This specification overrides all TIME parameters in the procedure if any are present.
- If this parameter is omitted, the standard job step time limit is assigned.

Specifying Main Storage Requirements for a Job Step (REGION)

The REGION parameter permits the programmer to specify the size of the main storage region to be allocated to the associated job step. The REGION parameter specifies the maximum amount of main storage to be allocated to the job. This amount must include the size of those components required by the user's program that are not resident in main storage.

The REGION parameter is used in conjunction with the ADDRSPC parameter to determine the total amount of main storage available to a program and to either allow or disallow paging.

Note: The REGION parameter has different meanings for OS/VS1 and OS/VS2. See the publication OS/VS JCL Services for detailed information.

To specify a region size, code the keyword parameter in the operand field of the EXEC statement.

```
REGION=(nnnnnK)
```

To request the maximum amount of main storage required by the job, replace the term "nnnnn" with the maximum number of contiguous 1024-byte areas allocated to the job step, e.g., REGION=52K. This number can range from 1 to 5 digits but must not exceed 16383.

If the REGION parameter is omitted or if a region size smaller than the default region size is requested, it is assumed that the default value is that established by the input reader procedure.

Notes:

- Region sizes for each job step can be coded by specifying the REGION parameter in the EXEC statement for each job step. However, if a REGION parameter is present in the JOB statement, it overrides REGION parameters in EXEC statements.
- For information on storage requirements to be considered when specifying a region size, see the appropriate Storage Estimates publication.

Specifying Address Space (ADDRSPC)

To take advantage of the storage facilities offered by OS/VS1 and OS/VS2, always specify ADDRSPC=VIRT.

```
ADDRSPC= {VIRT}
         {REAL}
```

Note that the compiler and its object code, including library subroutines, can run VIRTUAL, and should be run that way unless a non-COBOL program in the partition requires the REAL option.

DD STATEMENT

The data definition (DD) statement identifies each data set that is to be used in a job step, and it furnishes information about the data set. The DD statement specifies input/output facilities required

for using the data set; it also establishes a logical relationship between the data set and input/output references in the program named in the EXEC statement for the job step.

Figure 9 is a general format of the DD statement.

Parameters used most frequently for COBOL programs are discussed in detail. The other parameters (e.g., SEP and AFF) are mentioned briefly. For further information, see the publication OS/VS JCL Reference.

Name	Operation	Operand
// {ddname / procstep.ddname} [1]	DD	(see below and next page)

Operand[2]

Positional Parameters[3]

[{* / DATA} [,DLM=xx]] [DUMMY] [DYNAM][4]

Keyword Parameters [5] [6]

[DDNAME=ddname]

[{DSNAME / DSN} = {dsname / dsname(element) / *.ddname / *.stepname.ddname / *.stepname.procstep.ddname / &&name / &&name(element) / NULLFILE}] [12]

[QNAME=processname][14]

[DCB=(list of attributes)]

[DCB=({dsname / *.ddname / *.stepname.ddname / *.stepname.procstep.ddname} [,subparameter-list])] [7]

[SEP=(subparameter list)[8] / AFF=ddname] [11]

[COPIES=nnn]

[OUTLIM=number]

[TERM={RT / TS}]

Positional Subparameters Keyword Subparameters

[UNIT=(name[,[n/P][,DEFER]][,SEP=(list of up to 8 ddnames))[9] / UNIT=(AFF=ddname)] [11] [13]

[UCS=(character set code [,FOLD [,VERIFY]])]

See notes at end of figure.

Figure 9. The DD Statement (Part 1 of 3)

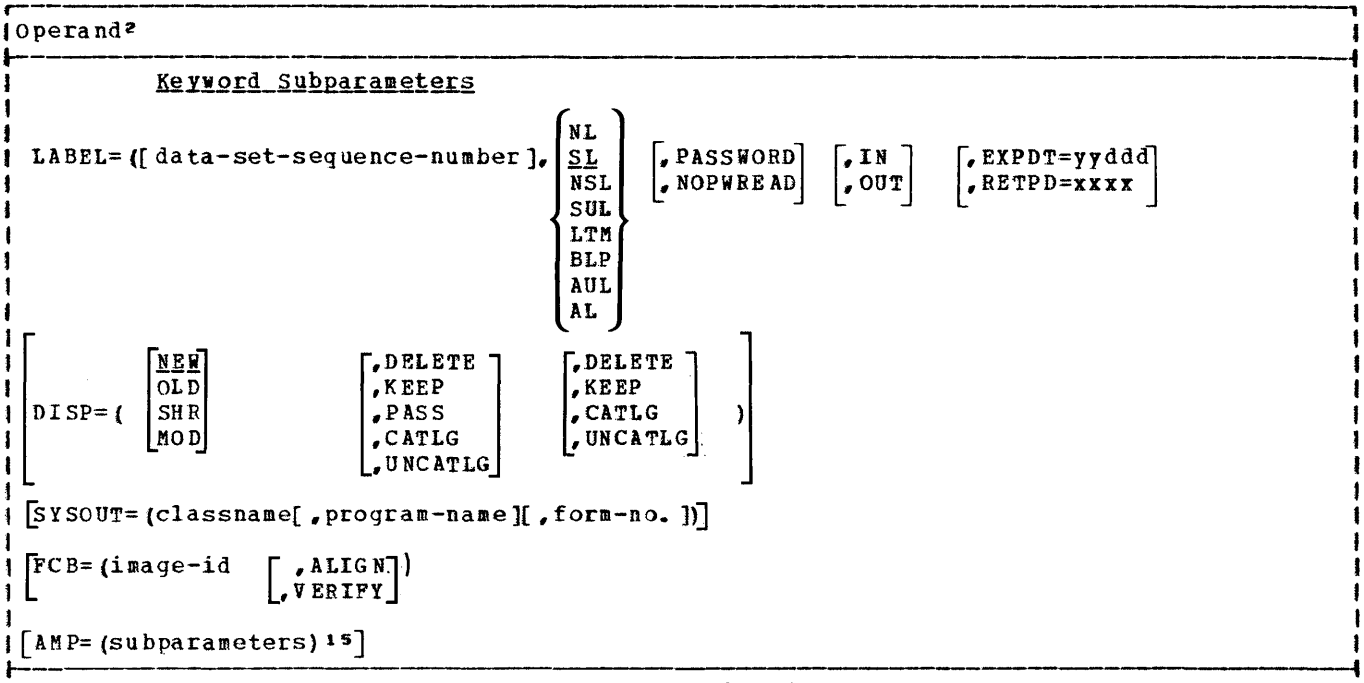
Operand[2]

Positional Subparameters

```
SPACE=( {TRK                                  }      ,(primary-quantity[,secondary-quantity],
        {CYL                                  }
        {average-record-length                }

                    [directory- or index-quantity])[,RLSE]  [,MXLG ]  [, ROUND ])
                                                            [,ALX  ]
                                                            [,CONTIG]

SPACE=(ABSTR,(quantity,beginning-address[,directory- or index-quantity]))

SPLIT=(n,  {CYL                               }  ,(primary-quantity[,secondary-quantity]))
           {average-record-length             }

SUBALLOC=(  {TRK                              }  ,(primary-quantity[,secondary-quantity]
            {CYL                              }
            {average-record-length            }

             [,directory-quantity]),  {ddname                    }  )
                                      {stepname.ddname           }
                                      {stepname.procstep.ddname  }

[{VOLUME}                                                                              [ ]]
[{VOL   } =([PRIVATE],[RETAIN],[volume sequence number],[volume count][,]             [ ]]
 SER=(volume-serial-number[volume-serial-number][10...])                               [ ]]
                                                                                       [ ]]
 REF=  {dsname                         }                                               [ ]]
       {*.ddname                       }                                               [ ]]
       {*.stepname.ddname              }                                               [ ]]
       {*.stepname.procstep.ddname     }                                               [ ]]
```

See notes at end of figure

Figure 9. The DD Statement (Part 2 of 3)



[1]The name field must be blank when concatenating data sets.
[2]All parameters are optional to allow a programmer flexibility in the use of the DD statement; however, a DD statement with a blank operand field is meaningless.
[3]If the positional parameter is specified, keyword parameters other than DCB cannot be specified.
[4]For OS/VS2 only.
[5]If subparameter-list consists of only one subparameter and no leading comma (indicating the omission of a positional subparameter) is required, the delimiting parentheses may be omitted.
[6]If subparameter-list is omitted, the entire parameter must be omitted.
[7]See "User-Defined Files" for the applicable subparameters.
[8]See the publication OS/VS JCL Reference.
[9]If only name is specified, the delimiting parentheses may be omitted.
[10]If only one volume-serial-number is specified, the delimiting parentheses may be omitted.
[11]The SEP and AFF parameters should not be confused with the SEP and AFF subparameters of the UNIT parameter.
[12]The value specified may contain special characters if the value is enclosed in apostrophes. If the only special character used is the hyphen, the value need not be enclosed in apostrophes. If DSNAME is a qualified name, it may contain periods without being enclosed in apostrophes.
[13]The unit address may contain a slash, and the unit type number may contain a hyphen, without being enclosed in apostrophes, e.g., UNIT=293/5,UNIT=2400-2.
[14]The QNAME= parameter is used in COBOL teleprocessing and must be the name of a TCAM destination queue.
[15]This parameter is for use with VSAM only. The subparameters are described in the chapter "VSAM File Processing."

Figure 9. The DD Statement (Part 3 of 3)

Name Field
ddname (Identifying the DD Statement)
is used:

- To identify data sets defined by this DD statement to the compiler or linkage editor (see "Compiler Data Set Requirements" and "Linkage Editor Data Set Requirements").
- To relate the data sets defined in this DD statement to a file described in a COBOL source program (see "User-Defined Files").
- To identify this DD statement to other control statements in the input stream.

procstep.ddname
is used to alter or add DD statements in cataloged procedures. The step in the cataloged procedure is identified by procstep. The ddname identifies either one of the following:

- A DD statement in the cataloged procedure that is to be modified by the DD statement in the input stream.
- A DD statement that is to be added to the DD statement in the procedure step.

* Parameter (Defining Data in an Input Stream)
indicates that data immediately follows this DD statement in the input stream. This parameter is used to specify a source deck or data in the input stream. The data cannot contain // or /* in the first two character positions of any record unless the DLM parameter is used. If (while scanning JCL) the system encounters a card that does not begin with //, it logically inserts a //SYSIN DD * card before it. The SYSIN data set thus created is delimited by the next /* or // job control card encountered.

DATA Parameter (Defining Data in an Input Stream)
also indicates a JCL deck or data in the input stream. The end of the data set must be indicated by a delimiter statement. The data cannot contain /* in the first two characters of any record unless the DLM parameter is used. The DD DATA statement must be the last DD statement of the job step. // may appear in the first and second positions in the record, for example, when the data consists of control statements of a procedure that is to be cataloged.

DLM Parameter (Changing Data Delimiter for Input Stream)
specifies the delimiter to be used instead of /* or // to terminate data defined in the input stream. Assigning a different delimiter allows the standard delimiter (/* or //) to be used as data in the input stream. The DLM parameter has meaning only on DD* and DD DATA statements. The data must be terminated with the characters assigned in the DLM parameter.

DUMMY Parameter (Bypassing Device Allocation and Input Operations on the Data set)
allows a program to attempt I/O operations on a data set during execution without the operating system's performing actual operations on the data set. (Programs compiled with the LANGLVL(2) option can specify the COBOL source statement SELECT OPTIONAL to obtain this ability and thus avoid supplying a dummy DD statement.) The DUMMY parameter is valid only for COBOL input sequential data sets. No device allocation, external storage allocation, or cataloging takes place for dummy data sets. When the DUMMY parameter is specified, a read request to an "opened" file results in an end of data set exit.

For a VSAM data set, if DUMMY is specified, an attempt to read results in an end-of-data condition. AMP='AMORG' must be specified if DUMMY is specified; see "VSAM-only JCL Parameter" in the chapter "VSAM File Processing."

Note: Compiler work files (including SYSUT5 and SYSUT6) cannot be described as DD DUMMY.

Data in the input stream is temporarily transferred to a direct-access device for later high-speed retrieval. Normally, the system stores it in a format that is not under control of the programmer. However, in some situations the programmer may be able to assign his own values through use of the BLKSIZE subparameter of the DCB parameter. He may also indicate the number of buffers to be assigned to transmitting the data, through use of the BUFNO parameter. For example, he may assign the following:

```
DCB=(BLKSIZE=800,BUFNO=2)
```

In VS1 and VS2 Release 2 and later, maximum performance is obtained for SYSIN and SYSOUT data sets when BLKSIZE=80 and BUFNO=1 are used. If not supplied by the

problem program, it can be supplied by the DCB parameter on the DD statement. If omitted from both sources, the defaults are BLKSIZE=80 and BUFNO=2.

DYNAM Parameter (Reserving Space under OS/VS2 with TSO)
specifies that space is to be reserved in internal tables so that data set requirements that arise during a TSO terminal session can be satisfied. This allows deferred definition of a data set until it is required. During LOGON processing for TSO, no devices or external storage are allocated to a data set defined by a DD DYNAM statement. When a data set is required, the actual device and external storage for the data set is then allocated. See the publication OS/VS2 TSO Guide, Order No. GC28-0644, for further information.

Note: No other parameter can be specified on a DD statement where DYNAM is specified.

DDNAME Parameter (Postponing the Definition of a Data Set)
defines a pseudo data set that will assume the characteristics of a real data set if a subsequent DD statement of the step is labeled with the specified ddname. When the DDNAME parameter is specified, it must be the first parameter in the operand. All other parameters, except the DCB subparameters BLKSIZE, BUFNO, and DIAGNS, are ignored and should be omitted when the DDNAME parameter appears (see "Using the Cataloged Procedures").

The ddname specifies a DD statement that, if present, supplies the attributes of the data set. If it is not present, the statement is ignored.

DSNAME Parameter (Identifying the Data Set)
allows the programmer to specify the name of the data set to be created or to refer to a previously created data set. Various types of names can be specified (see "Using the DD Statement" for a discussion of the various names) as follows:

- Fully qualified names: For data sets to be retrieved from or stored in the system catalog.
- Generation data group names: For an entire generation data group, or any single generation thereof.
- Simple names: For data sets that are not cataloged.
- Reference names: For data sets whose names are given in the DSNAME parameter of another DD statement in the same job.
- Temporary names: For temporary data sets that are to be named for the duration of one job only.

If the DSNAME parameter is omitted, the operating system assigns a unique name to the data set. (This parameter should be supplied for all except temporary data sets to allow future referencing of the data set.) DSNAME may be coded DSN.

DSNAME Subparameters
dsname
specifies the fully qualified name of a data set. This is the name under which the data set can be cataloged or otherwise identified on the volume.

dsname(element)
specifies a particular generation of a generated data group, a member of a partitioned data set, or an area of an indexed data set. To indicate a generation of a generated data group, the element is a zero or a signed integer. To indicate a member of a partitioned data set, the element is a name. To indicate an area of an indexed data set, the element is PRIME, OVFLOW, or INDEX (see "Using the DD Statement" for information about generation data groups and examples of partitioned data sets).

*.ddname
indicates that the DSNAME parameter (only) is to be copied from a preceding DD statement in the current job step.

*.stepname.ddname
indicates that the DSNAME parameter (only) is to be copied from the DD statement, ddname, that occurred in a previous step, stepname, in the current job. If this form of the subparameter appears in a DD statement of a cataloged procedure, stepname refers to a previous step of the procedure, or, if no such step is found, to a previous step of the current job.

*.stepname.procstep.ddname
indicates that the DSNAME parameter (only) is to be copied from a DD statement in a cataloged procedure. The EXEC statement that called for

execution of the procedure, as well as the step and DD statement of the procedure, must be identified.

&&name
allows the programmer to supply a temporary name for a data set that is to be deleted at the end of the job. The operating system substitutes a unique symbol for this subparameter. The programmer can use the temporary name in other steps to refer to the data set. The same symbol is substituted for each recurrence of this name within the job. Upon completion of the job, the name is dissociated from the data set. The same temporary name can be used in other jobs without ambiguity.

&&name(element)
allows the programmer to supply a name for a member of a temporary partitioned data set that will be deleted at the end of the step.

NULLFILE
serves the same function as the DUMMY parameter (described above).

QNAME Parameter (Defining the Data to be Accessed by TCAM)
specifies the name of a TPROCESS macro that defines a destination queue for messages that are to be processed by an application program and creates a process entry for the queue in the Terminal Table (see the section "Defining Terminal and Line Control Areas" in the chapter entitled "Using the Teleprocessing Feature").

Note: The DCB parameter is the only parameter that can be coded on a DD statement with the QNAME parameter. The only operands that may be specified as subparameters are BLKSIZE, BUFL, LRECL, OPTCD, and RECFM.

DCB Parameter (Describing the Attributes of the Data Set)
allows the programmer to specify at execution time, rather than at compilation time, information for completing the data control block associated with the data set (see "Execution Time Data Set Requirements" and "Additional File Processing Information" for further information about the data control block and DCB subparameters).

The first subparameter of this parameter may be used to copy DCB attributes from the data set label of a cataloged data set or from a preceding DD statement (see the publication OS/VS JCL Reference for detailed information about the DCB subparameter).

SEP and AFF Parameters (Optimizing Channel Usage)
allow the programmer to optimize the use of channels among groups of data sets. SEP indicates channel separation and AFF indicates channel affinity. SEP and AFF are ignored for any data sets that have been allocated devices by the automatic volume recognition (AVR) option.

If neither parameter is supplied, any available channel, consistent with the UNIT parameter requirement, is assigned. The affinity parameter groups two or more data sets so that they can be separated from another data set requesting channel separation. For indexed sequential data sets these parameters are written in the same way as those for any data set. For VSAM data sets, these parameters should not be used if the data and its index reside on unlike devices. They can be used in succeeding DD statements to refer to the first DD statement defining an indexed sequential data set. However, the second and third DD statements cannot request separation from or affinity to one another because they are unnamed. Thus, to establish channel separation and affinity for all of the areas, the name subparameter of the UNIT parameter must be used to request specific devices on specific channels.

UNIT Parameter (Requesting a Unit)
specifies the quantity and types of input/output devices to be allocated for use by the data set.

If the UNIT parameter is not specified in the current DD statement, there are several ways in which the unit information may be inferred by the system:

- If the current data set has already been created and it is either being passed to the current step, or if it has been cataloged, any unit name specified in this DD statement is ignored.

- If the REF subparameter of the VOLUME parameter is specified, the current data set is given affinity with the data set referred to; that

data set's defining DD statement provides the unit information.

- If the current data set is to operate in the split cylinder mode with a previously defined data set, it will reside on the unit specified in the DD statement for the previous data set.
- If the current data set is to use space suballocated from that assigned to a previously defined data set, it will reside on the same unit as the data set from which the space is obtained.
- If the current data set is assigned to the standard output class (SYSOUT is specified), it is written on the unit specified by the operator for class A.

If the current data set is in the input stream (defined by a DD * or DD DATA statement), the DD statement defining the data set should not contain a UNIT parameter.

If this parameter specifies a mass storage device for a data set being created, it is also necessary to reserve the space the data set will occupy, using another parameter of the DD statement. For VSAM data sets, the AFF and SEP subparameters should not be used if the data and its index reside on unlike devices. Depending on the way in which the space will be used, the SPACE, SPLIT, or SUBALLOC parameter can be specified. These parameters are discussed under individual headings.

If the UNIT parameter specifies a tape device, no SPACE, SPLIT, or SUBALLOC parameters are required.

The UNIT parameter must be specified if VOLUME=SER is specified in the DD statement.

UNIT Subparameters:

name
: specifies the name of an input/output device, a device class name, or any meaningful combination of input/output devices specified by an installation. (Mass storage devices and magnetic tape devices can be combined. No other device type combination is allowed.) Names and device classes are defined at system generation time. The device class names that are required for IBM cataloged procedures and are normally used by most installations are shown in Figure 10. These names can be specified by the installation at system generation time.

  The block size specified in the source program (in the BLOCK CONTAINS clause or in the record description) must not exceed the maximum block size permitted for the device. For example, the maximum block size for the IBM 2314 is 7294 characters, and the maximum block size for the IBM 2400 series is 32,760 characters.

  Note: When device-independence is specified by use of UT as the device class in the ASSIGN statement in the Environment Division, the device chosen by the system will be dependent on the DD statement. Therefore, if the user's installation has both an IBM 2314 and an IBM 3330 that may be used as utility devices, the user should write

      BLOCK CONTAINS 7294 CHARACTERS

  (or any number smaller than 7294) to ensure that the block can be contained on one track.

n
: specifies the number of devices to be allocated to the data set. If this parameter is omitted, 1 is assumed.

P
: specifies parallel mount.

DEFER
: indicates deferred mounting. Deferred mounting cannot be specified for a new output data set on a mass storage device or for an ISAM data set. For VSAM data sets, it indicates that the volumes are not to be mounted until access method services requires them.

SEP=(list of up to eight ddnames)
: specifies unit separation.

AFF=ddname
: specifies unit affinity.

Class Name	Class Functions	Device Type
SYSSQ	writing reading	mass storage magnetic tape
SYSDA	writing reading	mass storage

Figure 10. Device Class Names Required for IBM-Supplied Cataloged Procedures

COPIES Parameter (Requesting Additional Data Set Copies)
is specified when more than one copy of the output data sets is desired. It can be specified only with the SYSOUT parameter on the same DD statement. The maximum number of copies that can be requested is 255. For further information on the use of the COPIES parameter, see the publication OS/VS JCL Services.

OUTLIM Parameter (Specifying Output Record Limit)
is specified, for OS/VS1 only, to limit the number of logical records to be included in the output data set being routed through the output stream. For OS/VS2, OUTLIM is ignored. When the limit is reached, an exit provided by the System Management Facilities option is taken to a user-supplied routine that determines whether to cancel the job or increase the limit. If the exit routine is not supplied, the job is canceled. The largest number that can be specified is 16777215.

The OUTLIM parameter has meaning only if the System Management Facilities option is in use in the system and job, and step data collection was selected at system generation. OUTLIM is ignored unless SYSOUT is coded in the operand field of the same DD statement. If OUTLIM is not specified, the system default will be used unless SYSUDUMP or SYSABEND is being processed; in this case no output limiting is done.

TERM Parameter (Notify System of a Special Device)
specifies different information for OS/VS1 and OS/VS2. For VS1, the TERM parameter notifies the system to the presence of an RTAM (Remote Terminal Access Method) device used with RES (Remote Entry Services). For VS2 with TSO, the TERM parameter notifies the system that a data set is coming from or going to a time sharing terminal.

See the publication OS/VS1 RES System Programmer's Guide, Order No. GC28-6878, for detailed information on Remote Entry Services.

TERM Subparameter:
RT
indicates that a remote unit record device is in use for RTAM and that the usual allocation processing is to be bypassed. RT can only be specified on a DD statement for a job that is a system task. RT should not be specified on DD*, DD DATA and SYSOUT DD statements.

TS
indicates to the system that the input or output data being defined is coming from or going to a time sharing terminal. If TSO is not in use, the DD statement containing the TERM parameter is treated as a DD DUMMY statement. Only the DCB parameter can be specified with TERM=TS; any other parameters specified on the same DD statement are ignored.

UCS Parameter (Specifying Character Set for a 1403 or 3211 Printer)
describes the character set to be used for printing an output data set on a 1403 or 3211 printer. In order to use a particular special character set, an image of the character set must be contained in SYS1.IMAGELIB, and the chain or train corresponding to the character set must be available for use. The UCS parameter, the DDNAME parameter, and the DCB subparameters RKP, CYLOFL, and INTVL, are mutually exclusive. For further information on the UCS parameter, see the publication OS/VS System Programming Library: Data Management.

UCS Subparameters:
character set code
identifies the special character set to be used for printing the data set.

FOLD
specifies that the chain or train corresponding to the desired character set is to be loaded in the fold mode. The fold mode is described in the publication IBM 2821 Control Unit, Order No. GA24-3112.

VERIFY
specifies that the operator is to verify that the correct chain or train is mounted before the data set is printed. If the VERIFY subparameter is specified and the FOLD subparameter is not, a comma must precede VERIFY since FOLD is a positional subparameter and its absence must be indicated.

SPACE Parameter (Allocating Mass Storage Space)

specifies space to be allocated in a mass storage volume. Although SPACE has no meaning for tape volumes, if a data set is assigned to a device class that contains both mass storage devices and tape devices, SPACE should be specified. For VSAM data sets, space is allocated through access method services.

Two forms of the SPACE parameter may be used, with or without absolute track address (ABSTR). The ABSTR parameter requests that allocation begin at a specific address.

SPACE Subparameters:

{ABSTR / TRK / CYL / average-record-length}

specifies the unit of measurement in which storage is to be assigned. The units may be tracks (ABSTR or TRK), cylinders (CYL), or records (average-record-length, expressed as a decimal number). In addition, the ABSTR subparameter indicates that the allocated space is to begin at a specific track address. If the specified tracks are already allocated to another data set, they will not be reallocated to this data set.

Note: For ISAM data sets, only the CYL or ABSTR subparameter is permitted. When an ISAM data set is defined by more than one DD statement, all must specify either CYL or ABSTR; if some statements contain CYL and others ABSTR, the job will be abnormally terminated.

(primary-quantity[,secondary-quantity] [,directory- or index-quantity])

specifies the amount of space to be allocated for the data set. The primary quantity indicates the number of records, tracks, or cylinders to be allocated when the job step begins. For ISAM data sets, this subparameter specifies the number of cylinders for the prime, overflow, or index area (see "Execution Time Data Set Requirements"). The secondary quantity indicates how much additional space is to be allocated each time previously allocated space is exhausted. This subparameter must not be specified when defining an ISAM data set. If a secondary quantity is specified for a sequential data set, the program may receive control when additional space cannot be allocated to write a record. The directory quantity is used when initially creating a partitioned data set (PDS), and it specifies the number of 256-byte records to be reserved for the directory of the PDS. It can also specify the number of cylinders to be allocated for an index area embedded within the prime area when a new ISAM data set is being defined (see the publication OS/VS JCL Reference).

Note: The directory contains the name and the relative position, within the data set, for each member of a partitioned data set. The name requires eight bytes, the location four bytes. Up to 62 additional bytes can be used for additional information. For a directory of a partitioned data set that contains load modules, the minimum directory requirement for each member is 34 bytes.

RLSE

indicates that all unused external storage assigned to this data set is to be released when processing of the data set is completed.

{MXIG / ALX / CONTIG}

qualifies the request for the space to be allocated to the data set. MXIG requests the largest single block of storage that is greater than or equal to the space requested in the primary quantity. ALX requests the allocation of additional tracks in the volume. The operating system will allocate tracks in up to five blocks of storage, each block equal to or greater than the primary quantity. CONTIG requests that the space indicated in the primary quantity be contiguous.

If this subparameter is not specified, or if any option cannot be fulfilled, the operating system attempts to assign contiguous space. If there is not enough contiguous space, up to five noncontiguous areas are allocated.

ROUND

indicates that allocation of space for the specified number of records is to begin and end on a cylinder boundary. It can be used only when average record length is specified as the first subparameter.

quantity

specifies the number of tracks to be allocated. For an ISAM data set, this

quantity must be equivalent to an integral number of cylinders; it specifies the space for the prime, overflow, or index area (see "Execution Time Data Set Requirements").

beginning address
specifies the relative number of the track desired, where the first track of a volume is defined as 0. (Track 0 cannot be requested.) The number is automatically converted to an address based on the particular device assigned. For an ISAM data set this number must indicate the beginning of a cylinder.

directory quantity
defines the number of 256-byte records to be allocated for the directory of a new partitioned data set. It also specifies the number of tracks to be allocated for an index area embedded within the prime area when a new indexed data set is being defined. In the latter case, the number of tracks must be equivalent to an integral number of cylinders (see the publication OS/VS JCL Reference).

SPLIT Parameter (Allocating Mass Storage Space)
is specified when other data sets in the job step require space in the same mass storage volume, and the user wishes to minimize access-arm movement by sharing cylinders with the other data sets. The device is then said to be operating in a split cylinder mode. In this mode, two or more data sets are stored so that portions of each occupy tracks within every allocated cylinder.

Note: SPLIT should not be used a) when one of the data sets is an ISAM data set, or b) under VS2 Release 2 or later.

SPLIT Subparameters:
n
indicates the number of tracks per cylinder to be used for this data set if CYL is specified. If the average record length is specified, n is the percentage of the tracks per cylinder to be used for this data set.

{CYL / average-record-length}
indicates the units in which the space requirements are expressed in the next subparameter. The units may be cylinders (CYL) or physical records (in which case the average record length in bytes is specified as a decimal number not exceeding 65,535). If the average record length is given, and the data set is defined to have a key, the key length must be given in the DCB parameter of this DD statement.

primary-quantity
defines the number of cylinders or space for records to be allocated to the entire group of data sets.

secondary-quantity
defines the number of cylinders or space for records to be allocated each time the space allocated to any of the data sets in the group has been exhausted and more data is to be written. This quantity will not be split.

A group of data sets that share cylinders in the same device is defined by a sequence of DD statements. The first statement in the sequence must specify all parameters except secondary quantity, which is optional. Each of the statements that follow the first statement must specify only n, the amount of space required.

SUBALLOC Parameter (Allocating Mass Storage Space)
permits space to be obtained from another data set for which contiguous space was previously allocated. This enables data sets to be stored in a single volume. Space obtained through suballocation is removed from the original data set, and may not be further suballocated. The SUBALLOC parameter should not be used to obtain space for an ISAM data set, nor should it be used under VS2 Release 2 or later.

Except for the subparameters described below, the subparameters in the SUBALLOC parameter have the same meaning as those described in the SPACE parameter.

SUBALLOC Subparameters:
ddname
indicates that space is to be suballocated from the data set defined by the DD statement, ddname, that appears in the current step.

stepname.ddname
indicates that space is to be suballocated from the data set defined by the DD statement, ddname, occurring in a previous step, stepname. If this form of the subparameter appears in a DD statement in a cataloged procedure, stepname refers to a previous step of the procedure, or if no such step is found, to a previous step of the current job.

stepname.procstep.ddname
indicates that space is to be suballocated from a data set defined in a cataloged procedure. The first term identifies the step that called for execution of the procedure, the second identifies the procedure step, and the third identifies the DD statement that originally requested space.

VOLUME (VOL) Parameter (Specifying Volume Information)
specifies information about the volume(s) on which an input data set resides, or on which an output data set will reside. A volume can be a tape reel, or a mass storage device. Volumes can be used most efficiently if the programmer is familiar with the states a volume can assume. Volume states involve two criteria: the type of data set the programmer is defining and the manner in which the programmer requests a volume.

Data sets can be classified as one of two types, temporary or nontemporary. A temporary data set exists only for the duration of the step that creates it. A nontemporary data set can exist after the job is completed. The programmer indicates that a data set is temporary by coding:

- DSNAME=&&name
- No DSNAME parameter
- DISP=(NEW,DELETE), either explicitly or implied, e.g., DISP=(,DELETE)
- DSNAME=reference, referring to a DD statement that defines a temporary data set.

All other data sets are considered nontemporary. If the programmer attempts to keep or catalog a passed data set that was declared temporary, the system changes the disposition to PASS unless DEFER was specified in theUNIT parameter. Such a data set is deleted at the end of the job.

The manner in which the programmer requests a volume can be considered specific or nonspecific. A specific reference is implied whenever a volume with a specific serial number is requested. Any one of the following conditions denotes a specific volume reference:

- The data set is cataloged or passed from an earlier job step.
- VOLUME=SER is coded in the DD statement.
- VOLUME=REF is coded in the DD statement, referring to an earlier specific volume reference.

All other types of volume references are nonspecific. (Nonspecific references can be made only for new data sets, in which case the system assigns a suitable volume.)

The state of a volume determines when the volume will be demounted and what kinds of data sets can be assigned to it.

Mass Storage Volumes: Mass storage volumes differ from tape volumes in that they can be shared by two or more data sets processed concurrently by more than one job. Because of this difference, mass storage volumes can assume different volume states than tape volumes. The volume state is determined by one characteristic from each of the following groups:

Mount Characteristics	Allocation Characteristics
Permanently Resident	Public
Reserved	Private
Removable	Storage

Permanently resident volumes are always mounted. The permanently resident characteristic applies automatically to:

- All physically permanent volumes, such as 2305 Fixed Head Storage.
- The volume from which the system is loaded (the IPL volume).
- The volume containing the system data sets SYS1.LINKLIB, SYS1.PROCLIB, and job scheduler queue.

- Other volumes can be designated as permanently resident in a special member of SYS1.PROCLIB named PRESRES.

Permanently resident volumes are always public. The reserved characteristic applies to volumes that remain mounted until the operator issues an UNLOAD command. They are reserved by a MOUNT command referring to the unit on which they are mounted or by a system parameter library entry. The removable characteristic applies to all volumes that are neither permanently resident nor reserved. Removable volumes do not have an allocation characteristic when they are not mounted. A reserved volume becomes removable after an UNLOAD command is issued for the unit on which it resides.

The allocation characteristics, public, private, and storage, indicate the availability status of a volume for assignment by the system to temporary data sets, and, if the volume is removable, when it is to be demounted. A public volume is used primarily for temporary data sets and, if it is permanently resident, for frequently used data sets. It must be requested by a specific volume reference if a data set is to be kept or cataloged on it. If a public volume is removable, it is demounted only when its unit is required by another volume. The programmer can change a public volume to private status by specifying VOLUME=PRIVATE. A private volume must be requested by a specific volume reference. A new data set can be assigned to a private volume by specifying VOLUME=PRIVATE. If the volume is reserved, it remains mounted until the operator issues an UNLOAD command for the unit on which it resides. If it is removable, it will be demounted after it is used, unless the programmer specifically requested that it be retained (VOLUME=,RETAIN) or passed (DISP=,PASS). Once a removable volume has been made private, it will ultimately be demounted. To use it as a public volume, it must be remounted. A storage volume is used as an extension of main storage, to keep or catalog nontemporary data sets having nonspecific volume requests. The programmer can assign the PRIVATE option to storage volumes.

Figure 11 shows how mass storage volumes are assigned their mount and allocation characteristics.

Mount Characteristic	Allocation Characteristic		
	Public	Private	Storage
Permanently Resident	system parm. library or Default	system parm. library	system parm. library
Reserved	system parm. library or MOUNT command	system parm. library or MOUNT command	system parm. library or MOUNT command
Removable	Default	VOLUME=PRIVATE	na

na = Not applicable

Figure 11. Mass Storage Volume States

Magnetic Tape Volumes: The volume state of a reel of magnetic tape is also determined by a combination of mount and allocation characteristics:

Mount Characteristics	Allocation Characteristics
Reserved	Private
Removable	Scratch

The reserved-scratch combination is not a valid volume state. Reserved tape volumes assume their state when the operator issues a MOUNT command for the unit on which they reside. They remain mounted until the operator issues a corresponding UNLOAD command. Reserved tapes must be requested by a specific volume reference.

A removable tape volume is assigned the private characteristic when one of the following occurs:

- It is requested with a specific volume reference.
- It is requested for allocation to a nontemporary data set.
- The VOLUME parameter is coded with the PRIVATE option.

A removable-private volume is demounted after its last use in the job step, unless the programmer requests that it be retained.

All other tape volumes are assigned the removable-scratch state. The tape

volumes remain mounted until their unit is required by another volume.

Volume Parameter Facilities: The facilities of the VOLUME parameter allow the programmer to:

- Request private volumes (PRIVATE)
- Request that private volumes remain mounted until the end of the job (RETAIN)
- Select volumes when the data set resides on more than one volume (volume-sequence-number)
- Request more than one nonspecific volume (volume-count)
- Identify specific volumes (SER and REF)

These facilities are all optional. The programmer can omit the VOLUME parameter when defining a new data set, in which case the system assigns a suitable public or scratch volume.

VOLUME Subparameters:

PRIVATE
indicates that the volume on which space is being allocated to the data set is to be made private. If the PRIVATE, SER, and REF subparameters are omitted for a new output data set, the system assigns the data set to any suitable public or scratch volume that is available.

RETAIN
indicates that this volume is to remain mounted after the job step is completed. Volumes are retained so that data may be transmitted to or from the data set, or so that other data sets may reside in the volume. If the data set requires more than one volume, only the last volume is retained; the other volumes are previously dismounted. Another job step indicates when to dismount the volume by omitting RETAIN. If each job step issues a RETAIN for the volume, the retained status lapses when execution of the job is completed.

volume-sequence-number
is a 1- to 4-digit number that specifies the sequence number of the first volume of the data set that is read or written. The volume sequence number is meaningful only if the data set is cataloged and earlier volumes are omitted.

volume-count
specifies the number of volumes required by the data set. Unless the SER or REF subparameter is used this subparameter is required for every multivolume output data set.

SER
specifies one or more serial numbers for the volumes required by the data sets. A volume serial number consists of one to six alphanumeric characters. If it contains fewer than six characters, the serial number is left justified and padded with blanks. If SER is not specified and DISP is not specified as NEW, the data set is assumed to be cataloged, and serial numbers are retrieved from the catalog. A volume serial number is not required for new output data sets. Two volumes should not have the same serial number. When the SER parameter is included, the volume is treated as PRIVATE commencing with allocation for the current job step. If this subparameter is specified, the UNIT parameter must also be specified.

REF
indicates that the data set is to occupy the same volume(s) as the data set identified by dsname *.ddname, *.stepname.ddname, or *.stepname.procstep.ddname. Figure 12 shows the data set references.

If SER or REF is not specified, the control program will allocate any nonprivate volume that is available.

LABEL Parameter (Describing Data Set Label)
specifies information about the label or labels associated with the data set. If a data set is passed from a previous job step, label information is retained from the DD statement that specified DISP=(,PASS). A LABEL parameter, if specified in the DD statement receiving the passed data set, is ignored. If the LABEL parameter is omitted and the data set is not being passed, standard labeling is assumed. The operating system verifies mounting when the label parameter specifies standard labels (SL) or standard and user labels (SUL). Nonstandard labels can be specified only when installation-

Option	Refers to
REF=dsname	A data set named dsname
REF=*.ddname	A data set indicated by DD statement ddname in the current job step
REF=*.stepname.ddname	A data set indicated by DD statement ddname in the job step stepname
REF=*.stepname.procstep.ddname	A data set indicated by DD statement ddname in the cataloged procedure step procstep called in the job step stepname (see "Using the Cataloged Procedures")

Figure 12. Data Set References

written routines to write and process nonstandard labels have been incorporated into the operating system (see "User Label Processing" and the publication OS/VS Tape Labels for information about writing these routines).

LABEL Subparameters:

data-set-sequence-number
is a 4-digit number that identifies the relative location of the data set with respect to the first data set in a tape volume. (For example, if there are three data sets in a magnetic tape volume, the third data set is identified by data set sequence number 0003.) If the data set sequence number is not specified, the operating system assumes that it is 0001. (This option should not be confused with the volume sequence number, which represents a particular volume for a data set.)

{AL, AUL, BLP, LTM, , NL, SL, NSL, SUL}
specifies the kind of label used for the data set. AL indicates American National Standard labels. AUL indicates American National Standard user labels. BLP indicates that the system is not to perform label processing for the tape data set. LTM indicates that the data set has a leading tapemark. A , indicates that the data set has standard labels and another subparameter follows. NL indicates no labels. SL indicates IBM standard labels. NSL indicates nonstandard label. SUL indicates IBM standard and user labels.

EXPDT=yyddd
RETPD=xxxx
specifies how long the data set shall exist. The expiration date, EXPDT=yyddd, indicates the year (yy) and the day (ddd) that the data set can be deleted. The period of retention, RETPD=xxxx, indicates the period of time, in days, that the data set is to be retained. If neither is specified, the retention period is assumed to be zero.

PASSWORD
indicates that the data set is to be made accessible only when the correct password is issued by the operator. The operating system assigns security protection to the data set. In order to retrieve the data set, the operator must issue the password on the console.

NOPWREAD
indicates that the data set can be read without the password, but the correct password must be issued by the operator before the data set can be changed, extended, or deleted.

IN
indicates that the data set is to be processed for input only.

OUT
indicates that the data set is to be processed for output only.

DISP Parameter (Specifying Data Set Status and Disposition)
describes the status of a data set and indicates what is to be done with it after its last use, or at the end of the job. The job scheduler executes

the requested disposition functions at the completion of the associated job step. If the step is not executed because of an error found by the system before trying to initiate the step (e.g., an error in a job control language statement), the remaining statements are read and interpreted; however, none of the succeeding steps are executed, and the requested dispositions are not performed. This parameter can be omitted for data sets created and deleted during a single job step. Additional information about the relationship between the DISP parameter and the volume table of contents is contained in "Additional File Processing Information."

DISP Subparameters:

NEW

indicates that the data set is being generated in this step. If the status is omitted, the NEW subparameter is assumed.

OLD

indicates that the data set specified in the DSNAME parameter already exists.

SHR

has meaning only in a multiprogramming environment for existing data sets that reside on mass storage volumes. This subparameter indicates that the data set is part of a job in which operations do not prevent simultaneous use of the data set by another job. For a non-VSAM data set that is to be shared, the DD statement DISP parameter should be specified as DISP=SHR for every reference to the data set in a job. Unless this is done, the data set cannot be used by a concurrently operating job, and the job will have to wait until the particular file is free.

Note: For VSAM data sets, this subparameter alone does not guarantee that sharing will take place. For more information see the chapter "Data Security and Integrity" in the publication OS/VS Virtual Storage Access Method (VSAM) Programmer's Guide.

MOD

causes logical positioning after the last record in the data set. It indicates that the data set already exists and that it is to be added to, rather than read. When MOD is specified and neither the volume serial number is given nor the data set cataloged or passed from an earlier job step, MOD is ignored and NEW is assumed. If the volume serial number is given, it is assumed that the data set is on the specified volume.

DELETE

causes the space occupied by the data set to be released for other purposes at the end of the current step. If the data set is cataloged, and the catalog is used to locate it, reference to the data set is removed from the catalog. If it is on a mass storage device, all references are removed from the volume table of contents, and the device space is made available for use by other data sets. If the data set is on tape, the volume in which the data set resides is then available for use by other data sets.

KEEP

ensures that the data set remains intact until a DELETE parameter is exercised in either the current job orsome subsequent job. If the data set is on a mass storage device, it remains tabulated in the volume table of contents after completion of the job. When the volume containing the data set is to be dismounted, the operator is advised of the disposition.

PASS

indicates that the data set is to be referred to in a later step of the current job, at which time its disposition may be determined. When a subsequent reference to this data set is encountered, its PASS status lapses unless another PASS is issued. The final disposition of the data set should be specified in the last DD statement referring to the data set within the current job.

While a data set is in PASS status, the volume(s) on which it resides are, in effect, retained; that is, the system will attempt to avoid demounting them. If demounting is necessary, the system will ensure proper remounting, through operator messages. The unit name specified on the DD statement in the receiving step must be consistent with the unit name in the passing step.

CATLG

causes the creation, at the end of the job step, of an index entry in the system catalog pointing to the data set. The data set can be referred to by name in subsequent jobs, without the need for volume serial number or device type information from the

programmer. Cataloging also implies KEEP.

UNCATLG
causes the index entry that points to this data set to be removed from the index structure at the end of this step. The data set is not deleted. If it is on a mass storage volume, reference to it remains in the volume table of contents.

Note: The absence of DELETE, KEEP, PASS, CATLG, and UNCATLG indicates that no special action is to be taken to alter the permanent or temporary status of this data set. If the data set was created in this job, it will be deleted at the end of the current step. If the data set existed before this job, it will be kept.

The third subparameter indicates the disposition of the data set in the event the job step terminates abnormally. This is the conditional disposition subparameter. Explanations for DELETE, KEEP, CATLG, and UNCATLG are the same as those for normal termination. The following points should be noted when using the third subparameter.

- If a conditional disposition is not specified and the job step abnormally terminates, the requested disposition (the second subparameter) is performed.
- Data sets that were passed but not received by subsequent steps because of abnormal termination will assume the conditional disposition specified the last time they were passed. If a conditional disposition was not specified at that time, all new data sets are deleted and all other data sets are kept.
- A conditional disposition other than DELETE for a temporary data set is invalid and the system assumes that it is DELETE.

SYSOUT Parameter (Routing Data Set through the Output Stream)
schedules a printing or punching operation for the data set described by the DD statement.

SYSOUT Subparameters:
classname
specifies the system output class on which the data set is to be written. A classname is an installation specified 1-character name designating the output class to which the data set is to be written. Each classname is related to a particular output unit. Valid values for the SYSOUT parameter are A through Z and 0 through 9. A is the standard output class.

Note: Classes 0 through 9 should not be used except in cases where the other classes are not sufficient. These classes are intended for future features of systems.

(classname[,program-name][,form-no])
classname specifies the class associated with the output device to which the output data set is to be written. Output writers route data from the output classes to system output devices. The DD statement for this data set can also include a unit specification describing the intermediate mass storage device and an estimate of the space required. If there is a special installation program to handle output operations, its program-name should be specified. Program-name is the member name of the program, which must reside in the system library. If the output data set is to be printed or punched on a specific type of output form, a 4-digit form number should be specified. Form-no. is used to instruct the operator of the form to be used in a message issued at the time the data set is to be printed.

Notes:

- If both the program-name and form-no. are omitted, the delimiting parentheses can be omitted.
- If the Direct SYSOUT Writer is used to write a data set, both the form-no. and program-name are ignored. All parameters on the DD statement, i.e., UNIT or SPACE, are also ignored.

FCB Parameter (Specifying Output for a 3211 Printer or a 3525 Card Punch)
is used to select the forms control image to be used to print an output data set on a 3211 printer, or a 3525 card punch with the read feature. The FCB parameter will be ignored if the data set is not written to either one of these devices. The FCB parameter, the DDNAME parameter and the DCB subparameters RKP, CYLOFL, and INTVL, are mutually exclusive.

FCB Subparameters:
image-id
identifies the image to be loaded into the forms control buffer. For further information on the forms control

buffer, see the publication OS/VS Data Management for System Programmers.

ALIGN
VERIFY
requests the operator to check the alignment of the printer forms before the data set is printed or to verify that the image displayed on the printer is the desired one.

AMP Parameter (Specifying Information for VSAM Processing)
specifies information to be used for processing by VSAM. The AMP parameter and its subparameters are described under "VSAM-only JCL Parameter" in the chapter "VSAM File Processing".

ADDITIONAL DD STATEMENT FACILITIES

By specifying certain ddnames, the programmer can request the operating system to perform additional functions. The operating system recognizes these special-purpose ddnames:

- JOBLIB and STEPLIB to identify private user libraries
- SYSABEND and SYSUDUMP to identify data sets on which a dump may be written
- SYSCHK to identify the checkpoint data set written during the original execution of a processing program.
- JOBCAT and STEPCAT to identify VSAM user catalogs.

JOBLIB AND STEPLIB DD STATEMENTS

The JOBLIB and STEPLIB DD statements are used to concatenate a user's private library with the system library (SYS1.LINKLIB). Use of JOBLIB results in the system library being combined with the private library for the duration of a job; use of STEPLIB, for the duration of a job step. During execution, the library indicated in these statements is scanned for a module before the system library is searched.

The JOBLIB DD statement must appear immediately after the JOB statement and its operand field must contain at least the DSNAME and DISP parameters. The DISP parameter must contain PASS as the second subparameter if the library is to be made available to later job steps. Only one JOBLIB statement may be specified for a job but more than one library may be specified on a JOBLIB statement. The JOBLIB statement is meant to concatenate existing private libraries with the system library. It need not be specified for load modules created in the job or for permanent members of the system library (see "Checklist for Job Control Statements" and "Libraries" for examples).

The STEPLIB DD statement may appear in any position among the DD statements for the job step. The library should be defined as OLD. If the library is to be passed to other job steps, the second subparameter of the DISP parameter should be coded PASS. A later job step may then refer to the library by coding its STEPLIB DD statement as follows:

```
//STEPLIB DD DSNAME=*.stepname.STEPLIB, X
//    DISP=(OLD,PASS)
```

The STEPLIB statement overrides the JOBLIB statement if both are present in a job step.

SYSABEND AND SYSUDUMP DD STATEMENTS

The ddnames SYSABEND or SYSUDUMP identify a data set on which an abnormal termination dump may be written. The dump is provided for job steps subject to abnormal termination.

The SYSABEND DD statement is used when the programmer wishes to include in his dump the problem program storage area, the system nucleus, and the trace table if the trace table option had been requested at system generation time.

The SYSUDUMP DD statement is used when the programmer wishes to include only the problem program storage area.

The programmer may rout the dump directly to an output writer by specifying the SYSOUT parameter on the DD statement. In a multiprogramming environment, the programmer may also define the intermediate direct-access device by specifying the UNIT and SPACE parameters.

SYSCHK DD STATEMENT

The SYSCHK DD statement is required when a job is being submitted for deferred checkpoint/restart. It defines a

checkpoint data set written during the original execution of a processing program. For detailed information about the checkpoint/restart facilities, see the publication OS/VS Checkpoint/Restart.

The SYSCHK DD statement must immediately precede the first EXEC statement of the resubmitted job when restart is to begin at a checkpoint. The RESTART parameter must be included on the JOB statement; otherwise the SYSCHK DD statement will be ignored.

Different SYSCHK DD statement parameter specification rules apply depending on whether the checkpoint data set is cataloged or not. These rules are discussed in detail in the publication OS/VS JCL Reference.

JOBCAT AND STEPCAT DD STATEMENTS

The JOBCAT DD statement specifies the VSAM user catalog that is to be available throughout a VSAM processing job, while the STEPCAT DD statement specifies the VSAM user catalog that is to be available for a single job step in a VSAM processing job. For more detailed information on the facilities provided by the JOBCAT and STEPCAT DD statements, as well as the specification rules, see the publication OS/VS Virtual Storage Access Method (VSAM) Programmer's Guide.

PROC STATEMENT

The PROC statement may appear as the first control statement in a cataloged procedure and must appear as the first control statement in an in-stream procedure. The PROC statement must contain the term PROC in its operation field. For a cataloged procedure, the PROC statement assigns default values to symbolic parameters defined in the procedure; its operand field must contain symbolic parameters and their default values. The PROC statement marks the beginning of an in-stream procedure; its operand may contain symbolic parameters and their default values.

PEND STATEMENT

The PEND statement must appear as the last control statement in an in-stream procedure and marks the end of the in-stream procedure. It must contain the term PEND in the operation field. The PEND statement is not used for cataloged procedures. For further information about in-stream procedures, see "Testing a Procedure as an In-Stream Procedure" in "Using the Cataloged Procedures."

COMMAND STATEMENT

The operator issues commands to the system via the console or a command statement in the input stream. Commands can also be issued to the system via a command statement in the input stream. However, this should be avoided since commands are executed as they are read and may not be synchronized with execution of job steps. Command statements must appear immediately before a JOB statement, an EXEC statement, a null statement, or another command statement.

The command statement contains identifying characters (//) in columns 1 and 2, a blank name field, a command, and, in most cases, an operand field. The operand field specifies the job name, unit name, or other information being considered.

Note: A command statement cannot be continued, it must be coded on one card or card image.

DELIMITER STATEMENT

The delimiter statement marks the end of a data set in the input stream. The identifying characters /* must be coded into columns 1 and 2, the other fields are left blank. Comments are coded as necessary.

Note: The end of a data set need not be marked in an input stream that is defined by a DD * statement.

NULL STATEMENT

The null statement is used to mark the end of a job in an input stream. It causes the card reader file to be effectively closed. The identifying characters // are coded into columns 1 and 2, and all remaining columns are left blank.

COMMENT STATEMENT

The comment statement is used to enter any information considered helpful by the programmer. It may be inserted anywhere in the job control statement stream after the JOB Statement. (The comment statement contains a slash in columns 1 and 2, and an asterisk in column 3. The remainder of the card contains comments.) Comments are coded in columns 4 through 80, but a comment may not be continued onto another statement.

When the comment statement is printed on an output listing, it is identified by the appearance of asterisks in columns 1 through 3.

BATCH COMPILATION

The batch compile feature is used to compile multiple programs or subprograms with one invocation of the compiler. The object programs produced from the batch compilation may be link-edited into either one load module or separate load modules.

This feature must be requested at compile time by specification of BATCH in the PARM field or, if a cataloged procedure is used, in the PARM.COB field of the EXEC card. In the BATCH mode, all options specified on the EXEC card, as well as all default options, apply to every program in the batch unless specific options are overridden, via the CBL card, for an individual compilation.

The CBL card must be the first card in each program within a batch mode. The CBL card, in addition to separating logical program units, may be used to change existing options (as they were specified or defaulted to on the EXEC card) for that individual program, and has the following format:

```
CBL [option 1][,option 2]...[,option n]
```

The letters CBL may appear in any three consecutive columns 1 through 72, and the option(s) specified may be any PARM compiler option(s) except SIZE, BUF, BATCH, L120, L132, SYMDMP, and LVL, which are ignored if indicated.

Notes:

- A sequence number may appear in columns 1 through 6 of the CBL card.
- In most cases, an option specified on the CBL card overrides the corresponding EXEC card option for compilation of that one program only. However, this is not true in the case of options that require use of a file that will be used in a subsequent compilation in the batch, or in a subsequent job step. Generally speaking, it is unwise to use the CBL card to specify any such option, because use of that file in compilation may cause an abend. Options in this category are LOAD, LIB, DECK, FDECK, and CDECK.

  If some programs in a batch compilation require the use of one of these options and the other programs do not, specify the option on the EXEC card, and then specify the NO... form of the option on those CBL cards where the option is not wanted. For example, if programs 2, 3, and 5 in a batch compilation require FDECK and programs 1 and 4 require NOFDECK, then FDECK should be specified (or defaulted to) on the EXEC card, and NOFDECK specified on the CBL cards for programs 1 and 4. (It would not be possible to specify or default to NOFDECK on the EXEC card and then override it with FDECK on a CBL card.)
- If a CBL card is present and BATCH is not specified on the EXEC card, the CBL card is regarded as an invalid statement.
- If the compiler NAME option is specified on the CBL card, a linkage editor NAME control card is generated for this compilation, facilitating the link-editing of the program into a separate load module.
- The output of a batch compilation may be executed only if the member name specified at compile time is the name specified at execution time.
- The batch option may be used in conjunction with BASIS. This facility provides the COBOL programmer with the ability to combine a (multiple) BASIS library member(s) and/or a (multiple) COBOL source program(s) with one invocation of the compiler.
- The BATCH option and the SYMDMP option are mutually exclusive.

When the batch option is used in combination with BASIS, the following rules apply:

1. All the BASIS library members to be compiled must be members of the partitioned data set(s) referred to by the SYSLIB DD data set name(s).

2. Each BASIS library member must contain only one source program.

Figure 13 shows that with one invocation of the COBUCL cataloged procedure (see the chapter "Using the Cataloged Procedures"), the programs COMPILE1, COMPILE2, and COMPILE3 are compiled and two load modules created as follows:

1. COMPILE1 and COMPILE2 are link-edited together to form one load module with the member name of COMPILE2, a typical called/calling situation. (For further discussion of articulation between COBOL programs, see the chapter "Called and Calling Programs".) In this case, the entry point of the load module is still the first program, COMPILE1.

2. COMPILE3 is link-edited to create the load module with the member name of COMPILE3.

Figure 14 shows that with one invocation of the COBUCL procedure the programs PROG1 and PROG2 and BASIS library members PAYROLL and PAYROLL2 are compiled and four load modules are created. (An example of how to execute load modules created with the BATCH feature using the procedure COBUCL is given in Figure 13.)

```
//jobname          JOB          1,BATCH,MSGLEVEL=1
//COMPILE          EXEC[1]      COBUCL,PARM.COB='BATCH,NAME'
//COB.SYSIN        DD           *
 CBL NONAME
      ID DIVISION.
      PROGRAM-ID.  COMPILE1.
         .
         .
CBL NAME
      ID DIVISION.
      PROGRAM-ID.  COMPILE2.
         .
         .
CBL NAME
      ID DIVISION.
      PROGRAM-ID.  COMPILE3.
         .
         .
/*
//LKED.SYSLMOD     DD           DSN=BATCHRUN,SPACE=(TRK,(10,5,2)),....
/*
//COMPILE2         EXEC         PGM=COMPILE2
//STEPLIB[2]       DD           DSN=BATCHRUN[2],DISP=SHR,....
//  (Cards needed to execute COMPILE1 and COMPILE2)
/*
//COMPILE3         EXEC         PGM=COMPILE3
//STEPLIB          DD           DSN=BATCHRUN,DISP=SHR,....
//  (Cards needed to execute COMPILE3)
/*
```

[1]In the compile step, no special JCL is needed for SYSLIN because the COBUCL cataloged procedure is used (see the chapter "Using The Cataloged Procedures").
[2]In the link-edit step, a partitioned data set is created with the DSN of BATCHRUN.

Figure 13. Example of a Batch Compilation

```
//jobname         JOB          1,BATBASIS,MSGLEVEL=1
//COMP            EXEC         COBUCL,PARM.COB='BATCH,NAME,LIB'
//COB.SYSLIB      DD           DSN=LIBPOS,...[1]
//COB.SYSIN       DD           *
 CBL              NAME,NOLIB
                  IDENTIFICATION DIVISION.
                  PROGRAM-ID. PROG1.
                     .
                     .
 CBL              NAME
BASIS             PAYROLL
 CBL              NAME
BASIS             PAYROLL2
 CBL              NAME,NOLIB
                  IDENTIFICATION DIVISION
                  PROGRAM-ID. PROG2.
                     .
                     .
/*
//LKED.SYSLMOD[2] DD           DSN=BATCHBAS,SPACE=(TRK,(10,5,2)),...
/*
```

[1]This partitioned data set contains as separate members PAYROLL and PAYROLL2.
[2]The load modules of these four COBOL programs exist as separate members of a partitioned data set named BATCHBAS.

Figure 14. Creation of Four Load Modules with Programs PROG1 and PROG2 and BASIS Library Members PAYROLL and PAYROLL2

DATA SET REQUIREMENTS

COMPILER

A number of data sets may be defined for a compilation job step; six of these (SYSUT1, SYSUT2, SYSUT3, SYSUT4, SYSIN, and SYSPRINT) are always required. SYSUT5 is required if the SYMDMP or TEST option is invoked. SYSUT6 is required if FIPS flagging is requested. Additional data sets (SYSLIN, SYSPUNCH, SYSTERM, and SYSLIB and/or other COPY libraries) are optional.

For compiler data sets other than utility data sets, a logical record size can be specified by using the LRECL and BLKSIZE subparameters of the DCB parameter. The values specified must be permissible for the device on which the data set resides. LRECL equals the logical record size, and BLKSIZE equals LRECL multiplied by n, where n is equal to the blocking factor. If this information is not specified in the DD statement, it is assumed that the logical record sizes for the unblocked data sets have the following default values:

Unblocked Data Set	Default Value (bytes)
SYSIN	80
SYSLIN	80
SYSPUNCH	80
SYSLIB or other COPY libraries	80
SYSPRINT	121 or 133
SYSTERM	121

Note: The default for SYSPRINT has a value of 133 if 132 is selected with LSTCOMP or LSTONLY.

The ddname that must be used in the DD statement describing the data set appears as the heading for each description that follows. Figure 15 lists the function, device requirements, and allowable device classes for each data set. (See "Appendix D: Compiler Optimization" for further information on blocked compiler data sets other than utility data sets.)

SYSUT1, SYSUT2, SYSUT3, SYSUT4, SYSUT5, SYSUT6

The DD statements using these ddnames define utility data sets that are used by the compiler when processing the source module. The data set defined by the SYSUT1 DD statement must be on a mass storage device. Except for SYSUT5, which is needed at execution time, these data sets are

temporary and have no connection with any other job step. For example, the DD statement

```
//SYSUT1 DD UNIT=SYSDA,SPACE=(TRK,(40,10))
```

specifies that the data set is to be written on any available mass storage device, with a primary allocation of 40 tracks. Additional tracks, if required, are to be allocated in groups of 10. The data set is to be deleted at the end of the job step (by default).

Note: The NULLFILE or DUMMY parameter or the RLSE subparameter of the SPACE parameter should never be specified for any of these data sets. In addition, the data sets must be single volume, since the compiler uses the TCLOSE facility extensively and TCLOSE will not reposition multivolume data sets.

SYSIN

The data set defined by the SYSIN DD statement contains the input for the compiler, i.e., the source module statements that are to be processed. The data set assigned to this DD statement is a sequential data set, or a member of a partitioned data set. It may be part of the input stream. If so, it is commonly referred to as a SYSIN data set. For example,

```
//SYSIN    DD    *
```

specifies that the input data set follows in the input stream.

SYSPRINT

This data set is used by the compiler to produce a listing. Output may be directed to a printer, a mass storage device, or a magnetic-tape device. The listing will include the results of the default or specified options of the PARM parameter (i.e., diagnostic messages, the object code listing). For example, in the DD statement

```
//SYSPRINT  DD  SYSOUT=A
```

SYSOUT is the disposition for printer data sets, and A is the standard output class for printer data sets.

SYSTERM

The data set defined by the SYSTERM DD statement is used for certain compiler output for a TSO terminal user when the TERM option is specified. The compiler output consists of diagnostic and progress messages and compiler statistics. For example:

```
//SYSTERM DD TERM=TS
```

TERM=TS indicates that the SYSTERM data set is to be directed to the TSO terminal.

SYSPUNCH

The data set defined by the SYSPUNCH DD statement is used to punch an object module deck or, if FDECK or CDECK is specified, to punch a source deck. This data set can be directed to a card punch, mass storage device, or magnetic tape. For example, in the DD statement

```
//SYSPUNCH  DD  SYSOUT=B
```

SYSOUT is the disposition for punch data sets, and B is the standard output class for punch data sets.

Note: The SYSPUNCH DD statement is not required if NODECK is in effect. SYSPUNCH may be either a sequential data set or a member of a PDS.

ddname	Type	Function	Device Requirements	Allowable Device Classes
SYSIN (required)	Input/output	Reading the source program	Card reader Intermediate storage	SYSSQ, SYSDA, or the input stream device (specified by DD * or DD DATA)
SYSPRINT (required)		Writing the storage map, listings, and and messages	Printer Intermediate storage	SYSSQ, SYSDA, standard output class A
SYSTERM (optional)		Writing diagnostic and progress messages	Output device TSO terminal	
SYSPUNCH (optional)		Punching the object module deck	Card punch Mass storage Magnetic tape	SYSCP, SYSSQ, SYSDA, standard output class B
SYSLIN (optional)		Creating an object module data set as output from the compiler and input to the linkage editor	Mass storage Magnetic tape	SYSSQ, SYSDA
SYSUT1 (required)	Utility	Work data set needed by the compiler during compilation	Mass storage	SYSDA
SYSUT2 (required)		Work data set needed by the compiler during compilation	Mass storage Magnetic tape	SYSSQ, SYSDA
SYSUT3 (required)		Work data set needed by the compiler during compilation	Mass storage Magnetic tape	SYSSQ, SYSDA
SYSUT4 (required)		Work data set needed by the compiler during compilation	Mass storage Magnetic tape	SYSSQ, SYSDA
SYSUT5		Work data set needed when SYMDMP or TEST option is in effect	Mass storage Magnetic tape	SYSSQ, SYSDA
SYSUT6	Utility	Work data set needed when LVL option is in effect	Mass storage Magnetic tape	SYSSQ, SYSDA
SYSLIB and/or other COPY libraries (optional)	Library	Optional user source program libraries	Mass storage	SYSDA

Note: Once created, a SYSUT5 data set can be moved only to a device of the same type. That is, if the SYSUT5 data set is put on tape at compile time, that data set cannot be moved to a disk at execution time. The SYSUT5 data set must be unblocked.

Figure 15. Data Sets Used for Compilation

SYSLIN

The device defined by the SYSLIN DD statement is used by the compiler to store an object module. It may be on a mass storage or magnetic tape device. For example:

```
//SYSLIN DD  DSNAME=&&GOFILE,                X
//             DISP=(MOD,PASS),              X
//             UNIT=SYSDA,                   X
//             SPACE=(TRK,(30,10))
```

The temporary name of the data set is GOFILE, the parameter DISP=(MOD,PASS) indicates that the data is to be created or added to in this job step and is to be passed to another job step, which may be the linkage editor step. The device to be assigned for storage is a mass storage device on which 30 tracks are initially allocated to the data set. If more space is needed, tracks are allocated 10 at a time.

Note: The SYSLIN DD statement is not required if NOLOAD is in effect. SYSLIN may be either a data set or a member of a PDS.

SYSLIB and/or Other COPY Libraries

These DD statements define the libraries (PDS's) that contain the data requested by COPY statements (in the source module) or by a BASIS card in the input stream. The DD statement must be SYSLIB if a BASIS library is to be included, or if the COPY statement does not specify a library name (by qualifying the text name). Libraries must always be on mass storage devices. Note that more than one partitioned data set may be used for the library function by concatenating them with SYSLIB (see "Libraries" for an example). Although only one SYSLIB statement may be used in a compilation job step, multiple user-defined COPY libraries may be used. For example, in the DD statements

```
//SYSLIB    DD  DSNAME=USERLIB,DISP=OLD
//HOUSELIB  DD  DSNAME=COPYX,DISP=OLD
//PRIVLIB   DD  DSNAME=COPYZ,DISP=OLD
```

the names of the source libraries are USERLIB, COPYX, and COPYZ. DISP=OLD indicates that the libraries have been created in a previous job and are cataloged, or have been created in a previous step in this job. No other information need be given if the specified libraries have been cataloged.

Notes: Maximum blocksize for any COPY library is 16K (there is no such restriction for BASIS).

When concatenating SYSLIB, the library with the largest blocksize must be specified in the first DD statement.

The SYSLIB and other COPY library DD statements are not required if NOLIB is in effect.

LINKAGE EDITOR

Five data sets are required for linkage editor processing. Others may be necessary if secondary input is specified. In the following discussions, the ddname that must be used in the DD statement describing the data set appears as the heading for each description of the particular data set. For any user-defined data set, the ddname is defined by the programmer. Figure 16 lists the function, device requirements, and allowable device classes for each data set.

SYSLIN

The SYSLIN DD statement defines the data set that is primary input to linkage editor processing. Normally this data set consists of the output from a previous compilation job step. The primary input may also be linkage editor control statements, such as the INCLUDE, LIBRARY, or OVERLAY statements (see "Calling and Called Programs"). The input device assigned to this data set is either the device transmitting the input stream, if the input is an object module deck, or a device designated by the programmer. However, the data set may simply be passed from the previous compilation job step. For example, in the DD statement

```
//SYSLIN    DD  DSNAME=*.STEPNAME.SYSLIN,  X
//              DISP=(OLD,DELETE)
```

the data set is defined in the SYSLIN DD statement contained in the compiler job step, STEPNAME. DISP=(OLD,DELETE) indicates that the data set was created in a previous job step and is to be deleted at the end of this job step.

SYSPRINT

The data set defined by the SYSPRINT DD statement is used by the linkage editor to produce a listing. For example:

```
//SYSPRINT  DD  SYSOUT=A
```

Output may be directed to a printer or to an intermediate data set. The listing may include any options specified by the PARM parameter of the EXEC statement (a module map or cross reference list, diagnostic or informative messages, etc.).

ddname	Type	Function	Device Requirements	Allowable Device Classes
SYSLIN (required)	Input/ output	Primary input data, normally the output of the compiler	Mass storage Magnetic tape Card reader	SYSSQ, SYSDA, or the input stream device (specified by DD * or DD DATA)
SYSPRINT (required)		Diagnostic messages Informative messages Module map Cross-reference list	Printer Intermediate storage	SYSSQ, standard output class A
SYSLMOD (required)		Output data set for the load module	Mass storage	SYSDA
SYSUT1 (required)	Utility	Work data set	Mass storage	SYSDA
SYSLIB (required) for COBOL Library subroutines	Library	Automatic call library (SYS1.COBLIB is the name of the COBOL subroutine library)	Mass storage	SYSDA
SYSTERM (required if TERM option is specified)		Numbered error/warning messages	Printer TSO terminal	
User-specified (optional)	--	Additional object modules and load modules	Mass storage Magnetic tape	SYSDA, SYSSQ

Figure 16. Data Sets Used for Linkage Editing

SYSTERM

The SYSTERM DD statement is optional; it describes a data set that is used only for numbered error/warning messages. Although intended to define the terminal data set when the linkage editor is being used under TSO, the SYSTERM DD statement can be used in any environment to define a data set consisting of numbered error/warning messages that supplements the SYSPRINT data set.

SYSTERM output is defined by including a SYSTERM DD statement and specifying TERM in the PARM field of the EXEC statement. When SYSTERM output is defined, numbered messages are then written to both the SYSTERM and SYSPRINT data sets.

The SYSTERM DD statement is specified as follows:

```
//SYSTERM DD SYSOUT=A
```

SYSLMOD

The SYSLMOD DD statement defines the output data set, in this case the load module. The load module must be placed in a library as a named member. The library can be the Link Library (SYS1.LINKLIB) or a private user-defined library. Such libraries must always reside on a mass storage device, and space for the library is allocated when the library is created. For example, in the DD statement

```
//SYSLMOD DD DSNAME=SYS1.LINKLIB(MEMBER),X
//           DISP=OLD
```

the load module, MEMBER, is stored as a member of the link library. DISP=OLD indicates that the library is already created and additions are to be made to it.

```
//SYSLMOD  DD  DSNAME=LIB1(BALANCE),      X
//               DISP=(NEW,CATLG),        X
//               VOLUME=SER=111111,       X
//               SPACE=(TRK,(40,10,1)),   X
//               UNIT=SYSDA
```

The load module, BALANCE, is to be a member of a library, LIB1, which is to be created in this job step, with BALANCE as its firstmember. The mass storage volume to which it is directed is identified by the serial number, 111111. A primary quantity of 40 tracks is allocated to the library with an additional allocation for one 256-byte record to be used for the directory. If more space is needed for the library, tracks are added, 10 at a time. (However, no additional space can be allocated for the directory.)

Note: If the load module is placed in a private library, the JOBLIB or STEPLID DD statements must be specified in subsequent jobs that execute load modules from the library.

SYSUT1

The SYSUT1 DD statement defines a utility data set used by the linkage editor when processing object modules and load modules. The data set must be on a mass storage device. It is a temporary data set and has no connection with any other job step. For example:

```
//SYSUT1 DD UNIT=SYSDA,SPACE=(TRK,(40,10))
```

The data set is initially allocated 40 tracks on any available mass storage device. If more space is needed, tracks are added, 10 at a time. A temporary name is assigned to the data set for the job step.

SYSLIB

The SYSLIB DD statement assigns the named partitioned data set to the automatic call library from which modules may be automatically obtained by the linkage editor to resolve external references.

```
//SYSLIB   DD   DSNAME=SYS1.COBLIB,DISP=SHR
```

This statement assigns the COBOL subroutine library to the automatic call library. When there is a possibility that the compiler may have generated calls to any COBOL library subroutines, the SYSLIB statement must be specified (see "Appendix B: COBOL Library Subroutines" for a list of library subroutines, their functions, and entry points).

User-Specified Data Sets

Additional data sets may be defined for linkage editor processing. These data sets may be used as additional input sources of object modules or load modules. They may also be concatenated with the primary input data set or the automatic call library (see "Libraries").

LOADER

One data set (SYSLIN) is required for loader processing. Three are optional (SYSLIB, SYSLOUT and SYSTERM). (These ddnames can be changed during system generation with the LOADER macro instruction.) In addition, any DD statements and data required by the loaded program must be included in the input deck.

In the following discussions, the default ddname for the DD statement describing the data set appears as the heading for each description of the particular data set.

SYSLIN

The SYSLIN DD statement defines the data set that is primary input to the loader. This input can be either object modules produced by the COBOL compiler or load modules produced by the linkage editor, or both. The loader allows both object module and load module concatenation on SYSLIN. The data sets defined by the SYSLIN DD statements can be either sequential data sets or members of a partitioned data set, or both.

SYSLIB

The SYSLIB DD statement defines the data set containing IBM or user-written library routines to be included in the loaded program. The SYSLIB data set is searched when unresolved references remain after processing SYSLIN and, optionally, searching the link pack area of OS/VS2 or the resident reenterable modules feature of OS/VS1. The library may contain either object modules or load modules but not both. The data set defined by the SYSLIB DD statement must be a partitioned data set.

SYSLOUT

The SYSLOUT DD statement defines the data set used for error and warning messages and for an optional map of external references. The data set must be a sequential data set. The record format of SYSLOUT is always assumed to be FBSA.

EXECUTION TIME DATA SETS

Any number of data sets may be used for execution time processing. These data sets, or files, are identified in the source program, and each must be described by a DD statement. The ddname is used to link the DD statement to the COBOL ASSIGN clause in the source program that specifies the ddname. DD statement requirements for the DISPLAY, ACCEPT, EXHIBIT, and TRACE statements are discussed in the following text. DD statements that specify COBOL debugging aids and an abnormal termination dump are also discussed. Use of either the Sort or the RERUN feature requires additional DD statements. For information about these statements, see "Using the Sort Feature" and "Using the Checkpoint/Restart Feature."

DISPLAY Statement

The DISPLAY statement requires an associated DD statement unless the data is to be displayed on the console. The DD statements needed for each form of the DISPLAY statement are as follows:

Example 1:

```
DISPLAY {identifier} ...UPON SYSPUNCH
        {literal   }
```

//SYSPUNCH DD applicable parameters

It is assumed that SYSPUNCH is an unblocked data set that has a logical record length of 80 characters. For example:

```
//SYSPUNCH DD SYSOUT=B
```

However, the programmer can specify a blocked data set by using the subparameters of the DCB parameter as follows:

```
RECFM=FB,BLKSIZE=n*80
```

where:

n is the blocking factor

SYSPUNCH must be on a device where blocking is permitted. For example:

```
//SYSPUNCH    DD   UNIT=SYSSQ,                X
//                 DCB=(RECFM=FB,             X
//                 BLKSIZE=160),              X
//                 LABEL=(,NL)
```

Example 2:

When the UPON option is omitted, SYSOUT is the default option.

```
DISPLAY {identifier} ...
        {literal   }
```

//SYSOUT DD applicable parameters

It is assumed that SYSOUT is an unblocked data set that has a line width of 121 characters (1-byte for the control character).

For example:

```
//SYSOUT DD SYSOUT=A
```

However, the programmer can specify an alternate line width, recording mode, and/or a blocked data set by using the DCB parameter. To specify an alternate line width, the subparameters of the DCB parameter are used as follows:

```
LRECL=line width+1,BLKSIZE=LRECL value
```

To specify a blocked data set, the subparameters are used as follows:

```
RECFM=FBA,LRECL=line width+1,
BLKSIZE=n*(LRECL value),
```

where:

n is a blocking factor

SYSOUT must be on a device where blocking is permitted. The extra character in LRECL allows for the carriage control character. For example, to specify an alternate line width, the following SYSOUT statement can be used.

```
//SYSOUT      DD SYSOUT=A,DCB=(LRECL=133, X
//                BLKSIZE=133)
```

To specify a blocked data set, the following SYSOUT statement can be used.

```
//SYSOUT      DD   DSNAME=PRINTOUT,        X
//                 UNIT=SYSDA,...,         X
//                 DCB=(RECFM=FBA,         X
//                 LRECL=121,              X
//                 BLKSIZE=605),           X
//                 VOLUME=SER=111111
```

Note: If the problem uses the Sort/Merge feature, remember that SYSOUT is the default error message data set, and a conflict can arise. See "Additional DD Statements" in the chapter "Using the Sort/Merge Feature" for suggested solutions.

Example 3:

The DISPLAY statement can use a mnemonic-name rather than a system-name.

```
DISPLAY { identifier } ...UPON mnemonic-name
        { literal    }
```

where mnemonic-name is associated with the word SYSPUNCH or SYSOUT in the Environment Division.

```
// { SYSPUNCH } DD applicable parameters
   { SYSOUT   }
```

ACCEPT Statement

The ACCEPT statement requires an associated DD statement unless the data is being accepted from the console, format 2 of the ACCEPT statement is used, or ACCEPT MESSAGE is used (making possible use of the options DATE, DAY, and TIME). The DD statements for each form of the ACCEPT statement are as follows:

Example 1:

```
ACCEPT identifier
```

When the FROM option is omitted, SYSIN is the default option.

```
//SYSIN    DD    applicable parameters
```

Example 2:

```
ACCEPT identifier FROM mnemonic-name
```

where mnemonic-name is associated with the word SYSIN in the Environment Division.

```
//SYSIN DD applicable parameters
```

It is assumed that SYSIN is an unblocked data set that has a logical record length of 80 characters.

For example:

```
//SYSIN  DD  *
           (data)
/*
```

However, the programmer can specify a blocked data set by using the subparameters of the DCB parameter as follows:

```
RECFM=FB,BLKSIZE=n*80
```

where:

n is the blocking factor

SYSIN must be on a device where blocking is permitted. For example:

```
//SYSIN       DD   UNIT=2400,...,          X
//                 DCB=(RECFM=FB,          X
//                 BLKSIZE=160),           X
//                 LABEL=(,NL)
```

If a logical record length of other than 80 characters is desired, it must be specified in the LRECL field of the DCB parameter.

EXHIBIT or TRACE Statement

The EXHIBIT or TRACE statement requires a SYSOUT DD statement as discussed for DISPLAY.

Note: If the job step already includes a SYSOUT DD statement for some other use, another may not be inserted since all SYSOUT output from any source in the job step will be merged onto the one SYSOUT data set defined for that job step.

COBOL Debugging Aids

If one or more of the options FLOW, STATE, and SYMDMP is in effect, the following DD statement must be used:

```
//SYSDBOUT    DD    applicable parameters
```

If the output is routed through the output stream and written on a system output device, the following may be used:

```
//SYSDBOUT    DD    SYSOUT=A
```

The recording mode is FBA. The user can, however, specify a blocked data set and alternate recording mode by using the DCB subparameters.

Note: It is assumed that SYSDBOUT is an unblocked data set that has a line width of 121 bytes (one byte for a control character).

See the chapter "Symbolic Debugging Features" in this manual.

The following DD statement must be used to make the COBOL library module ILBODBE0 available at execution time:

```
//STEPLIB DD DSN=SYS1.COBLIB,DISP=SHR
```

If an error message is printed by the debugging modules, the COBOL library module ILBODBE0 is loaded dynamically from SYS1.COBLIB. This module is not link edited into the COBOL object program.

Abnormal Termination Dump

To obtain an operating system hexadecimal dump in case the job is abnormally terminated by the system, or by executing the COBOL statement CALL 'ILBOABN0' USING identifier, one of the following DD statements must be used:

```
//SYSABEND DD applicable parameters.

//SYSUDUMP DD applicable parameters.
```

The dump provided when the SYSABEND DD statement is used includes the system nucleus, the program storage area, and a trace table, if the trace table option was requested at system generation. The SYSUDUMP DD statement provides a dump of the program storage area. The applicable parameters are those for a physical sequential data set. If the dump is routed through the output stream and written on a system output device, the following DD statement may be used:

```
//SYSUDUMP DD SYSOUT=A
```

Note: If a COBOL program abnormally terminates, then a formatted dump is provided for all COBOL programs compiled with the SYMDMP option which could include the abnormally terminating program and its callers, up to and including the main program. The //SYSABEND or //SYSUDUMP DD card need not be included. For a discussion of the symbolic dumping option, as well as of other COBOL symbolic debugging options, see the chapter entitled "Symbolic Debugging Features."

COUNT Option

If the COUNT option is in effect, the following DD statement must be used:

```
//SYSDBOUT DD applicable parameters
```

For example, if the output is routed through the output stream and written on a system output device, the following may be used:

```
//SYSDBOUT DD SYSOUT=A
```

In addition to the SYSDBOUT DD statement, the SYSCOUNT DD statement must also be used:

```
//SYSCOUNT DD SYSOUT=A
```

COBOL Subroutine Library

The user may concatenate a library of selected COBOL object-time subroutines with the link library. (For information on how this can be accomplished, see the section "Sharing COBOL Library Subroutines" in the chapter entitled "Libraries").

USER NON-VSAM FILE PROCESSING

This section describes the processing of non-VSAM files. A description of VSAM file processing is in the section "VSAM File Processing."

USER-DEFINED FILES

Files that are processed in a COBOL program must be described as data sets to the operating system. Whenever a file is specified in a program by the following statement:

```
SELECT [OPTIONAL] file-name
    ASSIGN TO assignment-name
```

this file must be described in an FD file-name entry and in a DD statement in the execution-time job step. The ddname in the DD statement is a portion of the assignment-name (sometimes also known as "system-name") specified in the ASSIGN TO clause. In the assignment-name

```
UT-2400-S-TAXRATE
```

TAXRATE is the ddname portion of the assignment-name.

Note: The device-number specified in the assignment-name is ignored by the compiler. Actual device allocation is a function of the DD statement.

FILE NAMES AND DATA SET NAMES

The terms "file" (COBOL usage) and "data set" (operating system usage) have essentially the same meaning. There may, however, be a difference between the file-name and the data set name. The data set name always represents a specific data set. The file-name can, at different times, represent different data sets. The DD statement allows a programmer to select, at the time his program is executed, the specific data set that is to be associated with a particular file-name. This facility can be especially powerful when applied to input data sets.

The file-name is a name known within the COBOL program. Changing a file-name requires changing input/output statements and recompiling the program. Changing a DD statement when a program is executed is a simple procedure.

As an example, consider a COBOL program that might be used in exactly the same way for several different master files. It might contain the clause

```
SELECT MASTER ASSIGN TO
    DA-3330-D-MASTERA... .
```

In that case, the following DD statements, used at different times, would assign the different named data sets to the program:

```
//MASTERA   DD  DSNAME=MASTER1,...
//MASTERA   DD  DSNAME=MASTER2,...
//MASTERA   DD  DSNAME=MASTER3,...
```

If the first DD statement appears in the job step that calls for execution of the program, any reference within the program to MASTER is a reference to the data set named MASTER1; if the second DD statement appears, the reference is to MASTER2; if the third, the reference is to MASTER3.

However, if a file-name within a program is always to be applicable to only a single data set, the names might be written as follows:

```
SELECT TAXRATE ASSIGN TO
    UT-2400-S-TAXRATE...
```

The applicable DD statement might be:

```
//TAXRATE   DD   DSNAME=TAXRATE,...
```

Of the names, the ddname portion of the assignment-name that appears in the ASSIGN clause and the ddname of the DD statement must always be the same. The file-name and the data set name may be the same, or they may be different. (Of course, the file-name in the SELECT sentence must be the same as the FD name.)

If two or more files on direct-access devices have the same ddname and are open at the same time (i.e., the output from the files is being merged into one data set), the files must have no conflicting attributes. The foregoing also applies to SYSOUT data sets if they are written on an intermediate direct-access device.

The use of the DISPLAY, EXHIBIT, or READY TRACE verbs causes the library to open the target ddname (SYSOUT, SYSPUNCH, etc.) If the programmer has also assigned one of his output files to the same ddname, he must ensure that he has opened, written, and closed his file before the first execution of any of the previously mentioned verbs.

Additional considerations when using the Sort feature are described under "Additional DD Statements" in the chapter "Using the Sort Feature."

SPECIFYING INFORMATION ABOUT A FILE

Some of the information about the file must always be specified in the FD entry, SELECT sentence, APPLY, and other COBOL clauses. Other information must be specified in the DD statement. For example, the amount of space allocated for a mass storage output file must be specified in the DD statement by the SPACE, SPLIT, or SUBALLOC parameters. Certain characteristics of files cannot be expressed in the COBOL language, and may be specified on the DD statement for the file by the DCB parameter. This parameter allows the programmer to specify information for completing the data control block associated with the file (see "Additional File Processing Information" for a discussion of the data control block, and "Appendix C: Fields of the Data Control Block").

Each file used in the program must be referred to by a particular file processing technique. Four processing techniques are discussed in this publication. They are physical sequential (QSAM), direct (BSAM, BDAM), relative (BSAM, BDAM), and indexed (QISAM, BISAM).

A fifth processing technique, called partitioned data organization (BPAM), is discussed throughout the publication, when it is used for program storage.

A partitioned data set (PDS) is composed of named, independent groups of sequential data, each of which is called a member. Each member has a simple name stored in a directory that is part of the data set and that contains the location of each member's starting point. Partitioned data sets are used to store programs, and are often referred to as libraries.

The full range of facilities available in BPAM are not available to the COBOL programmer. A partitioned data set may be referred to in COBOL only by treating its members as physical sequential data sets.

FILE PROCESSING TECHNIQUES

DATA SET ORGANIZATION

A non-VSAM data set used by a COBOL program can have one of four types of organization: physical sequential, direct, relative, and indexed. The first type (sequential) may be on any input/output device that is supported. All other types must be on mass storage devices (see Figure 17 for information in determining the file processing technique to be used, according to data set organization).

1. A physical sequential data set is one in which records are organized solely on the basis of their successive physical positions.

2. A direct data set is one in which records are referred to by use of relative track addressing. An ACTUAL KEY specifies the track relative to the first track allocated to the data set and identifies the record on the track.

3. A relative data set is one in which records are referred to by use of relative record addressing. A NOMINAL KEY identifies the record location relative to the first record in the data set.

4. An indexed data set is one in which records are arranged on the tracks of a mass storage device so as to permit access in logical sequence (according to a key that is part of every record). A separate index or set of indexes maintained by the system indicates the location of each record. This permits random, as well as sequential, access to any record.

File Processing Requirements	ACCESS Clause and Organization Field (N) in System-name	Permissible Record Formats		Device Requirements	File Processing Technique	Organization Clause
		Blocked	Unblocked			
Write, read, and update standard sequential file	ACCESS SEQUENTIAL or ACCESS clause is omitted N=S	F,V,S	F,V,U	Mass Storage Magnetic Tape Unit Record	QSAM	Sequential (default)
Write and read a mass storage file with relative record addressing	ACCESS SEQUENTIAL or omitted N=R		F	Mass Storage	BSAM	Should not be specified
Read and update a mass storage file with relative record addressing	ACCESS RANDOM N=R		F	Mass Storage	BDAM	
Create and read a mass storage file with relative track addressing	ACCESS SEQUENTIAL or omitted N=D		F,V,U,S	Mass Storage	BSAM	
Create, read, update, and insert into a mass storage file with relative track addressing	ACCESS RANDOM N=D or W(REWRITE)		F,V,U,S	Mass Storage	BDAM	
Create a mass storage file with indexed sequential organization	ACCESS SEQUENTIAL or omitted N=I	F	F	Mass Storage	QISAM	
Read and update a mass storage file with indexed organization	ACCESS SEQUENTIAL or omitted N=I	F	F	Mass Storage	QISAM	
Read, update, and insert into a mass storage file with indexed random organization	ACCESS RANDOM N=I	F	F	Mass Storage	BISAM	

Figure 17. Determining the File Processing Technique

ACCESSING A PHYSICAL SEQUENTIAL FILE

A physical sequential file may only be accessed sequentially, i.e., records are read or written in the order in which they appear on the file. The file processing technique used to create and retrieve a physical sequential file is QSAM (Queued Sequential Access Method). Figure 18 shows the COBOL clauses that may be used with these files. Special considerations for these clauses are as follows:

1. The RESERVE clause can be used to specify more buffer areas, allowing overlap of input/output operations with the processing of data. If this clause is not used, additional buffers may be specified by using the BUFNO option in the DD statement. If no additional buffer areas are specified, two buffers are reserved by the system. When the SAME AREA clause is specified for the file, the number of buffers used is determined from the RESERVE clause or if the RESERVE clause is not present, it is given a default of two. The BUFNO option in the DD statement is ignored if the SAME AREA clause is specified.

2. If a WRITE AFTER POSITIONING statement is used, the record size specified in the FD entry must allow for the carriage control or stacker select character, even though the character is not to be printed or punched. For example, if the record size specified in the FD entry is 121, the actual record is 121 characters; however, only 120 characters are printed or punched.

   If the NOADV compiler is specified and a WRITE BEFORE/AFTER ADVANCING statement is used, the situation is the same as above; the record size specified in the FD entry must allow for the control character, even though the character is not to be printed.

   When the ADV compiler option is specified and a WRITE BEFORE/AFTER ADVANCING statement is used, the record size specified in the FD entry should be the same as the record to be printed. (The compiler adds one to the length specified in the FD when it sets the logical record length in the DCB.)

   Notes:

   - If the immediate destination of the record is a device that does not recognize a carriage control or stacker select character, the system assumes that the control character is the first character of the data. If the WRITE BEFORE/AFTER ADVANCING statement or the WRITE AFTER POSITIONING statement is not used, the first byte of the record is treated as data by the punch or printer.

     The compiler may direct extra records, containing the appropriate control characters, to the file to effect printer spacing as specified in the WRITE BEFORE/AFTER ADVANCING statement. These extra records are for spacing purposes only and will not appear externally if the file is assigned to an online printer. However, if the file is assigned to a device that does not recognize the control characters (for example, a tape or a direct-access device), the extra records are written onto the file. These extra records are produced only if ADVANCING more than three lines is specified or if both the BEFORE and AFTER options are specified for a file.

3. If the input device is the card reader, RECORDING MODE IS F should be specified. If RECORDING MODE IS V or S is specified, the first 8 bytes of the record will be interpreted as the control bytes required for files with format V or S records.

4. If physical sequential files are on magnetic tape, the record block size should be at least 18 bytes. Records less than 18 bytes in length will be read with no problems, unless a parity check occurs. If a parity check occurs while reading a record less than 18 bytes, it will be treated as a noise record and skipped over.

5. The S (standard) option can be specified in the DCB RECFM subparameter for a fixed/blocked record data set with only standard blocks (i.e., having no truncated blocks or unfilled tracks within the data set, except for the last block of the last track). If a fixed/blocked data set is created through the use of an American National Standard COBOL program, a truncated physical block may be written only by the executions of the CLOSE or CLOSE UNIT (or REEL) statement. Use of the standard block option (particularly for direct-access devices having the Rotational Positional Sensing feature) results in significant I/O performance improvements.

6. The T (TRACK OVERFLOW) option can be specified for the DCB RECFM

Data Management Techniques	Device Type	Access Method	KEY Clauses	OPEN Statement	Access Verbs	CLOSE Statement
QSAM	TAPE	SEQUENTIAL	NOT ALLOWED	INPUT [REVERSED | NO REWIND] [LEAVE | REREAD | DISP]	READ [INTO] AT END	[REEL] [LOCK | NO REWIND | POSITIONING | DISP]
				OUTPUT [NO REWIND | LEAVE | REREAD | DISP]	WRITE [FROM] [{BEFORE | AFTER} ADVANCING] [AFTER POSITIONING]	
				EXTEND		
QSAM	MASS STORAGE	SEQUENTIAL	NOT ALLOWED	INPUT	READ [INTO] AT END	[UNIT] [LOCK]
				OUTPUT	WRITE [FROM] INVALID KEY	
				EXTEND	WRITE [FROM] [{BEFORE | AFTER} ADVANCING] [AFTER POSITIONING]	
				I-O	READ [INTO] AT END WRITE [FROM] INVALID KEY REWRITE [FROM]	[LOCK]

Figure 18. COBOL Clause for Physical Sequential File Processing

subparameter of the DD statement for QSAM files with RECORDING MODE V, S, or F. Specification of the T option is equivalent to including the APPLY RECORD-OVERFLOW option in the source program, but use of the T option in the DD statement allows the user to make his selection at object time.

Figures 20 and 21 show the parameters in the DD statement that may be used with physical sequential files. All parameters except the DCB are described in "Job Control Procedures." Additional DCB subparameters not shown in the illustration are required for use with the Sort/Merge feature (see the chapter "Using the Sort/Merge Feature" for information on these parameters).

The DCB subparameters that can be specified in the DD statement for physical sequential files are as follows:

```
DCB=[DEN={0|1|2|3|4}]
    [,TRTCH={C|E|T|ET}]
    [,PRTSP={0|1|2|3}]
    [,MODE={C|E}]
    [,STACK={1|2}]
    [,OPTCD={W|C|WC|T|Q|Z}]
    [,BLKSIZE=integer]
    [,BUFNO=integer]
    [,EROPT={ACC|SKP|ABE}]
    [,RECFM={F|V|U|D[B|S|T][A|M]}]
    [,DIAGNS=TRACE]
    [,FUNC={I|R|P|W|O|X|T}]
```

DEN={0|1|2|3}
can be used with magnetic tape, and specifies a value for the tape recording density in bits per inch as listed in Figure 19. If no value is specified, 800 bits-per-inch is assumed for 7-track tape, 800 bits-per-inch for 9-track tape without dual density and 1600 bits-per-inch for 9-track tape with dual density, depending on the installation's generic definitions for unit names.

DEN Value	Tape Recording Density (Bits per inch)	
	7 Track	9 Track
0	200	--
1	556	--
2	800	800
3	--	1600
4	--	6250

Figure 19. DEN Values

TRTCH={C|E|T|ET}
is used with 7-track tape to specify the tape recording technique, as follows:

C - Specifies that the data-conversion feature is to be used; if data conversion is not available, only format F and format U records are supported by the control program.

E - Specifies that even parity is to be used; if omitted, odd parity is assumed.

T - Specifies that BCD to EBCDIC conversion is required.

ET- Specifies that even parity is to be used and BCD to EBCDIC conversion is required.

PRTSP={0|1|2|3}
specifies the line spacing on a printer as 0, 1, 2, or 3. If PRTSP is not specified, 1 is assumed.

The PRTSP subparameter is valid only if the unit specified for the file is a printer. It is not valid if the file is a report file, nor is it valid if the WRITE statement with the BEFORE/AFTER ADVANCING option or WRITE AFTER POSITIONING is specified in the COBOL source program. Single spacing always is assumed for a printer unless other information is supplied.

MODE={C|E}
can be used with a card reader, a card punch or a card-read punch and specifies the mode of operation as follows:

C - Specifies card image (column binary) mode.

E - Specifies EBCDIC code.

If this information is not supplied by any source, E is assumed.

STACK={1|2}
can be used with a card reader, a card punch, or a card-read punch, and it specifies which stacker bin is to receive the card. Either 1 or 2 is specified. If this information is not supplied by any source, 1 is assumed.

STACK should not be used when the WRITE statement with the AFTER ADVANCING or POSITIONING option is used to specify pocket selection.

OPTCD={W|C|T|Q|Z}
requests an optional service provided by the system as follows:

W - To perform a write validity check (on mass storage devices only).

C - To process using the chained scheduling method (see the publication OS/VS Data Management Services Guide).

T - To request user totaling facility.

Q - To translate to or from ASCII on tape

Z - To request the search direct option (see the publication OS/VS Data Management Services Guide).

Note: If the validity check is specified, the system verifies that each record transferred from main storage to mass storage is written correctly. Standard recovery procedures are initiated if an error is detected.

BLKSIZE=integer
is used to specify the block size. This clause is used only when BLOCK CONTAINS 0 RECORDS was specified at compile time.

BUFNO=number of buffers
is used to specify the number of buffers to be assigned to the file when neither the RESERVE nor the SAME AREA clause is specified for the file in the source program. The maximum number is 255.

EROPT={ACC|SKP|ABE}
specifies the options to be executed if an error occurs in writing or reading a record as follows:

ACC - To accept the error block for processing.

SKP - To skip the error block.

ABE - To terminate the job.

There are two cases when the subparameter can be specified:

- If no error processing declarative (USE sentence) is specified, the option is taken immediately.

- If an error processing declarative is specified, the option is taken after the error declarative returns control via a normal exit (and only if that is the case).

If no option is specified, ABE is assumed.

RECFM={F|V|U|D[B|S|T][A|M]}

specifies the format of the records on the data set (see the JCL manual for the ways in which these individual subparameters can be combined). Only the S and T subparameters have meaning for COBOL; COBOL ignores all others.

F - records are of fixed length.

V - records are of variable length.

U - records are of undefined length.

D - ASCII records of variable length.

B - records are blocked.

S - to expect the data set to consist of standard blocks.

T - to use the TRACK OVERFLOW option (this specification has the same effect as including the APPLY RECORD-OVERFLOW option in the source program).

A - records contain ANS device control characters.

M - records contain machine code control characters.

DIAGNS=TRACE

specifies the Open/Close/EOV trace option which gives a module-by-module trace of Open/Close/EOV's work area and the DCB. The Generalized Trace Facility with the proper options specified must be active in the system while the job that requested the trace is running; the options are MODE=EXT and TRACE=USR.

FUNC={I|R|P|W|O|X|T}

specifies the type of data set to be opened for the 3525 Card-Read-Punch-Print as follows:

I - interpret-punch data set.
R - read
P - punch
W - print
D - data protection for a punch data set
X - printer
T - two-line printer.

For the valid combinations of these values see the publication OS/VS JCL Reference.

	Device Type		
Parameter	Mass Storage	Magnetic Tape	Unit Record
DSNAME	as		
UNIT	as		
VOLUME	as		na
LABEL	SL SUL	SL BLP NL LTM NSL AL SUL AVL	NL
SPACE	as	na	
SUBALLOC	as	na	
SPLIT	as	na	
DISP	NEW MOD {,KEEP ,PASS ,CATLG ,DELETE}		SYSOUT=A,B...
DCB Device Dependent	OPTCD=W, WC	TRTCH, DEN	PRTSP, MODE, STACK
DCB General	OPTCD=C/T, BUFNO, BLKSIZE, EROPT=ABE RECFM=as		EROPT=ACC (printer only) EROPT=ABE
as = Applicable subparameters na = Not applicable			

Figure 20. DD Statement Parameters Applicable to Physical Sequential OUTPUT Files

Parameter	Device Type: Mass Storage	Magnetic Tape	Unit Record
DSNAME	as		
UNIT	Not required if cataloged	Not required if cataloged	as
VOLUME	Not required if cataloged	Not required if cataloged	na
LABEL	SI SUL	SL BLP NL LTM NSL AL SUL AUL	na
SPACE	na		
SUBALLOC	na		
SPLIT	na		
DISP		CLD SHR {,KEEP ,PASS ,CATLG ,UNCATLG ,DELETE}	
DCB Device Dependent	--	TRTCH, DEN	MODE, STACK
DCB General	OPTCD=C/T, BLKSIZE, BUFNO, ERCPT=ACC, SKP, ABE, RECFM=as		

as = Applicable subparameters
na = Not applicable

Figure 21. DD Statement Parameters Applicable to Physical Sequential INPUT and I-O Files

SPECIFYING ASCII FILE PROCESSING

If a program will process an ASCII (American National Standard Code for Information Interchange) QSAM file, the user must identify it as such in one of two ways. One technique is to use the CODE-SET phrase of the COBOL FD statement to reference an alphabet-name that was defined as STANDARD-1 (which is equivalent to ASCII). The other technique is to use the COBOL ASSIGN clause, with assignment-name having the following format:

comment-C-(buffer offset)-name

where:

C
: an organization code which specifies that an ASCII-encoded sequential file is to be processed, or that an ASCII-collated sort is to be performed.

buffer offset
: a two-character field that indicates the length of the block prefix for that file. This entry is required if a non-zero block prefix exists; it must, however, be omitted when an ASCII-collated sort is requested.

name
: a field of 1 to 8 characters that specifies the system-recognized name of the file. It is this external name

that appears in the name field of the DD card for the file.

If this ASSIGN technique is used, LANGLVL(1) must be specified.

PROCESSING ASCII FILES

Record format allowed for ASCII files are the following: mode F (fixed length), mode U (undefined), and mode D (variable length. D-mode records are of variable length with a four-byte record descriptor field for each record. The COBOL programmer processing variable-length records specifies V-mode records. Then the format information generated from the DCB parameter is internally converted to D mode. Format-D records cannot be explicitly specified by the user in a COBOL program.

Block Prefix

An ASCII file may have a variable-length field, called a block prefix, preceding the first logical record in a physical record. If this prefix exists on an ASCII file, its length must be indicated at compile time in the buffer offset field of the ASSIGN clause. The compiler places this length in the DCB parameter at compile time.

Whether the optional block prefix contains the block length or simply user information depends on the type of file specified (input or output) and the internal record mode (i.e., F, U, or D). These distinctions are made in the discussion that follows.

Files Opened as Input: Input files with either blocked or unblocked records have an optional block prefix of 0 to 99 bytes that does not contain the block length but may contain user information. For D-mode records, however, a block prefix of length four may contain the block length. Regardless of the record format, file processing is identical to that for files coded in EBCDIC.

Files Opened as Output: The block prefix for output files applies only to D-mode records and, when specified, must have a length of 4. The prefix must contain the length of the block, which length includes the buffer offset.

For any ASCII output file the ASSIGN clause may include a buffer offset of four. Alternatively, the programmer may omit this specification from the ASSIGN clause, instead making use of the phrase BLOCK CONTAINS 0 RECORDS. The offset can then be specified at execution time in the JCL. However, if BLOCK CONTAINS 0 RECORDS is used, the following options must be included in the JCL:

BUFOFF=(n)
: must be included in the DCB parameter of the DD card, where n is the length of the block prefix from 0 to 99 characters on input, and either 0 or 4 on output.

BLKSIZE=(n)
: must be included on the DD card, where n is the size of the block, including the length of the block prefix.

Notes:

- If a block prefix exists on an ASCII file and the BLOCK CONTAINS clause with the CHARACTERS option is used, the length of the block prefix must be included in the BLOCK CONTAINS clause.
- If either the RECORDS option is specified or the BLOCK CONTAINS clause is omitted, the compiler compensates for the block prefix (if specified).

Additional JCL considerationa for ASCII data sets follow.

LABEL= {AL / AUL / NL}

where AL specifies American National Standard labels, AUL specifies American National Standard and user labels, and NL indicates no labels.

The subparameters below are specified in the DCB parameter of the DD statement:

OPTCD=Q, where Q specifies an ASCII-encoded data set.

RECFM=D, where D represents a variable-length record, is an optional parameter. Whether or not this parameter is specified at execution time, the programmer must specify an ASCII file in the ASSIGN clause as well as a mode-V record. The compiler converts from mode V to mode D, or to the internal representation for a variable-length record.

BUFOFF=(L), where L indicates a four-byte block prefix that contains the block length including the block prefix.

Handling Numeric Data Items from ASCII Files

It is highly recommended that the programmer take advantage of the separately signed numeric data type. The SIGN clause (see "SIGN Clause" in the chapter "Programmer Considerations") can be used to specify the position and the mode of representation of the operational sign of numeric data items.

DIRECT FILE PROCESSING

The direct file processing technique is characterized by the use of the relative track addressing scheme. When this addressing scheme is used, the tracks of mass storage devices are consecutively numbered from 0 to n (where 0 equals the first track of the file, and n equals the last track). The positioning of logical records in a file is determined by the ACTUAL KEY supplied by the user in the Environment Division. The first part of the key, called the track identifier, specifies either the track on which space for the record is first sought or the track at which the search for a record is to begin. The second part, called the record identifier, serves as a unique identifier for the record. Files with direct data organization must be assigned to mass storage devices.

Format
ACTUAL KEY IS data-name

Data-name may be any fixed item from 5 through 259 bytes in length and must be defined in the File Section, Working-Storage Section, or Linkage Section. The following considerations apply when defining the ACTUAL KEY:

- Track Identifier
  The first four bytes of data-name are the track identifier. The identifier is used to specify the relative track address for the record and must be defined as a 5-integer binary data item whose maximum value does not exceed 65,535.
- Record Identifier
  The remainder of data-name, which is 1 through 255 bytes in length, is the record identifier. It represents the symbolic portion of the key field used to identify a particular record on a track.

The following example illustrates the use of the ACTUAL KEY clause:

```
ENVIRONMENT DIVISION.
        .
        .
        .
    ACTUAL KEY IS THE-ACTUAL-KEY.
        .
        .
        .
DATA DIVISION.
        .
        .
        .
WORKING-STORAGE SECTION.
01  THE-ACTUAL-KEY.
    05  TRACK-IDENT  PIC S9(5) COMP SYNC.
    05  RECORD-IDENT PIC X(25).
```

Note: The same record identifier may appear more than once in the same file when using COBOL. However, using the same record identifier is not recommended for the following reasons:

1. If they appear on the same track, only the first occurrence can be retrieved (using BDAM).
2. If an extended search is used in either creating or updating a file, the position of records containing duplicate record identifiers may be unpredictable.

With direct file processing, records must be unblocked and may be V-, U-, F-, or S-mode records. Figure 22 illustrates those parts of a directly organized file that are of importance to a COBOL programmer.

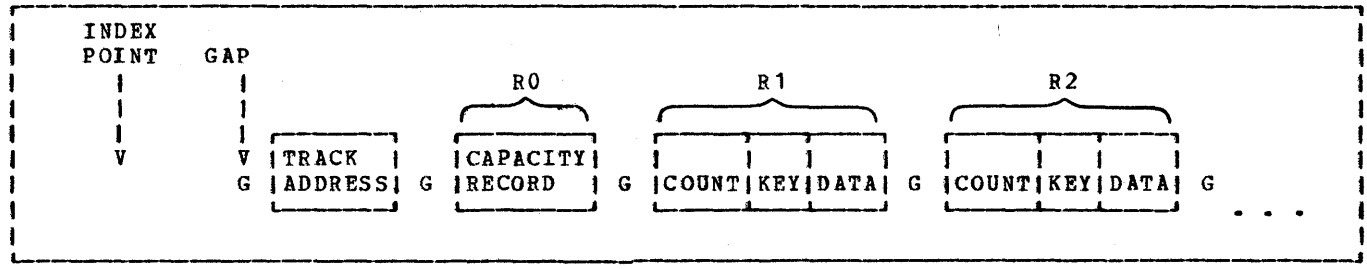


Figure 22. Directly Organized Data as it Appears on a Mass Storage Device

Each track contains the following:

Index Point
: There is one index point to indicate the physical beginning of each track.

G (Gaps)
: Gaps separate the different areas on the track. Certain equipment functions take place as the gap is rotating past the read/write head. The length of the gap varies with the device, the location of the gap, and the length of the preceding area. For instance, the gap that follows the index point is a different length than the gap that follows the track address.

Track Address
: This field defines the physical location of the track. It indicates the cylinder in which the track is located and the read/write head that services the track.

R0 (Capacity Record)
: This field indicates the amount of unused space available for additional records on the track.

R1, R2, ..., Rn
: These are physical records that contain the following:

  count area -- control information

  key area -- the-record identifier (1-255 bytes) as specified by the programmer in the ACTUAL KEY clause.

  data area -- the-data moved into the FD before a WRITE statement was executed.

The following example illustrates the relationship between the ACTUAL KEY and the positioning of records on a mass storage device during the creation of a direct file.

```
ENVIRONMENT DIVISION.
      .
      .
      .
    ACTUAL KEY IS THE-ACTUAL-KEY.
      .
      .
      .
DATA DIVISION.
FILE SECTION.
FD  DIRECT-FILE
    LABEL RECORDS ARE STANDARD.
01  REC-1              PIC X(200).
      .
      .
      .
WORKING-STORAGE SECTION.
01  THE-ACTUAL-KEY.
    05  TRACK-IDENT  PIC S9(5) COMP SYNC.
    05  RECORD-IDENT PIC X(3).
```

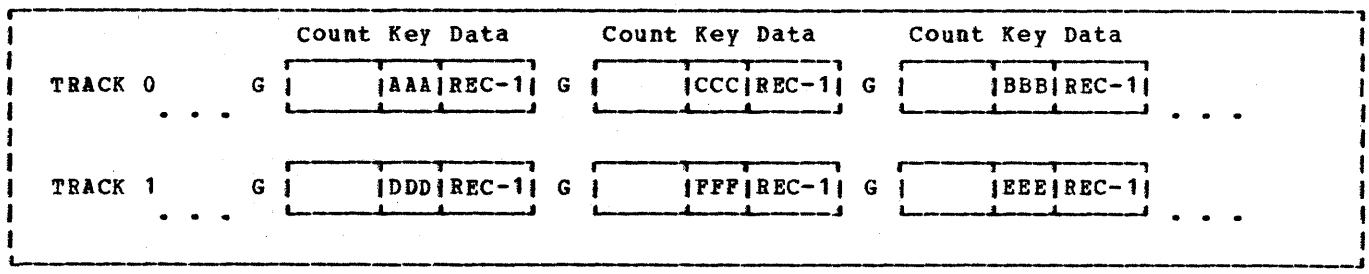


Figure 23. Sample Format of the First Two Tracks of a Direct File

Consider REC-1 being written six times; the contents of THE-ACTUAL-KEY varying with each WRITE instruction:

THE-ACTUAL-KEY

	TRACK IDENT	RECORD IDENT
WRITE 1	0	AAA
WRITE 2	0	CCC
WRITE 3	0	BBB
WRITE 4	1	DDD
WRITE 5	1	FFF
WRITE 6	1	EEE

Relative track 0 and relative track 1 of the mass storage device will appear as shown in Figure 23.

When the WRITE statement is executed, the system seeks the track that corresponds to the number contained in TRACK-IDENT. It then searches for the next available position into which a record may be placed. The system writes a count area, writes the contents of RECORD-IDENT in the key area, and writes the information contained in REC-1 in the data area.

Note: The record identifier is not included in the level-01 record description (REC-1). It will, however, be moved into the output buffer before being written on the mass storage device. Buffer areas, therefore, will be large enough to accommodate both the contents of REC-1 and the record identifier.

Dummy and Capacity Records

Once a direct file has been created, records can be added randomly on tracks formatted sequentially. Unless a track is already filled with data records, it is formatted by the compiler via the writing of dummy records (mode F) or of one capacity record (mode U, V, or S).

In order to format tracks, a COBOL subroutine executes instructions to write dummy records for F-mode files or write capacity records for V-, U-, or S-mode files. Dummy records are identified by the presence of the figurative constant HIGH-VALUE in the first byte of the record identifier portion of the ACTUAL KEY (unless changed by the program collating sequence, in which case the byte contains X'FF'). This indicates to the system that a record can be added to the file in the space assigned to the dummy record. (The user should not attempt to retrieve a dummy record by moving this configuration to the record identifier because it is considered an invalid key.) A capacity record is a single record at the physical beginning of each track that indicates the amount of space available for additional records. As V-, U-, or S-mode records are added to a track, the capacity record is written accordingly. Capacity records are never made available to the user.

When a file is created, it should contain enough dummy records, or appropriately written capacity records, to allow for future expansion. Once the file is closed, more space cannot be allocated and the extent of the file cannot be increased.

Note: Tracks that have been assigned to a file but are not formatted, are considered "allocated." The user should not attempt to write on tracks that have been allocated but not formatted.

Sequential Creation of Direct Data Set

The file processing technique used to create a direct file sequentially is BSAM (Basic Sequential Access Method).

- The associated COBOL statements are summarized in Figure 31.
- The associated JCL parameters are summarized in Figure 32.

The ACTUAL KEY is required. It specifies the relative track number on which the record is to be written. Since access is sequential, all records will be written serially in the sequence in which they are moved into the output buffer. It is, therefore, necessary that all records to be written on the first track (track identifier = 0) be processed before records to be written on the 2nd, 3rd, ..., nth track (track identifier = 1, 2, ..., n-1) are processed.

When records are written sequentially, the user need not update the contents of the track identifier portion of the ACTUAL KEY. A COBOL subroutine will update it as follows:

- Records will be written on the first available track until space is no longer available. At such time, the COBOL subroutine will increment the track identifier by 1, and continue writing on the next track.
- The value of track identifier used by the system is made available to the user in the track identifier portion of the ACTUAL KEY after the record is written.
- After a CLOSE or CLOSE UNIT statement has been executed, the COBOL subroutine places the relative track number of the last track written on (for a data, dummy, or capacity record) in the track identifier of the ACTUAL KEY.
- If the user updates the contents of track identifier and attempts to write on track 2 when tracks 0 through 4 are already full, the system will automatically adjust the track identifier to 5 (the next track with available space).

If the user wishes to skip tracks, the number of tracks, equal to the number of tracks to be advanced, must be added to the track identifier. The COBOL subroutine will then add dummy records (F-mode) or write capacity records (V-, U-, or S-mode) to complete the intervening track(s) (see "Dummy and Capacity Records"). If the value of track identifier for the initial WRITE is not 0, the subroutine will complete the preceding tracks with dummy or capacity records.

SPACE ALLOCATION FOR SINGLE VOLUME FILES: When a file is created sequentially, the number of primary tracks specified on the DD card must be available on the primary volume. If this quantity is not available, the job will not begin execution. Once execution begins however, the final allocation of space will not be made until the file is closed.

The following discussion illustrates the space allocated to a direct file created using BSAM. Figure 24 is an example of a user program that:

- Writes 350-1/2 tracks and then closes the file.
- Specifies SPACE=(TRK,(200,100)) on the associated DD card.

TRACK-LIMIT Clause Specified:

1. If the TRACK-LIMIT clause specifies TRACK-LIMIT = 499 and the file is closed after writing only 350-1/2 tracks:

   Note: A COBOL subroutine will format all remaining tracks up to and including the 500th track. This represents 150 extra tracks on which records may be added.

2. If the TRACK-LIMIT clause specifies TRACK-LIMIT = 300 and the program continues writing all 350-1/2 tracks:

   Note: The TRACK-LIMIT clause is ignored and the system allocates and formats as if no TRACK-LIMIT clause had been specified.

TRACK-LIMIT Clause Not Specified: If the TRACK-LIMIT clause is not specified, the system will allocate the primary extent (i.e., 200 tracks) and up to 16 secondary extents (i.e., 100 tracks each), as required. In Figure 24, the system allocates the first 200 tracks, all of which are completed. The second allocation, of 100 tracks, is also completed. The next 100-track allocation is, however, only partially used. The file is closed after writing on 350-1/2 tracks. At this time:

- A COBOL subroutine will format the rest of the 351st track. (Note that 351 tracks are actually relative tracks 0 through 350)
- The balance of 49 tracks will remain allocated but will not be formatted.

Note: In some of the foregoing cases, the number of tracks allocated to the file exceeds the number of tracks formatted by the COBOL subroutine. If the excess space was requested in track or block units, it should be released by specifying the RLSE option of the SPACE parameter.

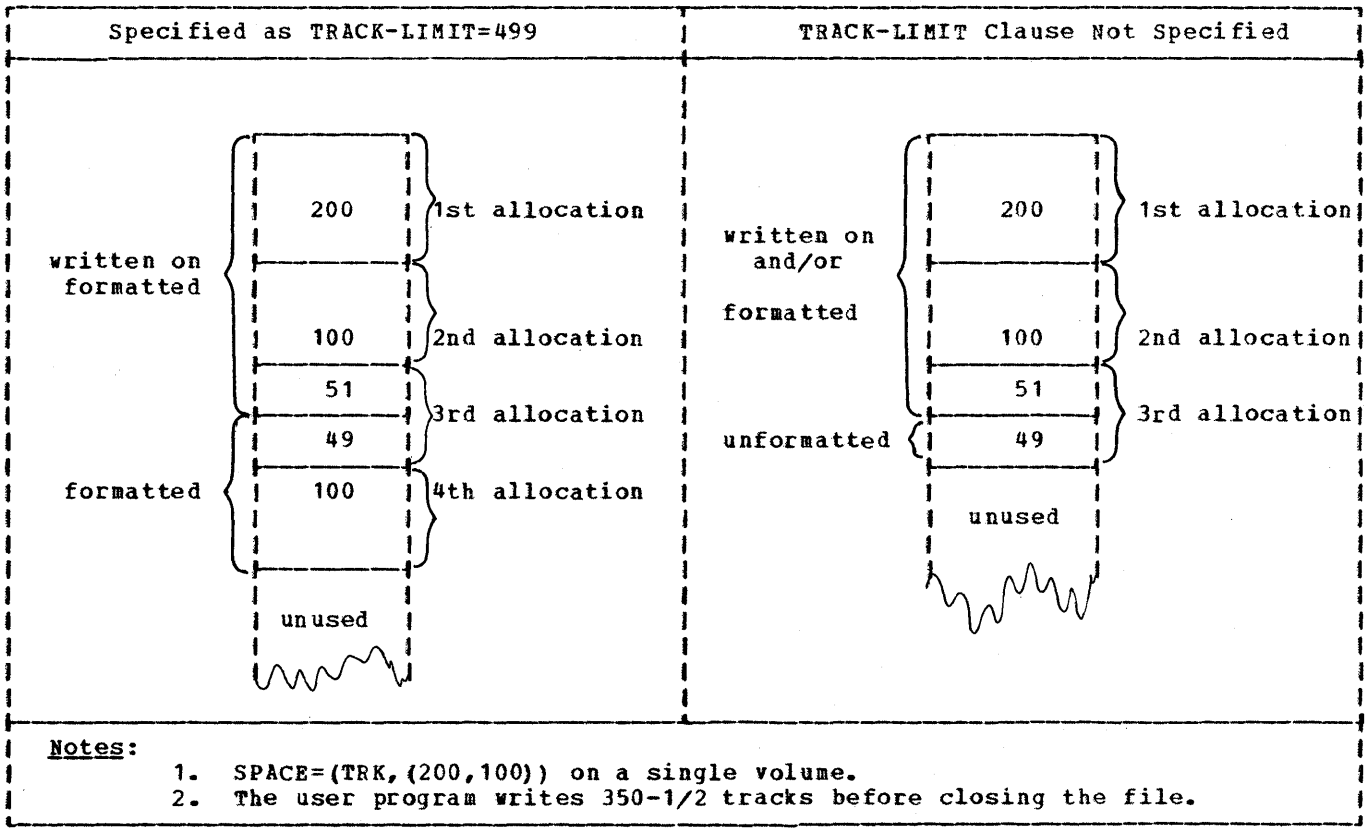


Figure 24. Sample Space Allocation for Sequentially Created Direct Files

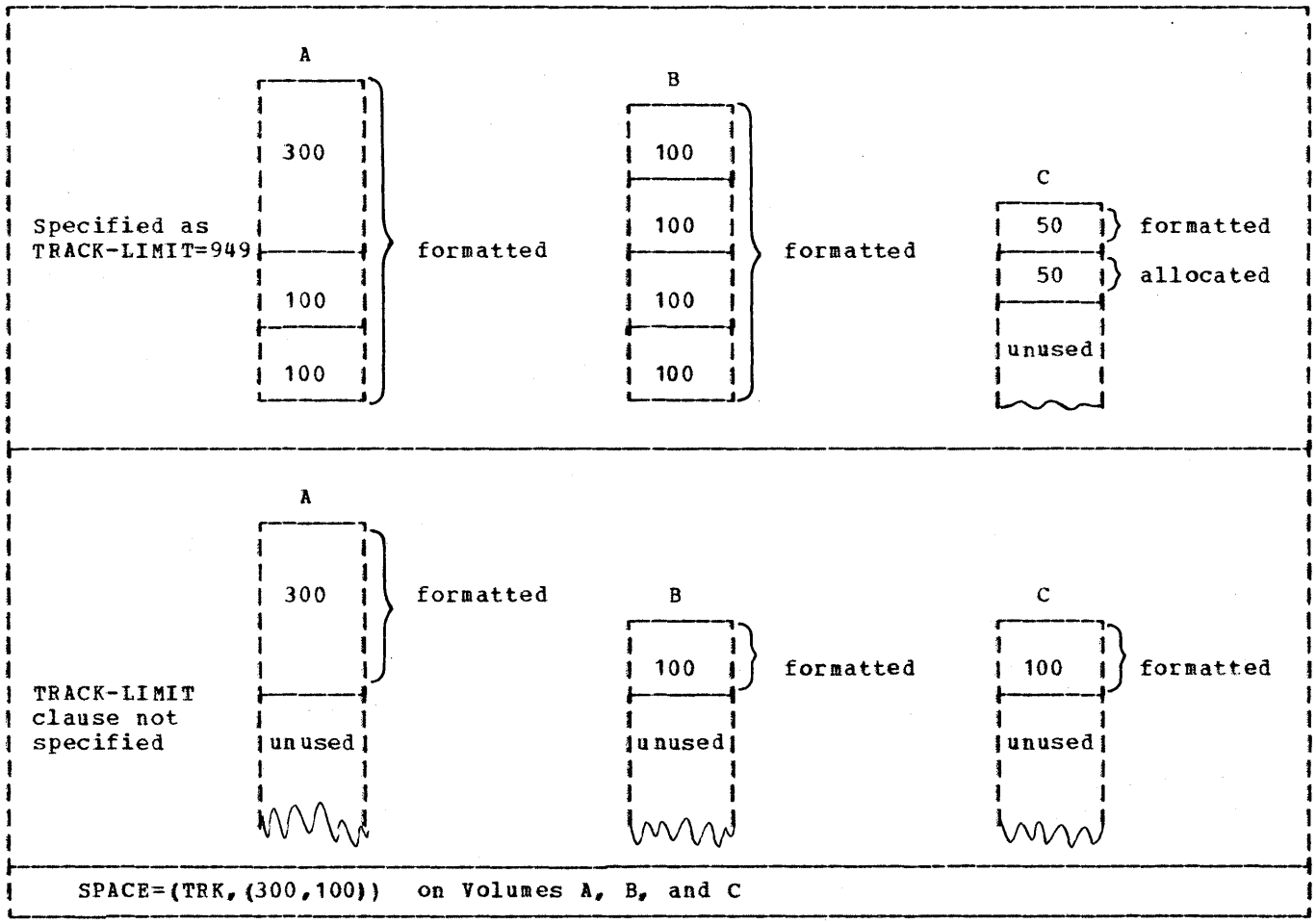


Figure 25. Sample Space Allocation for Randomly Created Direct Files

Random Creation of a Direct Data Set

The file processing technique used to create a direct file randomly is BDAM (Basic Direct Access Method).

- The associated COBOL statements are summarized in Figure 31.
- The associated JCL parameters are summarized in Figure 32.

Figure 30 (sample program) illustrates the random creation of a direct data set.

The ACTUAL KEY is required. When a direct file is created randomly, records need not be written in any particular sequence. The system seeks the track specified in the track identifier portion of the ACTUAL KEY and writes the record in the next available position on that track.

When a file is created using BDAM, the number of tracks specified in the primary extent must be available on the primary volume. If there are secondary volumes, one secondary extent must be available on each of the secondary volumes. If these extents are not available, the job will not begin execution. Once execution begins, the final allocation of space is determined by the TRACK-LIMIT clause and the SPACE and volume-count parameters of the DD card when the file is opened as an output file. Figure 25 illustrates the allocation and formatting of space when the TRACK-LIMIT clause is specified as well as when it is not specified (see "Dummy and Capacity Records" for a definition of allocate and format).

1. When a TRACK-LIMIT clause is specifed (Figure 25), the system will do the following:

   a. Allocate tracks, by blocks, until the quantity specified by the TRACK-LIMIT clause has been equaled or just exceeded.

   b. Format only the space specified in the TRACK-LIMIT clause, even if the space formatted is less than the space allocated.

2. When a TRACK-LIMIT clause is not specified (Figure 25), the first volume will be allocated and formatted according to the primary allocation quantity, and any succeeding volumes will be allocated and formatted from the secondary quantity, one quantity per volume.

Records cannot be written on those tracks that were allocated but unformatted. Any attempt to do so will have unpredictable results. Unformatted tracks can be released by specifying the RLSE option in the SPACE parameter on the corresponding DD statement. Only space requested in track or block units can be released. If the CYL subparameter was specified, the unformatted tracks cannot be released.

Unlike direct files created with BSAM, the BDAM processing technique allocates and formats tracks when the file is opened. This is significant because the system will not allocate secondary extents if the user attempts to write on more tracks than the quantity initially formatted.

Note: The extended search option may be used during random creation. See "Random Reading, Updating, and Adding to Direct Data Sets" for a detailed description.

Sequential Reading of Direct Data Sets

The file processing technique used to read a direct file sequentially is BSAM (Basic Sequential Access Method).

- The associated COBOL statements are summarized in Figure 31.
- The associated JCL parameters are summarized in Figure 32.

When a direct file is being read sequentially, records are retrieved in logical sequence. This logical sequence corresponds exactly to the physical sequence of the records on the mass storage device. Dummy records, if present, are also made available.

For reading a file sequentially, the ACTUAL KEY clause need not be specified; however:

- If the key is not specified, the user will have no way of distinguishing between real and dummy records (F-mode only). Dummy records can be recognized by testing for the presence of the figurative constant "HIGH VALUE" in the first position of the record identifier.
- If the ACTUAL KEY clause is specified, the record's key will be placed in the record identifier portion of the ACTUAL KEY during the execution of a READ statement. The track identifier, however, remains unchanged.

Random Reading, Updating, and Adding to Direct Data Sets

The file processing technique used to read, update, and add to a direct file randomly is BDAM (Basic Direct Access Method).

- The associated COBOL statements are summarized in Figure 31.
- The associated JCL parameters are summarized in Figure 32.

When records are being retrieved from a direct file randomly, the ACTUAL KEY is required to determine the track and to locate a particular record on that track. When a match is found, the data portion of the record is read. For an add operation, after locating the track, the system searches for the next available position on the track, and writes the new record. For an update operation, after locating the track, the system searches for the record specified in the record identifier portion of the ACTUAL KEY. (Note. A record in variable length BDAM files can be updated only with a record of the same length.)

In all of the foregoing cases, the specified track is the only one searched. If the desired record cannot be found, or room for an additional record cannot be found, the search terminates with an INVALID KEY condition. If the user wishes to extend the search to a specific number of tracks or to the entire file, the DCB OPTCD and LIMCT subparameters should be specified on the corresponding DD card. (Figure 30 illustrates the use of extended search.)

Multivolume Data Sets

Multivolume data sets, like single-volume data sets, may be created either randomly or sequentially.

Sequential Creation: When a file is created sequentially, the number of tracks specified in the primary extent must be available on the primary volume and the number of tracks specified in the secondary extent must be available on each of the secondary volumes. If extents are not available, execution of the job will not begin. Once execution begins, the primary, and as many secondary allocations as possible, are given to the first volume (up to 16 extents per volume). Subsequent volumes are allocated from the secondary specification.

If the CLOSE UNIT statement is executed, the current extent is formatted, volume switching procedures are executed, and the contents of ACTUAL KEY are updated to reflect the relative track number of the last track on the old volume. This is illustrated in the following example.

Consider the creation of a multivolume file whose space is allocated by:

```
SPACE=(TRK,(300,100))
```

1. When execution begins, the system allocates 300 tracks on the first volume. When the 300 tracks are used up, the system allocates 100 tracks more. Up to 16 allocations of 100 tracks each are possible.

2. If, after writing on 450 tracks, a CLOSE UNIT statement is executed, a COBOL subroutine will format the remaining 50 tracks of the current allocation before making the next unit available.

3. After the CLOSE UNIT statement is executed, a COBOL subroutine places the relative track number of the last track written on (for a data, dummy, or capacity record) in the track identifier of the ACTUAL KEY.

Note: A CLOSE UNIT statement always formats the tracks remaining on that unit from the current allocation. The formatting of tracks on the last unit, when a CLOSE file-name statement is executed, depends on the presence or absence of a TRACK-LIMIT clause, just as it did for single-volume files (see "Space Allocated for Single- Volume Files"). The RLSE option of the SPACE parameter applies only to the unformatted tracks at the end of the last unit.

Automatic Volume Switching: The user may choose to permit volume switching to occur automatically. This can be accomplished by writing on all allocated tracks until no more are available, or may be made available. This procedure, however, does not guarantee a specific distribution of records over the volumes, the placement of a particular record on a particular volume, or whether the data set is, in fact, multivolume.

Note: If the user permits system controlled volume switching, but specifies the file be created on more than one volume [e.g., VOL=SER=(V1,V2,V3)]; the system may write the entire file on the primary volume if there is enough room. The next time an attempt is made to open that file, since the system expects it to reside on three volumes, an ABEND will occur. This can be avoided by specifying:

```
VOL=(,,,3,SER=(V1,V2,V3))
```

This specifies the file be contained on one or more volumes.

To create a file with records distributed as evenly as posible over several volumes, the programmer must calculate the amount of space his file will require (see "Determination of File Space") and divide by the number of volumes. The result of this calculation (rounded) should be specified as both the primary and secondary allocation of the SPACE parameter of the associated DD statement. The programmer should execute CLOSE UNIT before the end of the initially allocated space on the first volume (that is, execute the CLOSE UNIT before writing the record that is to be first on the second volume).

For example, to distribute 2232 132-byte records as evenly as possible on two 2314 volumes, 37 tracks per volume are required and the SPACE parameter should specify (37,37). After writing the 1116th record the programmer should execute CLOSE UNIT and continue writing.

If the required space is overestimated and the records do not fill the last track(s), the compiler will write dummy records to complete them. These records are included in the record count and should be taken into account when trying to address records on subsequent volumes.

If the space required is underestimated, automatic volume switching may occur before the CLOSE UNIT is executed since space on the first volume is filled. If this has happened, the CLOSE UNIT starts a third volume.

If no secondary allocation has been specified and the program issues a CLOSE UNIT statement, the job will terminate abnormally, since the allocation of subsequent volumes is taken from the secondary allocation field of the SPACE parameter.

In the creation of an output file, performance is improved by specifying the CONTIG subparameter of the SPACE parameter in the DD statement. However, space allocation is more efficient if CONTIG is not specified.

Random Creation: When a file is created randomly, space allocation and formatting is done as described in "Random Creation of a Direct Data Set" (Figure 25). It is important to note that a CLOSE UNIT statement is not permitted when creating a file randomly.

The following description pertains to Figure 25:

1. When the TRACK-LIMIT clause is specified, the total extent of the file is 950 tracks. The only valid track identifiers are 0 through 949:
   - Tracks 000 through 499 are contained on volume A.
   - Tracks 500 through 899 are contained on volume B.
   - Tracks 900 through 949 are contained on volume C.
2. When the TRACK-LIMIT clause is not specified, the total extent of the file is 500 tracks. The only valid track identifiers are 0 through 499:
   - Tracks 000 through 299 are contained on volume A.
   - Tracks 300 through 399 are contained on volume B.
   - Tracks 400 through 499 are contained on volume C.

File Organization Field of the System-Name

The single character "D" or "W", specifying the file organization, must be coded as part of the system-name. The user should be aware of the following differences:

- Sequentially accessed files must specify organization "D".
- Randomly accessed files may specify "D" or "W". When opened input or output "D" and "W" function identically.

1. Opened output ("D" and "W"):

   WRITE adds a new record. If a record containing the same key already exists, the system will add the record anyway. The result will be records with duplicate keys.

2. Opened I-O ("W"):

   a. REWRITE automatically searches for a record with a matching record identifier, and updates it.

   b. WRITE adds a new record to the file whether or not a duplicate key already exists.

3. Opened I-O ("D"):

   a. REWRITE updates the file only if the preceding input/output statement was a READ of the same record.

   b. WRITE adds a new record to the file, whether or not a duplicate key already exists, if the preceding input/output statement was anything other than a READ of the same record.

Note: When a file is opened I-O (BDAM "D") the contents of ACTUAL KEY are moved to a save area during the execution of a READ statement. During the execution of a WRITE statement, the contents of ACTUAL KEY are compared to the contents of the save area to determine whether the system should add or update a record. A check is also made to assure that the preceding input/output statement was a READ. If it was a WRITE of any record, a new record is added to the file. Opening a file I-O (BDAM "W") omits the save and compare steps entirely. The system adds a record when a WRITE statement is executed and updates a record when a REWRITE statement is executed. It is, therefore, more efficient to use BDAM "W" than it is to use BDAM "D" if it is known in advance whether the record should be added or updated.

Determination of File Space: To determine the amount of space a data set requires, the following variables should be considered:

Device Type
Track Capacity
Tracks per Volume
Cylinders per Volume
Data length (block size)
Key Length
Device Overhead

Device overhead refers to the space required on each track for hardware data, i.e., address markers, count areas, inter-record gaps, Record 0, etc. Device overhead varies with each device and also depends on whether the blocks are written with keys. The formulas in Figure 26 may be used to compute the actual space required for each block, including device overhead.

Figure 27 lists device storage capacity, and Figure 28 lists capacity in records per track for several mass storage devices.

Programmers who require more detailed information on mass storage devices may refer to the IBM System/370 System Summary, Order No. GA22-7001.

Note: Specification of the "S" option in the DCB subparameter RECFM can increase 3330 performance (see the description of RECFM earlier in this chapter).

Randomizing Techniques

One method of determining the value of the track identifier portion of the ACTUAL KEY is called indirect addressing. Indirect addressing generally is used when the range of keys for a file includes a high percentage of unused values. For example, employee numbers may range from 000001 to 009999, but only 3000 of the possible 9999 numbers are currently assigned. Indirect addressing can also be used with nonnumeric keys. A nonnumeric field (e.g., alphanumeric), when moved to a computational field, will be packed and then converted to binary notation. Since packing eliminates the zone fields, the final binary item will be numeric.

Indirect addressing means that the key is converted to a value for the track identifier by use of some algorithm intended to limit the range of addresses. Such an algorithm is called a randomizing technique. Randomizing techniques need not produce a unique address for every record; in fact, such techniques usually produce synonyms. Synonyms are records whose keys randomize to the same address.

Two objectives must be considered in selecting a randomizing technique:

1. Every possible key in the file must randomize to an address within the designated range.
2. The addresses should be distributed evenly across the range so that there are as few synonyms as possible.

Note that one way to minimize synonyms is to allocate more space for the file than is actually required to hold all the records. For example, the percentage of locations actually used might comprise only 80 to 85 percent of the allotted space.

Division/Remainder Method: One of the simplest ways to address a directly organized file indirectly is to use the division/remainder method.

1. Determine the amount of locations required to contain the data file. Include a packing factor for additional space to eliminate synonyms. The packing factor should be approximately 20 percent of the total space allotted to contain the data file.

2. Select the nearest prime number that is less than the total of step 1. A prime number is a number divisible only by itself and the integer 1. Figure 29 is a partial list of prime numbers.

3. Clear any zones from the key that is to be used to calculate the track identifier of actual key. This can be accomplished by moving the key to a field described as COMPUTATIONAL.

4. Divide the key by the prime number selected.

5. Ignore the quotient; utilize the remainder as the relative location within the data file.

For example, assume that a company is planning to create an inventory file on a 2314 disk storage device. There are 8,000 different inventory parts, each identified by an 8-character part number. Using a 20 percent packing factor, 10,000 record positions are allocated to store the data file.

Method A: The closest prime number to 10,000, but under 10,000, is 9973. Using one inventory part number as an example, in this case #25DF3514, and clearing the zones, we have 25463514. Dividing by 9973 a quotient of 2553 results in a remainder of 2445. Thus, 2445 is the relative location of the record within the data file corresponding to part number 25DF3514. The record address can be determined from the relative location as follows:

1. Determine the number of records that can be stored on a track (e.g., 13 per track on a 2314, assuming each inventory record is 200-bytes long).

   Note: Because each data record has nondata components, such as a count area and inter-record gaps, track capacity for data storage will vary with record length. As the number of separate records on a track increases, inter-record gaps occupy additional byte positions so that data capacity is reduced. Track capacity formulas provide the means to determine total byte requirements for records of various sizes on a track (see Figures 26-28).

2. Divide the relative number (2445) by the number of records to be stored on each track.

3. The result, quotient = 188, now becomes the track identifier of the actual key.

Method B: Utilizing the same example, another approach will also provide the relative track address. Method B is illustrated in Figure 30:

1. The number of records that may be contained on one track is 13. Therefore, if 10,000 record locations are to be provided, 770 tracks must be reserved.

2. The prime number nearest, but less than 770, is 691.

3. Divide the zone-stripped key by the prime value. (In the example, 25463514 divided by 691 provides a quotient of 36850 and a remainder of 164. The remainder is the track identifier.)

Device Type	Bytes Required by Each Data Block			
	Blocks With Keys		Blocks Without Keys	
	Bi	Bn	Bi	Bn
2314 (2319)	146+1.043(KL+DL)	45+KL+DL	101+1.043(DL)	DL
2305-1	634+KL+DL	634+KL+DL	432+DL	432+DL
2305-2	289+KL+DL	289+KL+DL	198+DL	198+DL
3330-1,-2, 3330-11	191+KL+DL	191+KL+DL	135+DL	135+DL
3340	242+KL+DL	242+KL+DL	167+DL	167+DL
3350	267+KL+DL	267+KL+DL	185+DL	185+DL

Bi is any block but the last on the track.
Bn is the last block on the track.
DL is data length.
KL is key length.

Figure 26. Mass Storage Device Overhead Formulas

Device Type	Volume Type	Track Capacity	Tracks per Cylinder	Number of Cylinders	Total Capacity
2314 (2319)	Disk	7294	20	200	29,176,000
2305-1	Disk	14136	8	48	5,428,224
2305-2	Disk	14660	8	96	11,258,880
3330-1,-2	Disk	13030	19	404	101,751,270
3330-11	Disk	13030	19	808	203,502,340
3340	Disk	8535	12	348	34,944,768
3350	Disk	19069	30	555	317,498,850

Notes:
Capacity is indicated in bytes. For disk devices, total capacity is the number of bytes per demountable pack.

Figure 27. Mass Storage Device Capacities

Maximum Bytes per Record Formatted Without Keys						Records per Track	Maximum Bytes per Record Formatted With Keys					
2305-1	2305-2	2314 (2319)	3330-1 (3330-11)	3340	3350		2305-1	2305-2	2314 (2319)	3330-1 (3330-11)	3340	3350
14136	14660	7294	13030	8368	19069	1	13934	14569	7249	12974	8273	18987
6852	7231	3520	6447	4100	9442	2	6650	7140	3476	6391	4005	9360
4424	4754	2298	4253	2678	6233	3	4222	4663	2254	4197	2583	6151
3210	3516	1693	3156	1966	4628	4	3008	3425	1649	3100	1871	4546
2480	2773	1332	2498	1540	3665	5	2278	2682	1288	2442	1445	3583
1996	2278	1092	2059	1255	3024	6	1794	2187	1049	2003	1160	2942
1648	1924	921	1745	1052	2565	7	1446	1833	877	1689	957	2483
1388	1659	793	1510	899	2221	8	1186	1568	750	1454	804	2139
1186	1452	694	1327	781	1954	9	984	1361	650	1271	686	1872
1024	1287	615	1181	686	1740	10	822	1196	571	1125	591	1658
892	1152	550	1061	608	1565	11	690	1061	506	1005	513	1483
782	1040	496	962	544	1419	12	580	949	452	906	449	1337
688	944	450	877	489	1296	13	486	853	407	821	394	1214
608	863	411	805	442	1190	14	406	772	368	749	347	1108
538	792	377	742	402	1098	15	336	701	333	686	307	1016
478	730	347	687	366	1018	16	276	639	304	631	271	936
424	676	321	639	335	947	17	222	585	277	583	240	865
376	627	298	596	307	884	18	174	536	254	540	212	802
334	584	276	557	282	828	19	132	493	233	501	187	746
296	544	258	523	259	777	20	94	453	215	467	164	695
260	509	241	491	239	731	21	58	418	198	435	144	649
230	477	226	463	220	690	22		386	183	407	125	608
200	448	211	437	204	652	23		357	168	381	109	570
174	421	199	413	188	617	24		330	156	357	93	535
150	396	187	391	174	585	25		305	144	335	79	503
128	373	176	371	161	555	26		282	133	315	66	473
106	352	166	352	149	528	27		261	123	296	54	446
88	332	157	335	137	502	28		241	114	279	42	420
70	314	148	318	127	478	29		223	105	262	32	396
52	297	139	303	117	456	30		206	96	247	22	374

Figure 28. Mass Storage Device Track Capacity

Number	Nearest Prime Number Less than Number
500	499
600	599
700	691
800	797
900	887
1000	997
1100	1097
1200	1193
1300	1297
1400	1399
1500	1499
1600	1597
1700	1699
1800	1789
1900	1889
2000	1999
2100	2099
2200	2179
2300	2297
2400	2399
2500	2477
2600	2593
2700	2699
2800	2797
2900	2897
3000	2999
3100	3089
3200	3191
3300	3299
3400	3391
3500	3499
3600	3593
3700	3697
3800	3797
3900	3889
4000	3989
4100	4099
4200	4177
4300	4297
4400	4397
4500	4493
4600	4597
4700	4691
4800	4799
4900	4889
5000	4999
5100	5099
5200	5197
5300	5297
5400	5399
5500	5483
5600	5591
5700	5693
5800	5791
5900	5897

Figure 29. Partial List of Prime Numbers (Part 1 of 2)

Number	Nearest Prime Number Less than Number
6000	5987
6100	6091
6200	6199
6300	6299
6400	6397
6500	6491
6600	6599
6700	6691
6800	6793
6900	6899
7000	6997
7100	7079
7200	7193
7300	7297
7400	7393
7500	7499
7600	7591
7700	7699
7800	7793
7900	7883
8000	7993
8100	8093
8200	8191
8300	8297
8400	8389
8500	8467
8600	8599
8700	8699
8800	8793
8900	8893
9000	8999
9100	9091
9200	9199
9300	9293
9400	9397
9500	9497
9600	9587
9700	9697
9800	9791
9900	9887
10,000	9973
10,100	10,099
10,200	10,193
10,300	10,289
10,400	10,399
10,500	10,499
10,600	10,597

Figure 29. Partial List of Prime Numbers (Part 2 of 2)

```
00001   00101  IDENTIFICATION DIVISION.
00002   00102  PROGRAM-ID.  METHOD B.
00003   00103  ENVIRONMENT DIVISION.
00004   00104  CONFIGURATION SECTION.
00005   00105  SOURCE-COMPUTER. IBM-370.
00006   00106  OBJECT-COMPUTER.  IBM-370.
00007   00107  INPUT-OUTPUT SECTION.
00008   00108  FILE-CONTROL.
00009   00109      SELECT D-FILE ASSIGN DA-2314-D-MASTER
00010   00110      ACCESS IS RANDOM ACTUAL KEY IS ACT-KEY
00011   00112      TRACK-LIMIT IS 691.<-------------------------(2)
00012   00113      SELECT C-FILE ASSIGN UT-S-CARDS.
00013   00114  DATA DIVISION.
00014   00115  FILE SECTION.
00015   00116  FD  D-FILE
00016   00117      LABEL RECORDS ARE STANDARD.
00017   00118  01  D-REC.
00018   00119      02  PART-NUM PIC X(8).
00019   00120      02  NUM-ON-HAND PIC 9(4).
00020   00121      02  PRICE  PIC 9(5)V99.
00021   00122      02  FILLER PIC X(181).
00022   00201  FD  C-FILE
00023   00202      LABEL RECORDS ARE OMITTED.
00024   00203  01  C-REC.
00025   00204      02  PART-NUM PIC X(8).
00026   00205      02  NUM-ON-HAND  PIC 9(4).
00027   00206      02  PRICE PIC 9(5)V99.
00028              02  FILLER PIC X(61).
00029   00207  WORKING-STORAGE SECTION.
00030   00209  77  SAVE  PIC S9(8) COMP SYNC.
00031   00210  77  QUOTIENT  PIC S9(5) COMP SYNC.
00032   00211  01  ACT-KEY.
00033   00212      02  TRACK-ID  PIC S9(5) COMP SYNC.
00034   00213      02  REC-ID    PIC X(8).
00035   00214  PROCEDURE DIVISION.
00036                  OPEN INPUT C-FILE OUTPUT D-FILE.
00037   00303  READS.
00038   00304      READ C-FILE AT END GO TO EOJ.
00039   00305      MOVE CORRESPONDING C-REC TO D-REC.
00040   00306      MOVE PART-NUM OF C-REC TO REC-ID SAVE.
00041   00307      DIVIDE SAVE BY 691 GIVING QUOTIENT REMAINDER TRACK-ID.<-(1)
00042   00308  WRITES.
00043   00309      EXHIBIT NAMED TRACK-ID C-REC.
00044   00310      WRITE D-REC INVALID KEY GO TO INVALID-KEY.
00045   00311      GO TO READS.
00046   00312  INVALID-KEY.
00047   00313      DISPLAY 'INVALID KEY  ' TRACK-ID REC-ID.
00048   00314  EOJ.
00049   00315      CLOSE C-FILE D-FILE.
00050   00316      STOP RUN.
```

Figure 30. Sample Program for a Randomly Created Direct File (Part 1 of 2)

```
000010  IKF8003I-W     RANDOM OPTION OF ACCESS MODE IS CLAUSE IS AN EXTENSION TO LEVEL A.
000010  IKF8003I-W     ACTUAL KEY IS CLAUSE IS AN EXTENSION TO LEVEL A.
000011  IKF8002I-W     TRACK-LIMIT  CLAUSE IN SELECT SENTENCE IS AN EXTENSION TO ALL LEVELS.
000039  IKF8003I-W     CORRESPONDING OPTION IS AN EXTENSION TO LEVEL A.
000040  IKF8003I-W     QUALIFICATION OF DATA-NAMES AND PARAGRAPH-NAMES IS AN EXTENSION TO LEVEL A.
000041  IKF8003I-W     REMAINDER IN DIVIDE STATEMENT IS AN EXTENSION TO LEVEL A.
000043  IKF8002I-W     EXHIBIT STATEMENT IS AN EXTENSION TO ALL LEVELS.
000047  IKF8002I-W     APOSTROPHE USED AS QUOTE IS AN EXTENSION TO ALL LEVELS.
000049  IKF8003I-W     USE OF MULTIPLE FILE-NAMES IN CLOSE STATEMENT IS AN EXTENSION TO LEVEL A.
```

FIPS messages for LVL=A

```
XXGO     EXEC PGM=*.LKED.SYSLMOD,COND=((5,LT,COB),(5,LT,LKED))          00800270
XXSTEPLIB  DD DSN=VSCBL1.LIB,DISP=SHR,UNIT=2314,VOL=SER=DB143           00800280
XXDD1      DD DSN=&SYMDBG,DISP=(OLD,DELETE)                             00800290
XXSYSDBOUT DD SYSOUT=A                                                  00800300
//GO.SYSUDUMP  DD  SYSOUT=G
X/SYSUDUMP DD SYSOUT=A                                                  00800310
//GO.SYSOUT  DD  SYSOUT=G
X/SYSOUT   DD SYSOUT=A                                                  00800320
XXSYSPUNCH DD SYSOUT=E                                                  00800330
//GO.MASTER  DD  SPACE=(TRK,(500,100),RLSE),                           X
//             DCB=(OPTCD=E,LIMCT=5),UNIT=2314
//GO.CARDS  DD  *
//
IEF236I ALLOC. FOR FIG18    GO       STEP1
IEF237I 230   ALLOCATED TO PGM=*.DD
IEF237I 234   ALLOCATED TO STEPLIB
IEF237I 250   ALLOCATED TO DD1
IEF237I 230   ALLOCATED TO SYSDBOUT
IEF237I 250   ALLOCATED TO SYSUDUMP
IEF237I 230   ALLOCATED TO SYSOUT
IEF237I 250   ALLOCATED TO SYSPUNCH
IEF237I 235   ALLOCATED TO MASTER
IEF237I 230   ALLOCATED TO CARDS

TRACK-ID = 00149 C-REC = 82900801CD1
TRACK-ID = 00149 C-REC = 82900801CD2
TRACK-ID = 00149 C-REC = 82900801CD3
TRACK-ID = 00149 C-REC = 82900801CD4
TRACK-ID = 00070 C-REC = 82900031
TRACK-ID = 00149 C-REC = 82900801CD5
TRACK-ID = 00149 C-REC = 82900801CD6
TRACK-ID = 00149 C-REC = 82900801CD7
TRACK-ID = 00149 C-REC = 82900801CD8
TRACK-ID = 00149 C-REC = 82900801CD9
TRACK-ID = 00149 C-REC = 82900801CD10
TRACK-ID = 00149 C-REC = 82900801CD11
TRACK-ID = 00149 C-REC = 82900801CD12
TRACK-ID = 00149 C-REC = 82900801CD13
TRACK-ID = 00149 C-REC = 82900801CD14
TRACK-ID = 00149 C-REC = 82900801CD15
TRACK-ID = 00149 C-REC = 82900801CD16
TRACK-ID = 00039 C-REC = 829000003
TRACK-ID = 00149 C-REC = 82900801CD17
TRACK-ID = 00149 C-REC = 82900801CD18
TRACK-ID = 00149 C-REC = 82900801CD19
TRACK-ID = 00149 C-REC = 82900801CD20
TRACK-ID = 00157 C-REC = 82900809
TRACK-ID = 00149 C-REC = 82900801CD21
TRACK-ID = 00149 C-REC = 82900801CD22
TRACK-ID = 00149 C-REC = 82900801CD223
TRACK-ID = 00149 C-REC = 82900801CD24
TRACK-ID = 00149 C-REC = 82900801CD25
TRACK-ID = 00149 C-REC = 82900801CD26
```

Figure 30. Sample Program for a Randomly Created Direct File (Part 2 of 2)

Figure 30 is a sample COBOL program that creates a direct file using method B (see "Randomizing Technique") and provides for the possibility of synonym overflow. Synonym overflow will occur if a record randomizes to a track that is already full. The following discussion highlights some basic features. Circled numbers in the program example refer to corresponding numbers in the text that follows.

1. Since this randomizing technique ① employs the prime number 691 as its divisor, the largest possible remainder is 690. By the interaction between the TRACK-LIMIT clause ② and the SPACE parameter ③ , the program formats 692 tracks (i.e., relative tracks 000-691). This establishes track 691 as the only track that can contain synonym overflow from track 690.

2. The DCB subparameter ④ OPTCD=E is specified. If a synonym overflow condition arises, an extended search will be employed, and the additional record will be written in the first available position on the following track(s).

3. The DCB subparameter ⑤ LIMCT=5 is specified. This limits the extended search to five tracks. If no room is found within this limit, an invalid key condition results. A value should always be specified for the LIMCT subparameter when OPTCD=E is indicated. Otherwise, the default value of LIMCT, which is zero, will result in an error that will be treated as an exceptional input/output condition.

Note: The randomizing technique chosen should minimize the number of synonym overflows for two reasons:

1. The more extended search is employed during file creation, the more it will be required during record retrieval. Extended searches increase access time proportionately.

2. When an extended search is employed, the adjusted value of the track identifier is not made available to the user after the execution of a WRITE statement. The user, therefore, has no way of knowing the track on which an overflow record is actually written.

File Organization	Data Management Techniques	Access Method	KEY Clauses	OPEN Statement	Access Verbs	CLOSE Statement
D	BSAM	SEQUENTIAL	ACTUAL	INPUT	READ[INTO] AT END	[UNIT] [WITH LOCK]
				OUTPUT	WRITE[FROM] INVALID KEY	
D	BDAM	RANDOM	ACTUAL	INPUT	SEEK READ[INTO] INVALID KEY	[WITH LOCK]
				OUTPUT	SEEK WRITE[FROM] INVALID KEY	
				I-O	SEEK READ[INTO] INVALID KEY WRITE[FROM] INVALID KEY	
W	BDAM	RANDOM	ACTUAL	I-O	SEEK READ[INTO] INVALID KEY WRITE[FROM] INVALID KEY REWRITE[FROM] INVALID KEY	[WITH LOCK]

Figure 31. Direct File Processing on Mass Storage Devices

DD Statement Parameters Applicable to BSAM Input Files

DSNAME	Device	UNIT VOLUME	LABEL	SPACE	SUBALLOC	SPLIT	DISP	DCB
as	Mass Storage required	not required if cataloged	[SL or SUL]	as	na	na	{OLD / SHR} {,PASS / ,KEEP / ,CATLG / ,DELETE / ,UNCATLG}	na

DD Statement Parameters Applicable to BSAM Output Files

DSNAME	Device	UNIT VOLUME	LABEL	SPACE	SUBALLOC	SPLIT	DISP	DCB
as	Mass Storage required	as	[SL or SUL]	as RLSE	as	na	NEW {,KEEP / ,CATLG / ,PASS / ,DELETE} Note: MOD not meaningful	[DSORG=DA] OPTCD=[W,T]

DD Statement Parameters Applicable to BDAM Input and I-O Files

DSNAME	Device	UNIT VOLUME	LABEL	SPACE	SUBALLOC	SPLIT	DISP	DCB
as	Mass Storage required	not required if cataloged	[SL or SUL]	na	na	na	{OLD / SHR} {,PASS / ,KEEP / ,CATLG / ,UNCATLG / ,DELETE}	as specified at file creation

DD Statement Parameters Applicable to BDAM Output Files

DSNAME	Device	UNIT VOLUME	LABEL	SPACE	SUBALLOC	SPLIT	DISP	DCB
as	Mass Storage required	as	[SL or SUB]	as RLSE	as	na	NEW {,KEEP / ,CATLG / ,PASS / ,DELETE} Note: MOD not meaningful	[DSORG=DA] OPTCD=[W,E] LIMCT=n

as = Applicable subparameters
na = Not applicable

Figure 32. JCL Applicable to Directly Organized Files

RELATIVE FILE PROCESSING

Relative file processing is characterized by the use of the relative record addressing scheme. When this addressing scheme is used, the position of the logical records in a file is determined relative to the first record of the file starting with the initial value of zero. A NOMINAL KEY is used to identify randomly accessed records. Files with relative data organization must be assigned to mass storage devices.

Format
NOMINAL KEY IS data-name

Data-name must be defined as an 8-integer binary item whose value must not exceed 16,777,215. NOMINAL KEY must be defined in the Working-Storage Section.

The following example illustrates use of the NOMINAL KEY clause:

```
ENVIRONMENT DIVISION.
        .
        .
        .
    NOMINAL KEY IS THE-NOMINAL-KEY.
        .
        .
        .
DATA DIVISION.
        .
        .
        .
WORKING-STORAGE SECTION.
77  THE-NOMINAL-KEY PIC S9(8) COMP SYNC.
```

The relative file processing technique supports only unblocked fixed-length records.

Figure 33 illustrates those parts of a relatively organized file that are of importance to a COBOL programmer. The track format is similar to the format described for directly organized files (see section "Direct File Processing"). The following is a list of significant differences:

1. The records (R1, R2, ..., Rn) are formatted without a key area.
2. The COUNT area contains a record ID:
   a. 2 bytes containing the cylinder number.
   b. 2 bytes containing the read/write head.
   c. 1 byte containing a record number from 1 through 255.

Records on mass storage devices will always appear sequentially ranging from 0 to n, where n equals the highest key contained in the file.

The following example illustrates the relationship between the NOMINAL KEY and the positioning of records on a mass storage device.

```
ENVIRONMENT DIVISION.
        .
        .
        .
    NOMINAL KEY IS THE-NOMINAL-KEY.
        .
        .
        .
DATA DIVISION.
FILE SECTION.
FD  RELATIVE-FILE
    LABEL RECORDS ARE STANDARD.
01  REC-1              PIC  X(80).
        .
        .
        .
WORKING-STORAGE SECTION.
        .
        .
        .
77  THE-NOMINAL-KEY  PIC S9(8) COMP SYNC.
```

Consider REC-1 being written 200 times. With each execution of the WRITE statement, the content of THE-NOMINAL-KEY is incremented by 1, from 0 through 199. Since a 2314 mass storage device has room for only thirty nine 80-character records on each track (see "Determination of File Space" in "Direct File Processing") REC-1 will be written as follows:

- Relative records 0 through 38 will be on the first track.
- Relative records 39 through 199 will be on the second through sixth tracks.

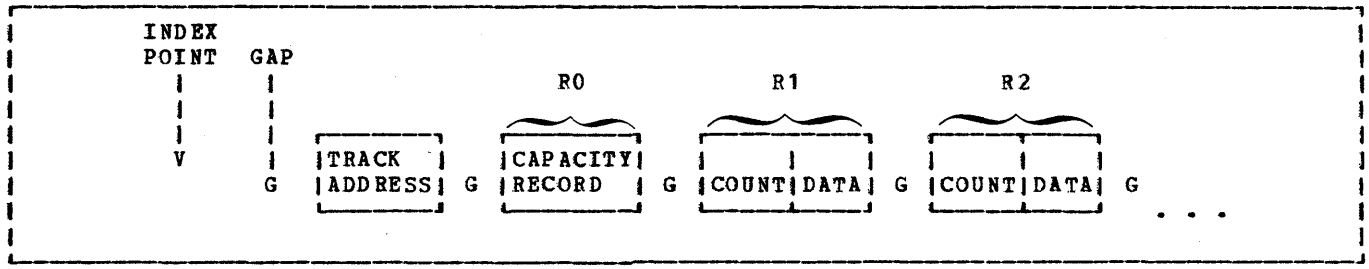


Figure 33. Relatively Organized Data as it Appears on a Mass Storage Device

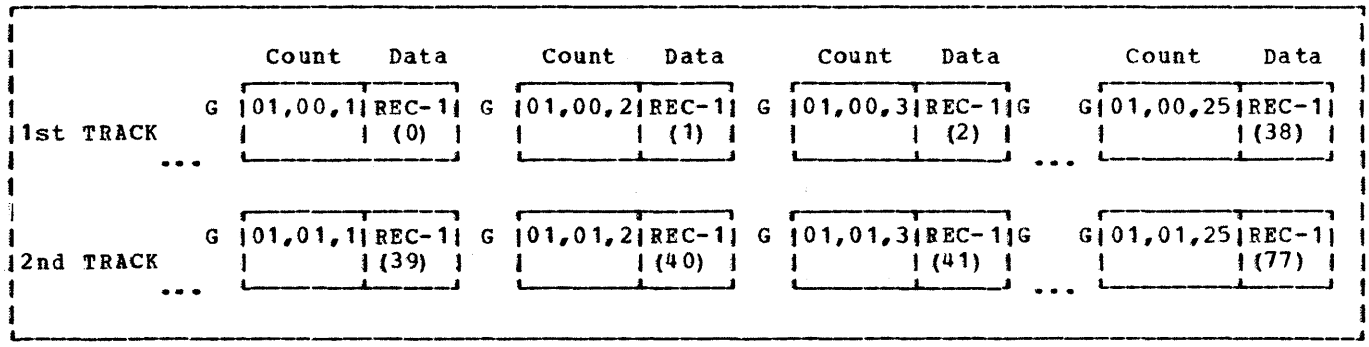


Figure 34. Sample Format of Two Tracks of a Relative File

If the two tracks assigned to RELATIVE FILE are "cylinder 01 track 00" and "cylinder 01 track 01," they would appear as shown in Figure 33.

It is important to note that information about the length of each record, the capacity of each track and the relative record number, as indicated by the NOMINAL KEY is used by the system to determine the exact location of each record. As indicated in Figure 34, the system converts each relative record number into a unique cylinder number, head number, and record number, which are written in the count area of each physical record.

Note: Since count areas do not appear in I-O buffers and there are no key areas, buffer size need be only large enough to accommodate data in REC-1.

Sequential Creation

Relative files must be created sequentially using the file processing technique BSAM (Basic Sequential Access Method).

- The associated COBOL statements are summarized in Figure 36.
- The associated JCL statements are summarized in Figure 37.

Figure 35 illustrates the creation of a relative data set.

Records in relative files, are arranged sequentially in the order in which they were written. The first record written is relative record 0, the second record is relative record 1, the nth record written is relative record n-1. A file containing 1000 records will thus contain relative records 0 through 999. The clause that allows the user to specify the relative record needed is the NOMINAL KEY clause.

When a relative file is being created, the NOMINAL KEY clause may be specified.

- If the NOMINAL KEY is specified and the value in the NOMINAL KEY (when a WRITE statement is executed) is greater than the next sequential relative number, the necessary number of dummy records is written by the compiler so that the actual record is written in the specified relative position. If the NOMINAL KEY for a WRITE statement is

less than the next sequential relative record number, the key is ignored and the record is written in the next available position.

- If the NOMINAL KEY is not specified, the system begins writing at relative record 0 and increments the relative record number by 1 for each additional WRITE statement. When the key is not specified, the user is responsible for insertion of dummy records. The only time the compiler will add dummy records is during the execution of a CLOSE or CLOSE UNIT statement.

  Note: Dummy records are identified by the presence of the figurative constant HIGH-VALUE in the first position of the record.

The relative block number of the last record written is placed in the NOMINAL KEY after a WRITE, CLOSE, or CLOSE UNIT statement, if the key is specified.

Once a file is created, more space cannot be allocated and the extent of the file cannot be increased. The only way to add records to an already existing file is to replace dummy records. Therefore, to allow for future additions, the user should create the file with as many excess dummy records as desired.

The allocation of space to a relative file (both single-volume and multivolume) is similar to the allocation of space described for a sequentially created direct file. Highlights and essential differences are discussed below:

- The relative file processing technique does not include a TRACK-LIMIT clause. Space allocation and formatting will, therefore, be determined by an interaction between the SPACE parameter of the DD card and the number of records written.

- The total number of tracks formatted will be determined when the file is closed. Dummy records will be added to complete the current track, if necessary.

- Tracks that are allocated but unformatted, and have been requested in track or block units, can be released by specifying the RLSE subparameter on the DD statement.

- When a unit of a multivolume file is closed, all tracks that have been allocated on the current unit are formatted (initialized with dummy records) before the next unit is made available. The RLSE subparameter of the DD statement applies only to the allocated tracks at the end of a data set.

Note: In order to determine the amount of space a data set requires, see Figures 26-28.

Sequential Reading

The file processing technique used to read a relative file sequentially is BSAM (Basic Sequential Access Method).

- The associated COBOL statements are summarized in Figure 36.

- The associated JCL parameters are summarized in Figure 37.

When a relative file is being read sequentially, the records are made available in the sequence in which the records were written. Dummy records are also made available. The NOMINAL KEY, if specified, will be ignored.

Random Access

The file processing technique used to read or update a relative file randomly is BDAM (Basic Direct Access Method).

- The associated COBOL statements are summarized in Figure 36.

- The associated JCL statements are summarized in Figure 37.

Since a relative file cannot be created randomly, the following restrictions exist:

1. The file cannot be opened as an output file.

2. The WRITE verb is not permitted.

A relative file with BDAM can be opened as input or I-O. Records are made available according to the contents of NOMINAL KEY. If the user wishes to update a file, it must be opened as I-O. Records can then be read into a single buffer, updated in that buffer, and rewritten from that buffer. If the user wishes to add records to a file, the file must have been created with excess dummy records. If dummy records are present, the file can be

opened as I-O and dummy records can be replaced by the additions. If dummy records are not present, additions cannot be made.

Note: Records cannot be deleted, but can be replaced by dummy records.

Figure 35 illustrates several basic characteristics of the relative file processing technique. It creates a relative file (R-FILE) using a card file (C-FILE) as input. C-FILE consists of 11 cards in the following sequence:

Card Number		Card Contents
1	010	NAME01
2	020	NAME02
3	030	NAME03
4	040	NAME04
5	050	NAME05
6	060	NAME06
7	000	THIS CARD IS OUT OF SEQUENCE
8	070	NAME07
9	080	NAME08
10	090	NAME09
11	100	NAME10

The program, during creation, exhibits the contents of NOMINAL KEY after the execution of each WRITE statement. After creation, the relative file is closed, reopened as an input file, and written out on the printer. The following discussion highlights some basic features. Circled numbers in the program example refer to corresponding numbers in the text.

1. The nominal keys, ① , that have been exhibited contain the relative record numbers of real records on the file. Relative records 10, 20, 30, 40, 50, 60, 61, 70, 80, 90, and 100 are real; all others are dummy records formatted by a COBOL subroutine. Note the nominal key N-KEY = 61. The initial value taken from C-FILE, card 7, was 000. This value, however, was not in logical sequence since relative records 000 through 060 had already been written. Therefore, a COBOL subroutine ignored the value 000 and adjusted it to the next appropriate relative record number (i.e., 61).

2. The contents of N-KEY for the first WRITE, ② , was 10. This means that a COBOL subroutine formatted relative records 0 through 9, placing the constand HIGH-VALUE in the first position of each record.

   Note: The constant HIGH-VALUE is exhibited as a blank since FF is not a printable character.

3. The contents of N-KEY for the second WRITE, ③ , was 20. Therefore, the COBOL subroutine formatted relative records 11 through 19.

4. The contents of N-KEY for the seventh WRITE, ④ , was initially 000. As explained in step 1 , N-KEY was adjusted to 61 and the record was written in the next available position.

5. Since this file was created on a 2314 mass-storage device, the track capacity for R-FILE is 39 records per track. Relative record 100 is, therefore, on the third track. Since the file is closed after writing relative record 100, the COBOL subroutine formats the rest of the third track. In this case, it means the addition of 17 dummy records, ⑤,

```
C0001   00101  IDENTIFICATION DIVISION.
CC002   00102  PROGRAM-ID. CREATER.
C0003   00103  REMARKS. ILLUSTRATE CREATION OF A RELATIVE FILE.
CC004   00104  ENVIRONMENT DIVISION.
00005   00105  CONFIGURATION SECTION.
CC006   00106  SOURCE-COMPUTER. IBM-370.
CC007   00107  OBJECT-COMPUTER. IBM-370.
00008   00108  INPUT-OUTPUT SECTION.
C0009   00109  FILE-CONTROL.
C0010   00110      SELECT R-FILE ASSIGN DA-2314-R-MASTER
C0011   00111          ACCESS IS SEQUENTIAL
C0012   00112          NOMINAL KEY IS N-KEY.
C0013   001125     SELECT C-FILE ASSIGN UR-S-CARDS.
C0014   001126     SELECT R-FILE2 ASSIGN DA-2314-R-MASTER.
C0015   001127     SELECT PRTFILE ASSIGN UR-S-PRTOUT.
C0016   00113  DATA DIVISION.
C0017   00114  FILE SECTION.
00018   00115  FD  R-FILE
C0019   00116      LABEL RECORDS ARE STANDARD
C0020   00117      RECORDING MODE IS F
C0021   00118      DATA RECORD IS DISK.
C0022   001184 01  DISK PIC X(80).
C0023   001185 FD  R-FILE2 LABEL RECORDS ARE STANDARD.
C0024   001186 01  DISK2 PIC X(80).
C0025   00201  FD  C-FILE
C0026   00202      LABEL RECORDS ARE OMITTED
C0027   00203      DATA RECORD IS CARD.
C0028   00204  01  CARD.
C0029   002041     02  C-KEY PIC 9(3).
C0030   002042     02  FILLER PIC X(77).
CC031   002043 FD  PRTFILE  LABEL RECORDS ARE OMITTED.
C0032   002044 01  PRT.
C0033   002045     02  FILLER PIC X.
C0034   002046     02  FIELD1  PIC X(132).
C0035   00205  WORKING-STORAGE SECTION.
C0036   00206  77  N-KEY PIC S9(8) COMP SYNC.
CC037   00207  PROCEDURE DIVISION.
C0038   00208      OPEN INPUT C-FILE
C0039   00209           OUTPUT R-FILE.
C0040   00210  R1. READ C-FILE AT END GO TO EOJ1.
C0041   00211      MOVE C-KEY TO N-KEY.
C0042   00212      WRITE DISK FROM CARD.
C0043   00213      EXHIBIT NAMED N-KEY.  GO TO R1.
C0044   00214  EOJ1.
C0045   00215      CLOSE C-FILE R-FILE.
00046   00216      OPEN INPUT R-FILE2  OUTPUT PRTFILE.
C0047   00217  R2. READ R-FILE2 AT END GO TO EOJ2.
00048   00218      MOVE DISK2 TO FIELD1.
C0049   00219      WRITE PRT AFTER 1 LINES  GO TO R2.
C0050   00220  EOJ2.
C0051   00230      CLOSE R-FILE2 PRTFILE.  STOP RUN.
```

Figure 35. Sample Program for Relative File Processing (Part 1 of 4)

```
F64-LEVEL LINKAGE EDITOR OPTIONS SPECIFIED LIST,XREF,LET
        DEFAULT OPTION(S) USED -  SIZE=(196608,65536)


                                CROSS REFERENCE TABLE

 CONTROL SECTION

   NAME     ORIGIN   LENGTH
CREATER       00       B8A
ILBOCOM0*    BC0      169
ILBODSP *    D30      9F0
ILBOEXT *   1720       50
ILBOQIO *   1770      56E
ILBOSAM *   1CE0      4C8
ILBOSPA *   21A8      670
ILBOSRV *   2818      48E
ILBOBEG *   2CA8      128
ILBOCMM *   2DD0      38B
ILBOCVB *   3160      412
ILBOMSG *   3578       F2

 ENTRY

   NAME     LOCATION     NAME     LOCATION     NAME     LOCATION     NAME     LOCATION
ILBOCOM       BC0
ILBODSP0      D32     ILBODSS0      D32
ILBOEXT0     1722     ILBOEXT1     1726
ILBOQIO0     1772
ILBOSAM0     1CE2
ILBOSPA0     21AA     ILBOSPA1     21AE     ILBOSPA2     21B2
ILBOSRV0     2822     ILBOSR5      2822     ILBOSR3      2822     ILBOSR       2822
ILBOSRV1     2826     ILBOSTP1     2826     ILBOST       282A     ILBOSTP0     282A
ILBOBEG0     2CAA
ILBOCMM0     2DD2     ILBOCMM1     2DD6
ILBOCVB0     3162     ILBOCVB1     3166
ILBOMSG0     357A

 LOCATION  REFERS TO SYMBOL  IN CONTROL SECTION      LOCATION  REFERS TO SYMBOL  IN CONTROL SECTION
     788       ILBOSRV0        ILBOSRV                 78C       ILBOSR5         ILBOSRV
     7C0       ILBOEXT0        ILBOEXT                 7C4       ILBOQIO0        ILBOQIO
     7C8       ILBOSAM0        ILBOSAM                 7CC       ILBODSP0        ILBODSP
     7D0       ILBOSPA0        ILBOSPA                 7D4       ILBOSRV1        ILBOSRV
     720       ILBOCOM0        ILBOCOM0               27FC       ILBOCVB0        ILBOCVB
    2868       ILBOCOM         ILBOCOM0               286C       ILBOCMM0        ILBOCMM
    2870       ILBOBEG0        ILBOBEG                2874       ILBOMSG0        ILBOMSG
    2878       ILBOSND2       $UNRESOLVED(W)
ENTRY ADDRESS      00

 TOTAL LENGTH      3670
****GO        DOES NOT EXIST BUT HAS BEEN ADDED TO DATA SET
AUTHORIZATION CODE IS         0.


N-KEY = 00000010
N-KEY = 00000020
N-KEY = 00000030
N-KEY = 00000040
N-KEY = 00000050
N-KEY = 00000060   (1)
N-KEY = 00000061
N-KEY = 00000070
N-KEY = 00000080
N-KEY = 00000090
N-KEY = 00000100
```

Figure 35. Sample Program for Relative File Processing (Part 2 of 4)

```
 10  NAME01
 10  NAME01
 10  NAME01
 10  NAME01
 10  NAME01
 10  NAME01
 10  NAME01
 10  NAME01
 10  NAME01
 10  NAME01
010  NAME01
 20   NAME02
 20   NAME02
 20   NAME02
 20   NAME02
 20   NAME02
 20   NAME02
 20   NAME02
 20   NAME02
 20   NAME02
020   NAME02
 30  NAME03
 30  NAME03
 30  NAME03
 30  NAME03
 30  NAME03
 30  NAME03
 30  NAME03
 30  NAME03
 30  NAME03
 30  NAME03
030  NAME03
 40   NAME04
 40   NAME04
 40   NAME04
 40   NAME04
 40   NAME04
 40   NAME04
 40   NAME04
 40   NAME04
 40   NAME04
 40   NAME04
040   NAME04
 50  NAME05
 50  NAME05
 50  NAME05
 50  NAME05
 50  NAME05
 50  NAME05
 50  NAME05
 50  NAME05
 50  NAME05
 50  NAME05
050  NAME05
 60   NAME06
 60   NAME06
 60   NAME06
 60   NAME06
 60   NAME06
 60   NAME06
 60   NAME06
 60   NAME06
 60   NAME06
 60   NAME06
060   NAME06
CC0  THIS  CARD  IS  OUT  OF  SEQUENCE
```

(2) (3) (4)

Figure 35. Sample Program for Relative File Processing (Part 3 of 4)

```
 70   NAME07
 70   NAME07
 70   NAME07
 70   NAME07
 70   NAME07
 70   NAME07
 70   NAME07
 70   NAME07
070   NAME07
 80  NAME08
 80  NAME08
 80  NAME08
 80  NAME08
 80  NAME08
 80  NAME08
 80  NAME08
 80  NAME08
 80  NAME08
080  NAME08
 90   NAME09
 90   NAME09
 90   NAME09
 90   NAME09
 90   NAME09
 90   NAME09
 90   NAME09
 90   NAME09
 90   NAME09
090   NAME09
 C0  NAME10
 00  NAME10
 C0  NAME10
 00  NAME10
 00  NAME10
 00  NAME10
 00  NAME10
 C0  NAME10
 00  NAME10
100  NAME10
 C0  NAME10
 C0  NAME10
 00  NAME10
 C0  NAME10
 00  NAME10
 00  NAME10
 00  NAME10
 00  NAME10
 00  NAME10
 00  NAME10
 00  NAME10
 00  NAME10
 C0  NAME10
 00  NAME10
 C0  NAME10
 00  NAME10
 00  NAME10
```

(5)

Figure 35. Sample Program for Relative File Processing (Part 4 of 4)

Data Management Techniques	Access Method	KEY Clauses	OPEN Statement	Access Verbs	CLOSE Statement
BSAM	SEQUENTIAL	[NOMINAL]	INPUT	READ[INTO] AT END	[UNIT] [WITH LOCK]
		NOMINAL	OUTPUT	WRITE[FROM] INVALID KEY	
BDAM	RANDOM	NOMINAL	INPUT	READ[INTO] INVALID KEY	[WITH LOCK]
			I - O	READ[INTO] INVALID KEY REWRITE[FROM] INVALID KEY	

Figure 36. Relative File Processing on Mass Storage Devices

INDEXED SEQUENTIAL FILE PROCESSING

The indexed sequential file processing technique arranges records on the tracks of a mass-storage device in a sequence determined by keys. The key is a control field that is a physical part of the record (defined in the FD) and is specified by the RECORD KEY clause in the Environment Division. The RECORD KEY clause identifies for the compiler the location and length of that item within the data record that will contain the key. It must always be specified.

Format
RECORD KEY IS data-name

Data-name may be any fixed-length item from 1 through 255 bytes in length.

When two or more record descriptions are associated with a file, a similar field must appear in each description, and must be in the same relative position from the beginning of the record, although the same data-name need not be used for both files.

Data-name must be defined to exclude the first byte of the record in the following cases:

1. Files with unblocked records.
2. Files from which records are to be deleted.
3. Files whose keys might start with a delete-code character (HIGH-VALUE).

For further information, see OS/VS Data Management Services Guide.

The position of each logical record in a file is determined by indexes created with the file and maintained by the system. The indexes are based on the RECORD KEYS and provide the following capabilities:

- Write and later read or update logical records in a sequential, ascending order (using QISAM) based on the collating sequence of the keys. This is done in a manner similar to that for sequential organization.
- Read or update individual logical records in a random manner (using BISAM). This method is somewhat slower per record than reading according to a collating sequence, since a search for pointers in indexes is required for the retrieval of each record.
- Insert new logical records at any point within the file (using BISAM). Using the indexes, the system locates the proper position for the new record and makes all necessary adjustments so that the sequence of the records, according to the keys, is maintained.

DD Statement Parameters Applicable to BSAM Input Files

DSNAME	Device	UNIT	VOLUME	LABEL	SPACE,SUBALLOC,SPLIT	DISP	DCB
as	Mass Storage required	not required if cataloged	[SL or SUL]	na		{OLD / SHR} (,PASS / ,KEEP / ,CATLG / ,DELETE / ,UNCATLG)	na

DD Statement Parameters Applicable to BSAM Output Files

DSNAME	Device	UNIT	VOLUME	LABEL	SPACE	SUBALLOC	SPLIT	DISP	DCB
as	Mass Storage required	as		[SL or SUL]	as RLSE	as	na	NEW (,KEEP / ,CATLG / ,PASS / ,DELETE) Note: MOD not meaningful	CPTCD={W,T} [DSORG=DA]

DD Statement PARAMETERS Applicable to BDAM Input and I-C Files

DSNAME	Device	UNIT	VOLUME	LABEL	SPACE,SUBALLOC,SPLIT	DISP	DCB
as	Mass Storage required	not required if cataloged	[SL or SUL]		na	{OLD / SHR} (,PASS / ,KEEP / ,CATLG / ,UNCATLG / ,DELETE)	as has been specified

as = Applicable subparameters
na = Not applicable

Figure 37. JCL Applicable to Relatively Organized Files

Indexes

There are two basic types of indexes: track indexes and cylinder indexes. There is one track index for each cylinder in the prime area (see "Indexed File Areas" for a description of prime area). The track index is written on the first track of the cylinder that it indexes. Each entry in the track index contains the identification of a specific track in the cylinder and the highest key on that track (Figure 38).

Figure 38 is the representation of a track index with the following areas:

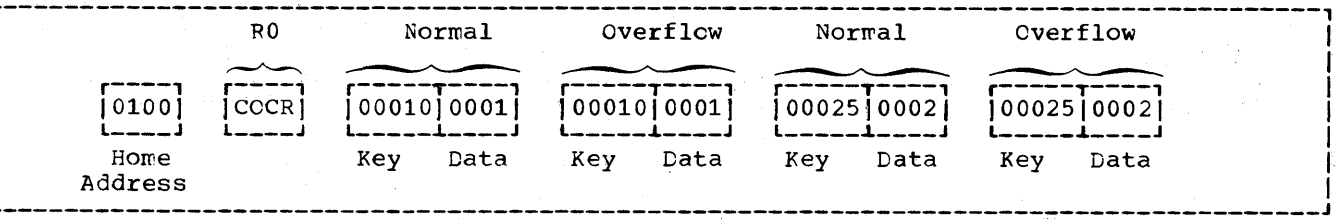


Figure 38. Track Index

Home Address -- This field defines the physical location of the track in which the index appears. It indicates the cylinder in which the track is located and the read/write head that services the track.

COCR (Cylinder Overflow Control Record) -- When a cylinder overflow area is specified (see "Indexed Sequential File Areas" for a description of overflow areas), R0 of each track index is used to keep track of overflow records and space available in the cylinder overflow area.

Normal Entry -- There is one normal and one overflow entry for each usable track in the cylinder. The Normal Entry contains two areas:

- Key -- the key of the highest record on the track specified in the Data area
- Data -- the home address of one of the prime tracks in the cylinder

Figure 38 shows that the highest key on track 1 is 10 and the highest key on track 2 is 25.

Overflow Entry -- The overflow entry is originally the same as the normal entry. It is changed when an attempt is made to add a record to a prime track on which space is no longer available. In this case, the overflow entry keeps track of the logical sequence of records although physically the record may be added to an overflow area.

There is one cylinder index for each file in which prime area data occupies more than one cylinder. The cylinder index contains one entry for each cylinder in the prime area; each entry pointing to the track index for a particular cylinder (Figure 39).

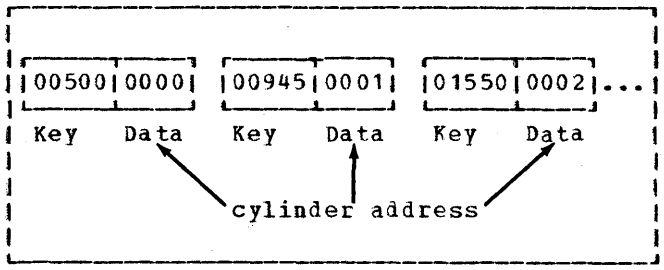


Figure 39. Cylinder Index

The cylinder index is formatted in the same fashion as the track index. Figure 39 shows that the highest key on cylinder 0 is 500, the highest key on cylinder 01 is 945, the highest key in cylinder 02 is 1550, etc.

Note: If an indexed sequential file is being read randomly, the system locates the given record by its key after a search of the cylinder index and the track index within the indicated cylinder. If the file is being read sequentially, starting, with the first record, no index search is performed.

Records, in indexed sequential files, may be either blocked or unblocked; but must be F-mode records. Figures 40 and 41 illustrate blocked and unblocked records as they appear on prime tracks of mass storage devices.

BLOCKED RECORDS

Count: contains control information

Key: contains the key of highest record in the block

Data(1, 2, ..., 6): each contains the information defined in the FD; including its own record key.

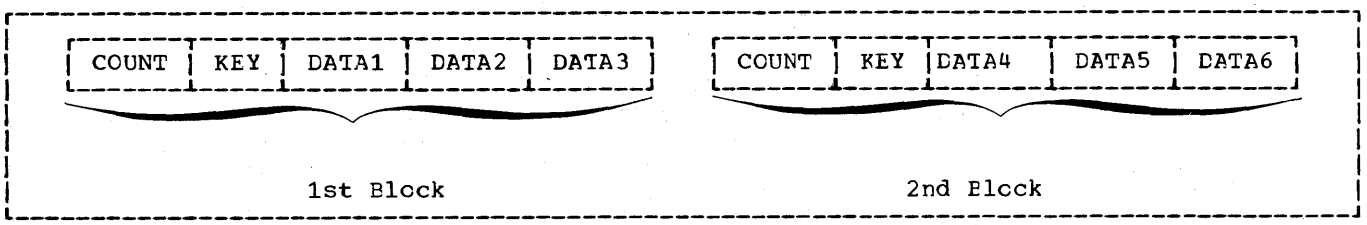


Figure 40. Blocked Records on an Indexed File

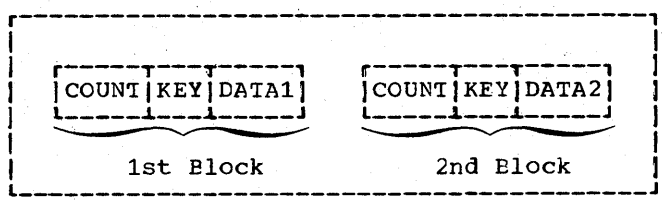


Figure 41. Unblocked Records on an Indexed File

UNBLOCKED RECORDS

Count: contains control information

Key: contains the key of the record that is in the block.

Data (1), (2), etc.: each contains the information defined in the FD; including its own record key.

Indexed Sequential File Areas

The programmer specifies the structure of an indexed sequential file and space to be allocated for it in the DD statement for the file when the file is created. In some instances, more than one DD statement is required. (These DD statements are described in "Using the DD Statements -- Single Volume Files.") The space being allocated must be divided into one, two, or three areas, depending on the needs of the programmer. These areas are: prime area, index area, and overflow area. The overflow area is optional.

Prime Area: The prime area is the area in which data records are written when the file is created or reorganized. These records are in a sequence determined by the record keys. The track indexes also use a portion of the reserved prime area. To reserve prime area space so that new logical records may be inserted without forcing records into an overflow area (described below), dummy records (records containing the figurative constant HIGH-VALUE in the first character position) may be written when the file is being created. The prime area may span multiple volumes and may consist of several noncontiguous areas.

Index Area: The index area contains the cylinder indexes and, if requested, master indexes (described later) for the file. This area exists for any file that has a prime area on more than one cylinder. Space for this area will be allocated separately from the prime area if specifically requested. The index area must be contained within one volume, but that volume need not be the same device type as the prime area volume. If not specifically requested, the index area will automatically be constructed in the independent overflow area, or, if there is no independent overflow area, it is constructed in the prime area.

Overflow Area: The overflow area is the area in which space is allocated for records forced from their original (prime) tracks by the insertion of new records. The fact that some records are stored in these areas, physically out of sequence, does not change the ability of QISAM to read the file in a logical sequence. An overflow area need not be specified if records are either not going to be added to the file, or sufficient space was originally reserved by writing dummy records in the prime area.

There are three ways in which space for an overflow area may be allocated:

1. Cylinder Overflow (Figure 42). Tracks on each cylinder can be reserved to hold the overflow of that cylinder (cylinder overflow option).

2. Independent Overflow (Figure 43). Space may be requested for an independent overflow area, using the dsname (OVFLOW) DD statement, either on the same volume or on a separate volume of the same device type as that of the prime area.

3. If the prime area is not filled when the file is created, the space remaining on the last cylinder on which data has been written will be designated as an independent overflow area (even though it is not requested directly). If a separate independent overflow area is requested, the remainder of the prime area is available for resuming a load operation.

Additional information about indexed file structure is contained in the publication OS/VS Data Management Services Guide.

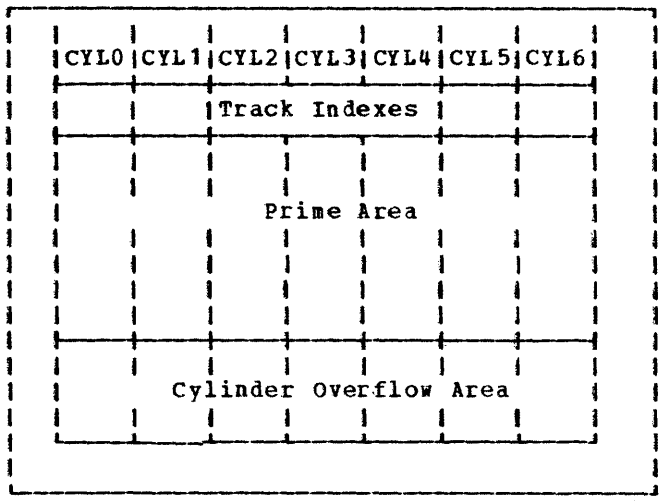


Figure 42. Cylinder Overflow Area

An advantage of having a cylinder overflow area is that additional seek operations are not required to locate overflow records. A disadvantage is that there will be unused space if additions are unevenly distributed throughout the file.

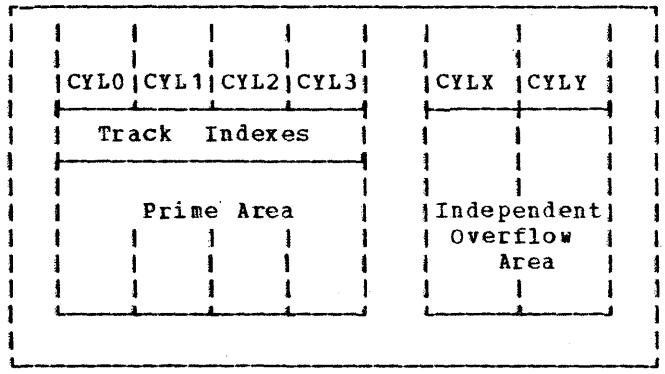


Figure 43. Independent Overflow Area

An advantage of having an independent overflow area is that less space need be reserved for overflows. A disadvantage is that accessing overflow records requires additional seek operations.

A suggested approach is to have cylinder overflow areas large enough to contain the average number of overflows caused by additions and an independent overflow area to be used as the cylinder overflow areas are filled.

Creating Indexed Sequential Files

Indexed files must be created sequentially using QISAM (Queued Indexed Sequential Access Method). Records must be arranged and written in ascending order according to the contents of RECORD KEY. If a WRITE statement is executed and the current contents of RECORD KEY is less than or equal to the previous contents of RECORD KEY, an INVALID KEY condition will result.

The structure of an indexed sequential file, and the space to be allocated to it, is specified in a DD statement(s). The space, which can be allocated in several different ways, must be sufficient for all areas of the file.

DD STATEMENT REQUIREMENTS FOR INDEXED FILES: The special parameter requirements for DD statements that define new indexed sequential files are discussed below. The discussion is oriented to indexed sequential files on one volume. Many of the parameters used for creating multivolume files are not discussed here. For more detailed information about parameters for both single-volume and multivolume files, see either of the publications OS/VS JCL Reference or OS/VS JCL Services.

ddname (name field)
The name field of the first or only DD statement defining the indexed sequential file can contain the symbolic identification ddname or procstep.ddname. Succeeding DD statements for the file must not be named.

DSNAME (DSN)
This parameter must be specified and is coded as follows:

$$\left\{\begin{matrix}\text{DSNAME}\\ \text{DSN}\end{matrix}\right\} = \left\{\begin{matrix}\text{dsname}\\ \text{\&\&name}\end{matrix}\right\} \quad [\,(\text{element})\,]$$

The first subparameter, dsname, or &&name must be the same in all the DD statements defining one data set. The element subparameter, INDEX, PRIME, or OVFLOW, indicates the type of area defined by the DD statement. If more than one DD statement is used to define a file, the order in which the statements should be placed in the input stream is as follows:

```
DD  DSNAME=dsname(INDEX)
DD  DSNAME=dsname(PRIME)
DD  DSNAME=dsname(OVFLOW)
```

Deviation from this sequence results in abnormal termination of the job. If the element subparameter is omitted PRIME is assumed. Note that an indexed sequential file cannot be specified by statements containing only index and overflow elements.

SPACE

This parameter specifies the space to be allocated for each of the separate areas on the device and must be included. Only cylinder (CYL) or absolute track (ABSTR) requests are permitted, and with ABSTR the designated tracks must encompass an integral number of cylinders. All the DD statements defining one indexed sequential file must specify the same subparameter, either CYL or ABSTR. When all the DD statements specify CYL, all must also specify or omit CONTIG, depending on whether the space allocated is to be contiguous or noncontiguous. The directory or index quantity subparameter of the SPACE parameter is used to request the number of cylinders to be allocated for an index area embedded within the prime area (see "Space Parameter" in "Job Control Procedures"). An embedded index resides in the middle of a track and saves searching time by first determining which half of the track contains the requested record.

SPLIT

This parameter should never be specified for an indexed sequential file, either for sharing a cylinder with indexed sequential files or for sharing it with an indexed sequential file and another type of file.

DISP

This parameter is written as it would be for any new file that cannot be cataloged. The CATLG subparameter must not be specified unless only one DD statement is used to allocate the file space (see "Cataloging Files" for additional information about cataloging indexed sequential files).

DCB

This parameter must be specified for each DD statement and is coded as follows:

```
DCB=(DSORG=IS
  [,BUFNO=integer]
  [,OPTCD={Y|I|R|W|L|M|U,NTM=integer}]
  [,BLKSIZE=integer])
```

The DSORG=IS subparameter is required and indicates that the organization of the file is sequential. The DCB subparameters of all the DD statements defining one file must not conflict. For example, if the OPTCD=Y subparameter appears in the first DD statement, the subsequent DD statements should also contain OPTCD=Y. To avoid any errors, code all the DCB subparameters on the first DD statement. Code DCB=*.ddname on the remaining statements; ddname is the name of the DD statement that contains the DCB subparameters. The subparameters are discussed below.

BUFNO=number of buffers

This subparameter is used to specify the number of buffers to be assigned to the file if no RESERVE or SAME AREA clause is specified for the file in the source program. The maximum number is 255; however, the maximum number allowed for an installation may differ and is established at system generation time.

OPTCD=options

This subparameter is used to tell the system that certain additional facilities are to be provided for this file. Any combination of the following options can be specified for the OPTCD subparameter. If more than one option is specified, the options are written as a character string (i.e., without intervening commas or blanks). Note that if certain of these options are used, an additional subparameter must also be specified as indicated. In addition to the information supplied, the COBOL compiler will supply OPTCD=L.

- OPTCD=L: This option requests that the control program delete marked records. Marked records will be deleted when space for new records is required.
- OPTCD=Y: This option requests that a cylinder overflow area be created. It specifies that a certain number of tracks on each cylinder are to be reserved to contain any overflow records from other tracks on that cylinder. Another DCB subparameter, CYLOFL=xx, must also be written. The xx specifies the number of tracks on the cylinder to be reserved for the overflow area. The maximum number is 99.
- OPTCD=I: This option requests that an independent overflow area be reserved. It is used in conjunction with DSNAME=dsname (OVFLOW) parameter in the DD

statement used to allocate the independent area.

- OPTCD=M: This option requests that a master index be created (see "Master Index" for a discussion of master indexes). Another DCB subparameter, NTM=xx, must also be written. It specifies the maximum number of tracks to be contained in the cylinder index before a higher level index is created. The maximum value that can be specified is 99.
- OPTCD=R: This option requests reorganization criteria feedback, as described in "Reorganizing Files."
- OPTCD=W: This option requests the system to perform a write-validity check.
- OPTCD=U: This option requests that track index entries be accumulated in main storage until there are enough entries to fill a track. When the track is full all the entries will be written out. If enough main storage cannot be obtained entries will be written two at a time.

The following is an example of how the OPTCD subparameter can be used:

```
DCB=(DSORG=IS,OPTCD=M,NTM=20)
```

The foregoing example requests that a master index be created when the cylinder index exceeds 20 tracks.

BLKSIZE=integer
specifies the blocksize. This clause is used only if BLOCK CONTAINS 0 RECORDS was specified at compile time.

Note: Figure 44 shows the parameters that may be used in a DD statement when processing indexed sequential files opened as output. Additional information about indexed sequential file structure is contained in the publication OS/VS Data Management Services Guide.

Using the DD Statements -- Single-Volume Files: The following examples refer to files that can be contained on one volume. Additional information about DD statements, including details on multivolume file allocation, can be found in the publication OS/VS JCL Reference.

All three areas for an indexed sequential file can be contained on a single volume if they are small enough. If such is the case and the programmer elects to allow the system to subdivide storage into the prime and index areas when the file is created, he need only code the following DD statement:

```
//ddname   DD  DSNAME=dsname(PRIME),          X
//              SPACE=(CYL,(no. of            X
//              cylinders)),UNIT=unit,        X
//              DCB=(DSORG=IS,...)
```

The DD statement given will produce a prime area with the index area occupying the last cylinder(s) of the space in the prime area. If any track(s) remain on the last cylinder after the index area, they are used as an independent overflow area; if no track(s) remain, an overflow area does not exist.

If the programmer definitely wants an independent overflow area, he must provide a second DD statement as follows:

```
//ddname   DD  DSNAME=dsname(PRIME),          X
//              SPACE=(CYL,(no. of            X
//              cylinders)),UNIT=unit,        X
//              VOLUME=SER=222222,            X
//              DCB=(DSORG=IS,OPTCD=I,...)
//           DD  DSNAME=dsname(OVFLOW),       X
//              SPACE=(CYL,(no. of            X
//              cylinders)),UNIT=unit,        X
//              VOLUME=SER=222222,            X
//              DCB=*.ddname
```

These DD statements will produce a prime area and a separate overflow area with the index area at the end of the overflow area. All three areas reside on the same volume.

Note: When more than on DD statement is used, only the first can be named. The others must not have a data definition name (ddname) but all must have the same data set name (dsname).

ddname	ddname used only for first DD statement of each file
DSNAME (DSN)	{dsname / &&name} (INDEX) (PRIME) (OVFLOW) Note: If more than one DD statement is used, elements must be in this order.
Device	Mass storage required
UNIT	DEFER not permitted
SEP, AFF	Restricted, see "Job Control Procedures"
VOLUME	Volume sequence number subparameter not applicable
LABEL	SL
SPACE	CYL ,... [,,CONTIG] ABSTR
SUBALLOC	Not applicable
SPLIT	Not applicable
DISP	NEW[1] [,KEEP / ,PASS / ,DELETE]
DCB[2]	Required: DSORG=IS Optional: BUFNO=xxx BLKSIZE=xxxx OPTCD={W|M|Y|I|R|L|U}
[1]MOD not meaningful. CATLG allowed only if all areas are allocated with a single DD statement	
[2]The DCB parameter should be the same for each DD statement	

Figure 44. DD Statement Parameters Applicable to Indexed Files Opened as Output

If the programmer desires more control in the placement of the index area, he can subdivide storage before the data set is created by providing another DD statement as follows:

```
//ddname  DD  DSNAME=dsname(INDEX),          X
//              SPACE=(CYL,(no. of           X
//              cylinders)),UNIT=unit,       X
//              VOLUME=SER=333333,           X
//              DCB=(DSORG=IS,...)
//          DD  DSNAME=dsname(PRIME),        X
//              SPACE=(CYL,(no. of           X
//              cylinders)).UNIT=unit,       X
//              VOLUME=SER=333333,           X
//              DISP=(disp),DCB=*.ddname
```

These DD statements will produce two separate areas: index and prime. Each area is on the same volume.

If, along with more control of his index, the programmer wishes an independent overflow area, a third DD statement (OVFLOW) can be specified, as detailed earlier. The sequence will be:

```
//ddname  DD  DSNAME=dsname(INDEX),...
//          DD  DSNAME=dsname(PRIME),...
//          DD  DSNAME=dsname(OVFLOW),...
```

These DD statements will produce three separate areas: index, prime, and overflow.

Note that the OPTCD subparameter of the DCB parameter in each of the DD statements must specify an independent overflow area (OPTCD=I). All three areas reside on the same volume if so specified in the VOLUME parameter.

Note: The sequence of the DSNAME parameter elements in all of the foregoing examples must be followed when placing the DD statements into the input stream, or an abnormal termination of the job will result.

The example in Figure 45 defines a new indexed sequential file that consists of three separate areas. All three areas reside on the same volume. The volume is on an IBM 2314 Disk Storage Drive.

Cataloging Files: An indexed file can be cataloged if:

- All the areas of the file are allocated with a single DD statement. Such a file is cataloged in the usual manner by specifying the DISP parameter in the DD statement:

  DISP=(NEW,CATLG)

- The areas are allocated with more than one DD statement, but all volumes are on the same type of device. Such a file is cataloged using the IEHPROGM utility program (see the publication OS/VS Utilities).

An indexed sequential file that is being created cannot be cataloged if its areas are on different device types. An existing indexed sequential file cannot be cataloged through the specification of the CATLG subparameter of the DISP parameter in the DD statement.

```
//FILE    DD  DSNAME=ISM(INDEX),UNIT=2314,SPACE=(CYL,(1)),  X
//            VOLUME=SER=111111,DCB=(DSORG=IS,OPTCD=I,...)
//        DD  DSNAME=ISM(PRIME),UNIT=2314,SPACE=(CYL,(5)),  X
//            VOLUME=SER=111111,DISP=(,KEEP),DCB=*.FILE
//        DD  DSNAME=ISM(OVFLOW),UNIT=2314,SPACE=(CYL,(1)), X
//            VOLUME=SER=111111,DISP=(,KEEP),DCB=*.FILE
```

Figure 45. Example of DD Statements for New Indexed Files

Note: The DD statement(s) defining a new or existing indexed sequential file can appear in cataloged procedures.

Calculating Space Requirements: To determine the number of cylinders required for an indexed sequential file, the programmer must consider the number of records that will fit on a cylinder, the number of records that will be processed, and the amount of space required for indexes and overflow areas. In making the computations, additional space is also required for device overhead.

Note: The allocation of space to the different areas of an indexed sequential file is permanent. New allocations can be achieved only by recreating the file. It is, therefore, important to remember:

- Unused space on the last cylinder on which data was written, in the prime area, is converted to an independent overflow area. Space allocated in excess of this cannot be released and will be wasted.
- Excess space allocated to overflow or index areas cannot be released.

Detailed information on space allocation can be found in the publication OS/VS Data Management Services Guide.

Master Index: QISAM provides a master index facility to avoid inefficient serial searches of large cylinder indexes. The master index provides an index to the cylinder index. The programmer can specify with the DCB parameter in his DD statement(s) (see "DD Statement Requirements for Indexed Sequential Files" in "Creating Indexed Sequential Files") that a master index be built if the size of a cylinder index exceeds a certain number of tracks. Each entry in the master index points to a track of the cylinder index. If the size of the master index exceeds the number of tracks specified in the NTM parameter of the DD statement, the master index is automatically indexed by a higher level master index. Three such higher level master indexes can be constructed.

COBOL Considerations: When creating indexed sequential files, the QISAM file processing technique is used. The following COBOL programming considerations should be noted:

- RECORD KEY Clause. The RECORD KEY clause in the SELECT sentence of the Environment Division is required. It is used to specify the location of the key within the record itself. If the RECORD KEY clause has a PICTURE clause that specifies that the item is binary (COMPUTATIONAL), zero is the lowest number acceptable as the first record. A negative key is considered to be larger than a positive key; therefore, if a record is inserted into the file, a negative key would place the record after those records with positive keys.
- Dummy Records. To reserve space for records to be added at a later time, when creating indexed sequential files, dummy records can be written with the delete code (the figurative constant HIGH-VALUE) in the first byte. Dummy records and their deletion are described in "Using the WRITE Statement."
- Required and optional COBOL statements are summarized in Figure 47.

Reading or Updating Indexed Sequential Files Sequentially

QISAM can be used to read or update an existing indexed sequential file. Adding a record to an already existing file, however, can be done only with BISAM (see "Accessing an Indexed File Randomly").

When QISAM is used to read an input file, the READ statement makes available one logical record at a time in an ascending sequence determined by the record keys. Dummy records are not made available. If there are records in the overflow area, this sequence will not correspond exactly to the physical sequence of the records in the file. The file must have been created using QISAM.

When QISAM is used to update an I-O file, the READ and REWRITE statements permit updating-in-place or deletion of a logical record. Logical records are read sequentially and may be either updated and rewritten, or rewritten unaltered, from the same area. Alteration of record length or insertion of new records is not permitted. A logical record is marked for deletion by moving the figurative constant HIGH-VALUE into the first character position of the record and then using the REWRITE statement. Records in the file that contain this deletion code are not made available on input.

The discussion that follows is primarily concerned with indexed sequential files that can be contained on a single volume. Additional information about processing existing indexed sequential files accessed sequentially, including multivolume files, can be found in the publication OS/VS JCL Reference.

Parameter Requirements: In the DD statement(s) indicating an existing indexed file, the following differences and requirements should be noted:

DCB

The DSORG=IS subparameter must be specified, whereas the BUFNO subparameter is optional. The OPTCD field must not be specified again. Any OPTCD subparameter facilities that were specified when the file was created are in effect as long as the data set exists. For example, if the programmer specified the write-validity check option (OPTCD=W) when he created the file, the option is still in effect at the time of any subsequent WRITE statement. The BLKSIZE and LRECL subparameters must not be specified.

DSNAME (DSN)

This parameter is written DSNAME=dsname. The element subparameters (INDEX, PRIME, OVFLOW), must not be written.

DISP

The first subparameter must be OLD. The second subparameter cannot be CATLG or UNCATLG (see "Cataloging Files" above for more information on cataloging indexed sequential files).

Note: For further information about Indexed Sequential parameters, see "DD Statement Requirements for Indexed Sequential Files" in "Creating Indexed Sequential Files."

Only one DD statement is needed to specify an existing file if all of the areas are on one volume. The following is an example of a DD statement that can be used when processing a single-volume QISAM file.

```
//ddname  DD  DSNAME=dsname,                X
//              DCB=(DSORG=IS,...),          X
//              UNIT=unit,DISP=OLD
```

Further details about DD statements for existing single-volume and multivolume indexed sequential files can be found in the publication OS/VS JCL Reference. Reference.

Note: Figure 46 shows the parameters that may be used in a DD statement when processing indexed sequential files opened as INPUT or I-O. Additional information about indexed file structure is contained in the publication OS/VS Data Management Services Guide.

Reorganizing Files: As new records are added to an indexed sequential file, chains of records may be created in the overflow area if one exists. The access time for retrieving records in an overflow area is greater than that required for retrieving records in the prime area. Input/output performance is, therefore, sharply reduced when many overflow records develop. For this reason, an indexed sequential file can be reorganized as soon as the need becomes evident. The system maintains a set of statistics to assist the programmer when reorganization is desired. These statistics are maintained as fields of the file's data control block. They are made available when APPLY REORG-CRITERIA is specified. If these statistics are desired, the OPTCD subparameter of the DCB parameter must have included the OPTCD=R parameter in each of the DD statements when the file was created. Additional information about reorganizing files is contained in the publication OS/VS Data Management Services Guide.

Sequential Retrieval Using the START Statement: For indexed sequential INPUT and I-O files, retrieval starts with the first nondummy record in the file. If the programmer wishes to begin processing at a point other than the beginning of the file, he can do so through the use of the START verb. When the START statement is used, the retrieval starts sequentially from the record specified in the NOMINAL KEY.

ddname	ddname used only for first DD statement of each file
DSNAME	dsname Note: Element subparameter must not be used.
Device	Mass storage required
UNIT	Applicable subparameter Note: Not needed if file is cataloged.
SEP, AFF	Restricted; see "Job Control Procedures"
VOLUME	Applicable subparameters
LABEL	SL
SPACE	Not applicable
SUBALLOC	Not applicable
SPLIT	Not applicable
DISP	OLD[1] [,KEEP / ,PASS / ,DELETE]
DCB	Required: DSORG=IS Optional: BUFNO=xxx (not allowed for BISAM) LRECL=xxx
[1]CATLG UNCATLG not permitted.	

Figure 46. DD Statement Parameters Applicable Indexed Sequential Files Opened as INPUT or I-O

COBOL Considerations: When processing an already existing file with QISAM, the following COBOL programming considerations should be noted:

- RECORD KEY Clause. The RECORD KEY always in the SELECT sentence of the Environment Division is required, just as it is when creating the file. Note other record key considerations under "Accessing an Indexed Sequential File Randomly."
- Delete Option. In order to keep the number of records in the overflow area to a minimum, and to eliminate unnecessary records, an existing record may be marked for deletion. This is done by moving the figurative constant HIGH-VALUE into the first character position of the record (unless it has been changed by the program collating sequence--in which case a X'FF' must be moved). The record is not physically deleted unless it is forced off its prime track by the insertion of a new record (see "Using the WRITE Statement" in "Accessing an Indexed File Randomly"), or if the file is reorganized. Records marked for deletion may be replaced (using BISAM) by new records containing equivalent keys. Execution of the READ statement in QISAM does not make available a record marked for deletion, whether the record has been physically deleted or not. Dummy records and deletion are discussed further in "Accessing an Indexed Sequential File Randomly."

Accessing an Indexed Sequential File Randomly

The file processing technique used for random retrieval of a logical record, the random updating of a logical record, and/or the random insertion of a record is BISAM (Basic Indexed Sequential Access Method). When accessing an indexed sequential file randomly, both NOMINAL KEY and RECORD KEY must be specified. The format of the NOMINAL KEY is described briefly below:

Format
NOMINAL KEY IS data-name

Data-name may be any fixed-length Working Storage item from 1 through 255 bytes in length. If it is part of a logical record, it must be at a fixed displacement from the beginning of that record description (see the publication IBM VS COBOL for OS/VS for additional information).

Since a RECORD KEY is used to identify a record to the system, the record keys associated with the logical records of the file may be thought of as a table of arguments. When a record is read or written, the contents of NOMINAL KEY is used as a search argument that is compared to the record keys of the file.

The following example illustrates the use of the NOMINAL KEY clause.

```
ENVIRONMENT DIVISION.
        .
        .
        .
    NOMINAL KEY IS NOM-KEY
    RECORD KEY IS REC-KEY.
        .
        .
        .
DATA DIVISION.
FILE SECTION.
FD  INDEXED-FILE
    LABEL RECORDS ARE STANDARD.
01  REC-1.
    02  DELETE-CODE  PIC X.
    02  REC-KEY      PIC 9(5).
        .
        .
        .
WORKING-STORAGE SECTION.
77  NOM-KEY          PIC 9(5).
```

Because of their complementary use of the indexed file organization, much of the information discussed above for QISAM also applies to BISAM. Differences are noted below.

Using the WRITE Statement: The programmer can use the WRITE statement to add a new record into an indexed file. The record is added on the basis of the value specified in the NOMINAL KEY. The contents of the NOMINAL KEY are used to locate the two records in the file between which the new record is to be inserted. The records sought are those that have values less than and greater than the values in the nominal key field. Two methods can be used to add records.

In the first method, the key to be added is a new key value. The record is inserted in place so that the sequence of the keys is maintained. If an overflow area exists, the insertion may cause records to be forced off the prime track into the overflow area. Dummy records forced off the track in this way are physically deleted and are not written in the overflow area.

In the second method, the key of the record to be added has the same value as that of a known dummy record. If the dummy record has not been physically deleted, it is replaced by the new record. If it has been physically deleted, the record is inserted as though it had a new key value. If the key of the record to be added has the same value as a record other then a dummy record, an INVALID KEY condition will result.

Notes:

- Records with a key higher (or lower) than the current highest (or lowest) key of the file may be added.
- Whenever a WRITE statement is executed the contents of RECORD KEY and NOMINAL KEY must be identical. Except in the case of dummy records, this value must be unique in the file.

Using the REWRITE Statement: If a record is to be updated, the indexed file should be opened as I-O and the REWRITE statement should be used. All REWRITE statements must be preceded by a READ statement. However, a READ statement can be followed by either a WRITE, REWRITE, or another READ.

Note: Whenever a REWRITE statement is executed the value contained in NOMINAL KEY and RECORD KEY must be identical.

Using the READ Statement: Records are retrieved on the basis of the value specified in the NOMINAL KEY. If the key of a record marked for deletion is specified and the record has not been physically deleted, it will be produced. If the record has been physically deleted, the READ statement will cause an INVALID KEY condition and control will go to the INVALID KEY routine if specified.

Note: Although the RECORD KEY clause must be specified, no value need be moved to the record key field before the execution of the READ statement. The search for the desired record is based on the contents of NOMINAL KEY.

COBOL Considerations: When processing an indexed file randomly, the following COBOL programming considerations should be noted:

- RECORD KEY Clause and NOMINAL KEY Clause. The RECORD KEY and NOMINAL KEY clauses in the SELECT sentence of the Environment Division are required. The RECORD KEY clause is used to specify the location of the key within the record itself. The NOMINAL KEY is used as a search argument to locate the proper record, and must not be defined within the file being processed. Note that since a RECORD KEY is defined within a record, the contents of RECORD KEY are not available after a WRITE statement has been executed for that record.

Data Management Techniques	Access Method	KEY Clauses	OPEN Statement	Access Verbs	CLOSE Statement
QISAM	SEQUENTIAL	RECORD NOMINAL	INPUT	READ [INTO] AT END START INVALID KEY	[WITH LOCK]
			OUTPUT	WRITE [FROM] INVALID KEY	
			I-O	READ [INTO] AT END START INVALID KEY REWRITE [FROM]	
BISAM	RANDOM	RECORD NOMINAL	INPUT	READ [INTO] INVALID KEY	[WITH LOCK]
			I-O	READ [INTO] INVALID KEY WRITE [FROM] INVALID KEY REWRITE [FROM] INVALID KEY	

Figure 47. Indexed Sequential File Processing on Mass Storage Devices

- TRACK-AREA Clause. Specifying the clause results in a considerable improvement in efficiency when a record is added to the file. If a record is added and the TRACK-AREA clause was not specified for the file, the contents of the NOMINAL KEY field are unpredictable after the WRITE statement is executed. In this case, the key must be reinitialized before the next WRITE statement is executed.

  Even if TRACK-AREA is specified, if the addition of a record causes another record to be bumped off the track and into the overflow area, the contents of the NOMINAL KEY are unpredictable after a WRITE.

- APPLY REORG-CRITERIA Clause. If the OPTCD=R parameter was specified on the DD card for an indexed sequential file when it was created, the APPLY REORG-CRITERIA clause can be used to obtain the reorganization statistics when the file is closed. These statistics are moved from the data control block to the identifier specified in the clause when a CLOSE statement is executed for the file.

- APPLY CORE-INDEX Clause. This clause specifies that the highest level index will reside in core storage during input/output operations. Otherwise, the index will be searched on the volume, and processing time will be longer.

- Required and optional COBOL statements are summarized in Figure 47.

USING THE DD STATEMENT

Each data set that is defined by a DD statement is either to be created, or has been previously created and is to be retrieved. In either case, the data set must have a disposition; for example, if the data set is being created, the disposition must indicate whether the data set is to be cataloged, kept, or deleted. Other DD parameters may simply indicate that the data set is in the input stream or that ultimately the data set is to be printed or punched.

The following sections summarize the DD statement parameters and show examples for various uses of the DD statement. These sections include information about cataloging data sets and creating or referring to generation data groups; examples of cataloged data sets and partitioned data sets are included. For additional information about partitioned data sets see "Libraries." Also see "Appendix I: Checklist for Job Control Procedures" for additional examples of the DD statement used in job control procedures.

CREATING A NON-VSAM DATA SET

When creating a non-VSAM data set, the programmer ordinarily will be concerned with the following parameters:

1. The data set name (DSNAME) parameter, which assigns a name to the data set being created.

2. The unit (UNIT) parameter, which allows the programmer to state the type and quantity of input/output devices to be allocated for the data set.

3. The volume (VOLUME) parameter, which allows specification of the volume in which the data set is to reside. This parameter also gives instructions to the system about volume mounting.

4. The space (SPACE), split cylinder (SPLIT), and suballocation (SUBALLOC) parameters, for mass storage devices only, which permit the specification of the type and amount of space required to accommodate the data set.

5. The label (LABEL) parameter, which specifies the type and some of the contents of the label associated with the data set.

6. The disposition (DISP) parameter, which indicates what is to be done with the data set by the system when the job step is completed.

7. The DCB parameter, which allows the programmer to specify additional information to complete the DCB associated with the data set (see "User-Defined Files"). This allows additional information to be specified at execution time to complete the DCB constructed by the compiler for a data set defined in the source program.

Figure 48 shows the subparameters that are frequently used in creating data sets. Additional subparameters are discussed in "Job Control Procedures."

```
{DSNAME}   {dsname                }
{DSN   } = {dsname(element)       }
           {&&name                }
           {&&name(element)       }

UNIT=(name[,unit count])

{VOLUME}
{VOL   } =([PRIVATE][,RETAIN][,volume-sequence-number][,volume-count]

          [,SER=(volume-serial-number[,volume-serial-number]...)]
          [                                                     ]
          [       dsname                                        ]
          [       *.ddname                                      ]  )
          [,REF=  *.stepname.ddname                             ]
          [       *.stepname.procstep.ddname                    ]

SPACE=(  {TRK                    }  ,(primary-quantity[,secondary-quantity]
         {CYL                    }
         {average-record-length  }

       [,directory-quantity]))

SPLIT=(n,[CYL                   ] [,(primary-quantity,[secondary-quantity])]
         [average-record-length ]

LABEL=([data-set sequence-number], {NL } [,EXPDT=yyddd])
                                   {SL }
                                   {NSL} [,RETPD=xxxx  ]
                                   {SUL}

DISP=(  [NEW]   [,DELETE]   [,DELETE]  )
        [MOD]   [,KEEP  ]   [,KEEP  ]
                [,PASS  ]   [,CATLG  ]
                [,CATLG ]

DCB=(subparameter-list)
```

Figure 48. DD Statement Parameters Frequently Used in Creating Data Sets

Creating Unit Record Data Sets

Data sets whose destination is a printer or card punch are created with the DD statement parameters UNIT and DCB.

UNIT: Required. Code unit information using the 3-digit address (e.g., UNIT=00E), the type (e.g., UNIT=1403), or the system-generated group name (e.g., UNIT=PRINTER).

DCB: Required only if the data control block is not completed in the processing program. Valid DCB subparameters are listed in "Appendix C: Fields of the Data Control Block."

Creating Data Sets on Magnetic Tape

Tape data sets are created using combinations of the DD statement parameters UNIT, LABEL, DSNAME, DCB, VOLUME, and DISP.

UNIT: Required, except when volumes are requested using VOLUME=REF. A unit can be assigned by specifying its address, type, or group name, or by requesting unit affinity with an earlier data set. Multiple output units and defer volume mounting can also be requested with this parameter.

LABEL: Required when the tape has user labels or does not have standard labels, and when the data set does not reside first on the reel. It is also used to assign a retention period and password protection.

DSNAME: Required for data sets that are to be cataloged or used by a later job.

DCB: Required only when data control block information cannot be specified in COBOL. Usually, such attributes as the logical record length (LRECL) and buffering technique (BFTEK) will have been specified in the processing program. Other attributes, such as the OPTCD field and the tape recording technique (TRTCH), are more appropriately specified in the DD statement. Valid DCB subparameters are listed in "Appendix C: Fields of the Data Control Block."

VOLUME: Optional, this parameter is used to request specific volumes. If VOLUME=REF is specified, and the existing data sets on the specified volume(s) are to be saved, indicate the data set sequence number in the LABEL parameter.

DISP: Required for data sets that are to be cataloged, passed, or kept. The programmer can specify conditional disposition as the third term in the DISP parameter to indicate how the data set is to be treated if the job step abnormally terminates.

Creating Sequential (BSAM or QSAM) Data Sets on Mass Storage Devices

Sequential data sets are created using combinations of the DD statements parameters UNIT, DSNAME, VOLUME, LABEL, DISP, DCB, and one of the space allocation parameters SPACE, SPLIT, or SUBALLOC.

UNIT: Required, except when volumes are requested using VOLUME=REF or space is allocated using SPLIT or SUBALLOC. Assign a unit by specifying its address, type, or group name, or by requesting unit affinity.

DSNAME: Required for all but temporary data sets.

Label: Required to specify label type and to assign a retention period or password protection.

DCB: Required only when data control block information is not completely specified in the processing program. Usually, such attributes as the logical record length (LRECL) and buffering technique (BFTEK) will have been specified in the processing program. Other attributes, such as the OPTCD field are more appropriately specified in the DD statement. Valid DCB subparameters are listed in "Appendix C: Fields of the Data Control Block."

VOLUME: Optional. This parameter requests specific volumes (SER and REF), specific volumes when the data set resides on more than one volume (seq #), multiple nonspecific volumes (volcount), private volumes (PRIVATE), or private volumes that are to remain mounted until the end of the job (RETAIN).

DISP: Required for data sets that are to be cataloged, passed, or kept. The programmer can specify conditional disposition as the third term in the DISP parameter to indicate how the data set is to be treated if the job step abnormally terminates.

SPACE, SPLIT, SUBALLOC: One of these is required for all new mass storage data sets.

Creating Direct (BDAM) Data Sets

Direct (BDAM) data sets are created using the same subset of DD statement parameters as sequential data sets, with the exception of the SPLIT parameter. Valid DCB subparameters for BDAM data sets are listed in "Appendix C: Fields of the Data Control Block."

Creating Indexed (BISAM and QISAM) Data Sets

Indexed (ISAM) data sets are created using combinations of the DD statement parameters UNIT, DSNAME, VOLUME, LABEL, DISP, DCB, and SPACE. The ISAM data sets occupy three areas of storage: an index area that contains master and cylinder indexes, a prime area that contains the data records and track indexes, and an optional overflow area to hold additional records when the prime area is exhausted. Detailed information on creating and retrieving indexed sequential data sets is presented in "Appendix H: Creating and Retrieving Indexed Sequential Data Sets."

Creating Data Sets in the Output Stream

New data sets can be written on a system output device in much the same way as messages. A data set is directed to the output stream with the SYSOUT and DCB parameters.

SYSOUT: Required. The output class through which the data set is routed must be specified. Output classes are identified by a single alphanumeric character.

DCB: Required only if complete data control block information has not been specified in the processing program.

When using a priority scheduler, data sets are not routed directly to a system output device. They are stored by the processing program on an intermediate mass storage device and later written on a system output device. In addition to the SYSOUT and DCB parameters, DD statements defining a data set of this type can also contain UNIT and SPACE parameters. All other parameters must be absent.

SYSOUT: Required. The output class through which the data set is routed must be specified. Output classes are identified by a single alphanumeric character.

DCB: Required only if complete data control block information has not been specified in the processing program. Data control block information is used when the data set is written on an intermediate mass storage volume and read by the output writer. However, the output writer's own DCB attributes are used when the data set is written on the system output device. Valid DCB parameters are listed in "Appendix C: Fields of the Data Control Block."

UNIT: Optional. An intermediate mass storage device is assigned if UNIT is specified. A default device is assigned if this parameter is omitted.

SPACE: Optional. Estimate the amount of mass storage space required. A default estimate is assumed if this parameter is omitted.

Note: When a Direct SYSOUT Writer is used (OS/VS1 only), the scheduler functions as a sequential scheduler. The SYSOUT data sets of the particular output class from any of the elegible job classes are not stored on an intermediate storage device, but are written directly to the system output device. When Direct SYSOUT Writer is used, all the parameters on the DD card are ignored. For detailed information on Direct SYSOUT Writer, see the publication OS/VS1 Planning and Use Guide.

Examples of DD Statements Used To Create Data Sets

The following examples show various ways of specifying DD statements for data sets that are to be created. In general, the number of parameters and subparameters that are specified depend on the disposition of the data set at the end of the job step. If a data set is used only in the job step in which it is created and is deleted at the end of the job step, a minimum number of parameters are required. However, if the data set is to be cataloged, more parameters should be specified.

Example 1: Creating a data set for the current job step only.

```
//SYSUT1    DD UNIT=SYSDA,SPACE=(TRK,(50,10))
```

This example shows the basic required DD statement for creating and storing a data set on a mass storage device. The UNIT parameter is required unless the unit information is available from another source. If the data set were to be stored on a unit record or a tape device, the SPACE parameter would not be needed. The operating system assigns a temporary data set name and assumes a disposition of (NEW, DELETE).

Example 2: Creating a data set that is used only for the current job.

```
//SYSLIN    DD DSNAME=&&TEMP,DISP=(MOD,PASS),UNIT=SYSSQ,                X
//               SPACE=(TRK,(50))
```

This example shows a DD statement that creates a data set for use in more than one step of a job. The system assigns a unique symbol for the name, and this same symbol is substituted for each recurrence of the &&TEMP name within the job. The data set is allocated space on any available mass storage or tape device. If a tape device is selected, the SPACE parameter is ignored. The disposition specifies that the data set is either new or is to be added to (MOD), and is to be passed to the next job step (PASS). This DD statement can be used for specifying the data set that is created as output from the compiler and that is to be used as input to the linkage editor. By specifying MOD, separately compiled object modules can be placed in sequence in the same data set.

Note: If MOD is specified for a data set that does not already exist, the job may be abnormally terminated when a volume reference name, a volume serial number, or the disposition CATLG is specified or when the dsname is indicated by a backwards reference.

Example 3: Creating a data set that is to be kept but not cataloged.

```
//TEMPFILE  DD DSN=FILEA,DISP=(,KEEP),SPACE=(TRK,(30,10)),              X
//             UNIT=DIRECT,VOL=(,RETAIN,SER=AA70)
```

The example shows a DD statement that creates a data set that is kept but not cataloged. The data set name is FILEA. The disposition (,KEEP) specifies that the data set is being created in this job step and is to be kept. It is kept until a disposition of DELETE is specified on another DD statement. The KEEP parameter implies that the volume is to be treated as private. Private implies that the volume is unloaded at the end of the job step but because RETAIN is specified, the volume is to remain mounted until the end of the job unless another reference to it is encountered. The DIRECT parameter is a hypothetical device class, containing only mass storage devices. The volume with serial number AA70, mounted on a device in this class, is assigned to the data set. Space for the data set is allocated as specified in the SPACE parameter. The data set has standard labels since it is on a mass storage volume.

If the volume serial number were not specified in the foregoing example, the system would allocate space in an available nonprivate volume. Because KEEP is specified, the volume becomes private. (Another data set cannot be stored on a private volume unless its volume serial number is specified or affinity with a data set on the volume is stated.) The volume serial number of the volume assigned, if applicable, is included in the disposition message for the data set. Disposition messages are messages from the job scheduler, generated at the end of the job step.

Example 4: Creating a data set and cataloging it.

```
//DDNAMEA   DD DSNAME=INVENT.PARTS,DISP=(NEW,CATLG),                    X
//             LABEL=(,,EXPDT=77031),UNIT=DACLASS,                      X
//             VOLUME=(,REF=*.STEP1.DD1),                               X
//             SPACE=(CYL,(5,1),,CONTIG)
```

This example shows a DD statement that creates a data set named INVENT.PARTS and catalogs it in the previously created system catalog. The data set is to occupy the same volume as the data set referred to in the DD statement named DD1 occurring in the job step named STEP1. The UNIT parameter is ignored since REF is specified. Five cylinders are allocated to the data set, and if this space is exhausted, more space is allocated, one cylinder at a time. The five cylinders are to be contiguous. The disposition (CATLG), implies that the volume is to be private. The INVENT.PARTS is to have standard labels. The expiration date is the 31st day of 1977.

Example 5: Adding a member to a previously created library.

```
//SYSLMOD   DD DSNAME=SYS1.LINKLIB(INVENT),DISP=OLD
```

This DD statement adds a member named INVENT to the link library (SYS1.LINKLIB). When a member is added to a previously created data set, OLD should be specified.

Example 6: Creating a library and its first member.

```
//SYSLMOD   DD DSNAME=USERLIB(MYPROG),DISP=(,CATLG),                    X
//             SPACE=(TRK,(50,30,3)),UNIT=3330,VOLUME=SER=111111
```

This DD statement creates a library, USERLIB, and places a member, MYPROG, in it. The disposition (,CATLG) indicates that the data set is being created in this job step (NEW is the default condition for the DISP parameter and is indicated by the comma) and is to be cataloged. The data set is to have standard labels. Space is allocated for the data set in a volume on a mass storage device that is an IBM 3330 unit. Initially, 50 tracks are allocated to the data set, but when this space is exhausted, more tracks are added, 30 at a time. The SPACE parameter must be specified when the library is created, and it must include allocation of space for the directory. SPACE cannot be specified when new members are added. If additional space is required when new members are added, the secondary allocation, if specified, will be used. Three 256-byte records are to be used for the directory. The volume serial number of the volume on which the library is to reside, is 111111.

Example 7: Replacing a member of an existing library.

```
//SYSLMOD   DD DSNAME=MYLIB(CASE3),DISP=OLD
```

This DD statement replaces the member named CASE3 with a new member with the same name. If the named member does not exist in the library, the member is added as a new member. In the foregoing example, the library is cataloged.

Example 8: Creating and adding a member to a library used only for the current job.

```
//SYSLMOD   DD DSNAME=&&USERLIB(MYPROG),DISP=(,PASS),UNIT=SYSDA,        X
//             SPACE=(TRK,(50,,1))
```

This DD statement creates and adds a member to a temporary library. It is similar to the DD statement shown in Example 6, except that a temporary name is used and the data set is not cataloged nor kept but is simply passed to the next job step. Since the data set is to be used only for this one job, it is not necessary to specify VOLUME and LABEL information. This statement can be used for a linkage edit job step in which the module is to be passed to the next step.

Note: If DISP=(,DELETE) is specified for a library, the entire library will be deleted.

RETRIEVING PREVIOUSLY CREATED NON-VSAM DATA SETS

The parameters that must be specified in a DD statement to retrieve a previously created data set depend on the information that is available to the system about the data set. For example,

1. If a data set on a magnetic-tape or mass storage volume was created and cataloged in a previous job or job step, all information for the data set, such as volume, space, etc., is stored in the catalog and data set label. This information need not be repeated. Only the dsname and disposition parameters need be specified.

2. If the data set was created and kept in a previous job but has not been cataloged, information concerning the data set, such as space, record format, etc., is stored in the data set label. However, the unit and volume information must be specified unless available elsewhere.

3. If the data set was created in the current job step, or in a previous job step in the current job, the information in the previous DD statement is available to the system and is accessible by referring to the previous DD statement. Only the dsname and disposition parameters need be specified.

Note: A programmer may wish to change the previous disposition of a data set. For example, if KEEP was specified when the data set was created, the DD statement that retrieves the data set may change the disposition by specifying CATLG.

Figure 49 shows the parameters that are used to retrieve previously created data sets.

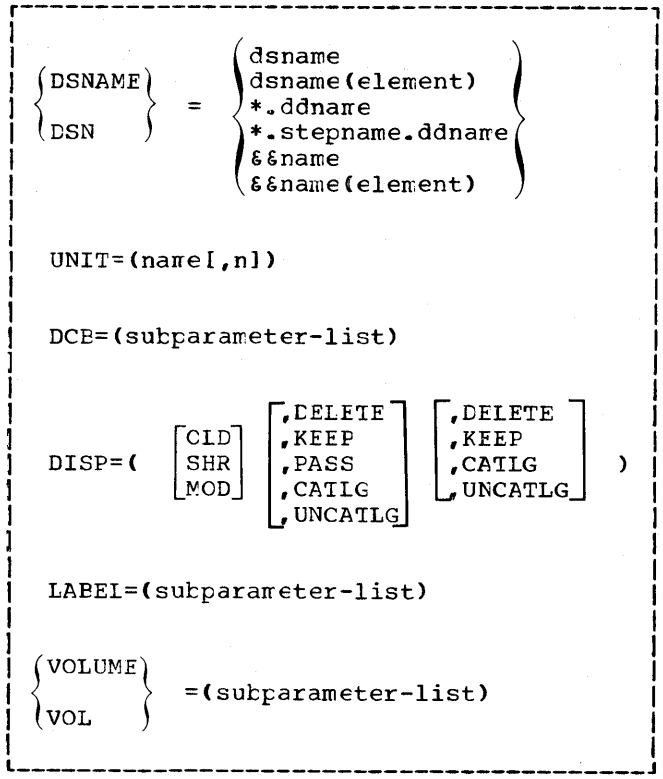


Figure 49. Parameters Frequently Used in Retrieving Previously Created Data Sets

Retrieving Cataloged Data Sets

Input data sets, assigned a disposition of CATLG or cataloged by the IEHPROGM utility program, are retrieved using the DD statement parameters DSNAME, DISP, LABEL, and DCB. The device type, volume serial number, and data set sequence number (if tape) are stored in the catalog.

DSNAME: Required. The data set must be identified by its cataloged name. If the catalog contains more than one index level, the data set name must be fully qualified.

DISP: Required. The status (OLD or SHR) of the data set must be given and an indication made as to how it is to be treated after its use, unless it is to remain cataloged. The programmer can specify as the third term in the DISP parameter a conditional disposition to indicate how the data set is to be treated if the job step abnormally terminates.

LABEL: Required only if the data set does not have a standard label.

DCB: Required only if complete data control block information is not specified by the processing program and the data set label. To save recoding time, DCB attributes can be copied from an existing DCB parameter and modified if necessary. Valid DCB subparameters are listed in "Appendix C: Fields of the Data Control Block."

Note: In addition to the disposition UNCATLG, a cataloged data set can be passed to a later step (PASS) or deleted (DELETE).

Retrieving Noncataloged (KEEP) Data Sets

Input data sets that were assigned a disposition of KEEP are retrieved by their tabulated name and location, using the DD statement parameters DSNAME, UNIT, VOLUME, DISP, LABEL, and DCB.

DSNAME: Required. The data set must be identified by the name assigned to it when it was created.

UNIT: Required, unless VOLUME=REF is used. The unit must be identified by its address, type, or group name. If the data set requires more than one unit, give the number of units. Deferred volume mounting and unit separation can be requested with this parameter.

VOLUME: Required. The volume(s) must be identified with serial numbers or, if the data set was retrieved earlier in the same job, with VOLUME=REF. If the volume is to be PRIVATE, it must be so designated. If a private volume is to remain mounted until a later job step uses it, RETAIN should be designated.

DISP: Required. The status (OLD or SHR) of the data set must be given and an indication made as to how it is to be treated after its use. The programmer can specify conditional disposition as the third term in the DISP parameter to indicate how the data set is to be treated if the job step abnormally terminates.

LABEL: Required if the data set does not have a standard label. If the data set resides with others on tape, its sequence number must be given.

DCB: Required for all indexed sequential data sets. Otherwise, required only if complete data control block information is not supplied by the processing program and the data set label. To save recoding time, copy DCB attributes from an existing DCB parameter, and modify them if necessary. Valid DCB subparameters are listed in Appendix C.

Retrieving Passed Data Sets

Input data sets used in a previous job step and passed are retrieved using the DD statement parameters DSNAME, DISP, and UNIT. The data set's unit type, volume location, and label information remain available to the system from the original DD statement.

DSNAME: Required. The original data set must be identified by either its name or the DD statement reference term *.stepname.ddname. If the original DD statement occurs in a cataloged procedure, the procedure stepname must be included in the reference term.

DISP: Required. The data set must be identified as OLD, and an indication made as to how it is to be treated after its use. The programmer can specify conditional disposition as the third term in the DISP parameter to indicate how the data set is to be treated if the job step abnormally terminates.

UNIT: Required only if more than one unit is allocated to the data set.

Retrieving Data through an Input Stream

Data sets in the form of decks of cards or groups of card images can be introduced to the system through an input stream by interspersing them with control statements. To define a data set in the input stream, mark the beginning of the data set with a DD statement and the end with a delimiter statement. The DD statement must contain one of the parameters * or DATA. Use DATA if the data set contains job control statements and an * if it does not. Two DCB subparameters can also be coded when defining a data set in the input stream: BLKSIZE and BUFNO. Coding the DLM parameter permits termination of data with a delimiter other than /*.

Notes:

- The input stream can be on any device supported by QSAM.
- Each job step and procedure step can be associated with several data sets in an input stream. All such data sets except the first in the job must be preceded by DD * or DD DATA statements.
- The characters in the records must be coded in BCD or EBCDIC.
- If the data is preceded with a DD * statement, a /* delimiter following the data is optional.

Examples of DD Statements Used to Retrieve Data Sets

Example 1: Retrieving a cataloged data set.

```
//CALC     DD   DSNAME=PROCESS,DISP=(OLD,PASS,KEEP)
```

This DD statement retrieves a cataloged data set named PROCESS. No UNIT or VOLUME information is needed. Since PASS is specified, the volume in which the data set is written is retained at the end of the job step. PASS implies that a later job step will refer to the data set. The last step in the job referring to the data set should specify the final disposition. If no other DD statement refers to the data set, it is assumed that the status of the data set is as it existed before this job. In the event of an abnormal termination, the KEEP disposition explicitly states the disposition of the data set.

Example 2: Retrieving a data set that was kept but not cataloged.

```
//TEMPFILE DD   DSNAME=FILEA,UNIT=DIRECT,VOLUME=SER=AA70,DISP=OLD
```

This DD statement retrieves a kept data set named FILEA. (This data set is created by the DD statement shown in Example 3 for creating data sets.) The data set resides on a device in a hypothetical device class, DIRECT. The volume serial number is AA70.

Example 3: Referring to a data set in a previous job step.

```
//SAMPLE   JOB
//STEP1    EXEC PGM=IKFCBL00,PARM=DECK
           .
           .
           .
//SYSLIN   DD   DSNAME=ALPHA,DISP=(NEW,PASS),UNIT=SYSSQ
//STEP2    EXEC PGM=IEWL
//SYSLIN   DD   *.STEP1.SYSLIN,DISP=(OLD,DELETE)
```

The DD statement SYSLIN in STEP2 refers to the data set defined in the DD statement SYSLIN in STEP1.

Example 4: Retrieving a member of a library.

```
//BANKING  DD   DSNAME=PAYROLL(HOURLY),DISP=OLD
```

The DD statement retrieves a member, HOURLY, from a cataloged library, PAYROLL.

DD STATEMENTS THAT SPECIFY UNIT RECORD DEVICES

A DD statement may simply indicate that data follows in the input stream or that the data set is to be punched or printed. Figure 50 shows the parameters of special interest for these purposes.

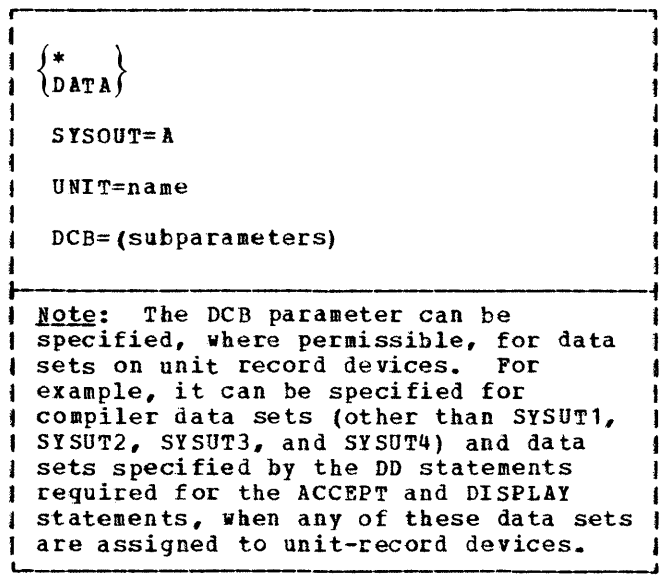


Figure 50. Parameters Used To Specify Unit Record Devices

Example 1: Specifying data in the card reader.

```
//SYSIN     DD *
```

The asterisk indicates that data follows in the input stream. The data must be followed by a delimiter statement if it contains // or /* in columns 1 and 2.

Example 2: Specifying a printer data set.

```
//SYSPRINT  DD SYSOUT=A
```

SYSOUT is the system output parameter; A is the standard device class for printer data sets.

Example 3: Specifying a card punch.

```
//SYSPUNCH  DD SYSOUT=B
```

B is the standard device class for punch devices.

CATALOGING A DATA SET

A data set is cataloged whenever CATLG is specified in the DISP parameter of the DD statement that creates or uses it. This means that the name and volume identification for the data set are placed in a system index called the catalog. (See "Processing with QISAM" in the section "Execution Time Data Set Requirements" for information about cataloging indexed sequential data sets.) The information stored in the catalog is always available to the system; consequently, only the data set name and disposition need be specified in subsequent DD statements that retrieve the data set. See Example 4 in "Creating Non-VSAM Data Sets," and Example 1 in "Retrieving Non-VSAM Data Sets."

If DELETE is specified for a cataloged data set, any reference to the data set in the catalog is deleted unless the DD statement containing DELETE retrieves the data set in some way other than by using the catalog. If UNCATLG is specified for a cataloged data set, only the reference in the catalog is deleted; the data set itself is not deleted.

Note: A "cataloged data set" should not be confused with a "cataloged procedure" (see "Using the Cataloged Procedures").

GENERATION DATA GROUPS

It is sometimes convenient to save data sets as elements or generations of a generation data group (DSNAME=dsname (element)). A generation data group is a collection of successive, historically related data sets. Identification of data sets that are elements of a generation data group is based upon the time the data set is added as an element. That is, a generation number is attached to the generation data group name to refer to a particular element. The name of each element is the same, but the generation number changes as elements are added or deleted. The most recent element is 0, the element added previous to 0 is -1, the element added previous to -1 is -2, etc. A generation data group must always be cataloged.

For example, a data group named PAYROLL might be used for a weekly payroll. The elements of the group are:

PAYROLL(0)
PAYROLL(-1)
PAYROLL(-2)

where PAYROLL(0) is the data set that contains the information for the most current weekly payroll, and is the most recent addition to the group.

When a new element is added, it is called element(+n), where n is an integer greater than 0. For example, when adding a new element to the weekly payroll, the DD statement defines the data set to be added as PAYROLL(+1); at the end of the job the system changes its name to PAYROLL(0). The element that was PAYROLL(0) at the beginning of the job becomes PAYROLL(-1) at the end of the job, and so on.

If more than one element is being added in the same job, the first is given the number (+1), the next (+2) and so on.

NAMING DATA SETS

Each data set must be given a name. The name can consist of alphanumeric characters and the special characters, hyphen and the +0 (12-0 multipunch). The first character of the name must be alphabetic. The name can be assigned by the system, it can be given a temporary name, or it can be given a user-assigned name. If no name is specified on the DD statement that creates the data set, the system assigns to the data set a unique name for the job step. If a data set is used only for the duration of one job, it can be given a temporary name (DSNAME=&&name). If a data set is to be kept but not cataloged, it can be given a simple name. If the data set is to be cataloged it should be given a fully qualified data set name. The fully qualified data set name is a series of one or more simple names joined together so that each represents a level of qualification. For example, the data set name DEPT999.SMITH.DATA3 is composed of three simple names that are separated by periods to indicate a hierarchy of names. Starting from the left, each simple name indicates an index or directory within which the next simple name is a unique entry. The rightmost name identifies the actual location of the data set.

Each simple name consists of one to eight characters, the first of which must be alphabetic. The special character period (.) separates simple names from each other. Including all simple names and periods, the length of a data set name must not exceed 44 characters. Thus, a maximum of 21 qualification levels is possible for a data set name.

Programmers should not use fully qualified data set names that begin with the letters SYS and that also have a P as the nineteenth character of the name. Under certain conditions, data sets with the above characteristics will be deleted.

EXTENDING NON-VSAM DATA SETS

A processing program can extend an existing data set by adding records to it. If the EXTEND phrase of the OPEN statement is specified (QSAM data sets only), COBOL positions the data set immediately following the last logical record. Subsequent WRITE statements then add records as though the data set had been opened with the OUTPUT phrase. (If LINAGE was specified, the initial position at the time of the OPEN EXTEND is assumed to be at the beginning of a page.) The DD statement for the data set to be extended need be no different than for a normal COBOL output file. Although the user need not specify DISP=MOD on the DD statement, the system implements the EXTEND request as if it had been; consequently, any system restrictions for DISP=MOD also apply to the EXTEND file.

When OPEN EXTEND is not specified in the COBOL program, a sequential data set (QSAM or other) can still be extended by including DISP=MOD on the DD statement for the data set's retrieval. When MOD is specified, the system positions the appropriate read/write head after the last record in the data set.

If a disposition of CATLG for an extended data set that is already cataloged is indicated, the system updates the catalog to reflect any new volumes caused by the extension. When extending a multivolume data set where number of volumes might exceed the number of units used, the programmer should either specify a volume count or deferred mounting as part of the volume information. This ensures data set extension to new volumes.

ADDITIONAL FILE PROCESSING INFORMATION

The following topics are discussed in this section: the data control block, error processing for COBOL files, and volume and data set labels.

More information about input/output processing is contained in the publication OS/VS Data Management Services Guide.

DATA CONTROL BLOCK

Each non-VSAM data set is described to the operating system by a data control block (DCB). A data control block consists of a group of contiguous fields that provide information about the data set to the system for scheduling and executing input/output operations. The fields describe the characteristics of the data set (e.g., data set organization) and its processing requirements (e.g., whether the data set is to be read or written). The COBOL compiler creates a skeleton DCB for each data set and inserts pertinent information specified in the Environment Division, FD entry, and input/output statements in the source program. The DCB for each file is part of the object module that is generated. Subsequently, other sources can be used to enter information into the data control block fields. The process of filling in the data control block is completed at execution time.

Additional information that completes the DCB at execution time may come from the DD statement for the data set and, in certain instances, from the data set label when the file is opened.

Overriding DCB Fields

Once a field in the DCB is filled in by the COBOL compiler, it cannot be overriddenby a DD statement or a data set label. For example, if the buffering factor for a data set is specified in the COBOL source program by the RESERVE clause, it cannot be overridden by a DD statement. In the same way, information from the DD statement cannot be overridden by information included in the data set label.

Identifying DCB Information

The links between the DCB, DD statement, data set label, and input/output statements are the filename, the system name in the ASSIGN clause of the SELECT statement, the ddname of the system-name, and the dsname (Figure 51).

1. The filename specified in the SELECT statement and in the FD entry of the COBOL source program is the name associated with the DCB.
2. Part of the system-name specified in the ASSIGN clause of the source program is the ddname link to the DD statement. This name is placed in the DCB.
3. The dsname specified in the DD statement is the link to the physical data set.

The fields of the data control block are described in the tables in Appendix C. They identify those fields for which information must be supplied by the source program, by a DD statement, or by the data set label. For further information about the data control block, see the discussion of the DCB macro instruction for the appropriate file processing technique in the publication OS/VS Data Management Services Guide.

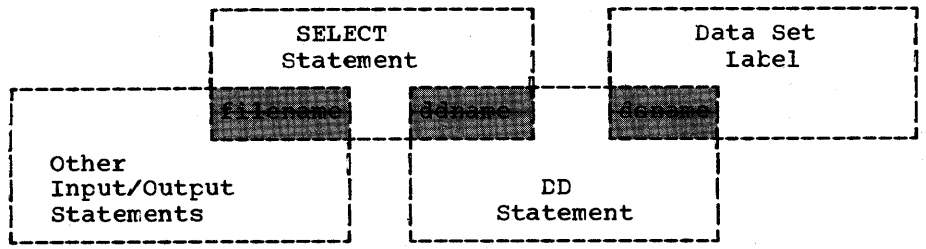


Figure 51. Links between the SELECT Statement, the DD Statement, the Data Set Label, and the Input/Output Statements

ERROR PROCESSING FOR NON-VSAM COBOL FILES

The actions taken after an I/O error depend on a number of things:

- What access method is being used
- What type of error it is
- What sort of error handling statements the program contains (if any)
- For certain error types on a QSAM file, what DCB EROPT subparameter was specified

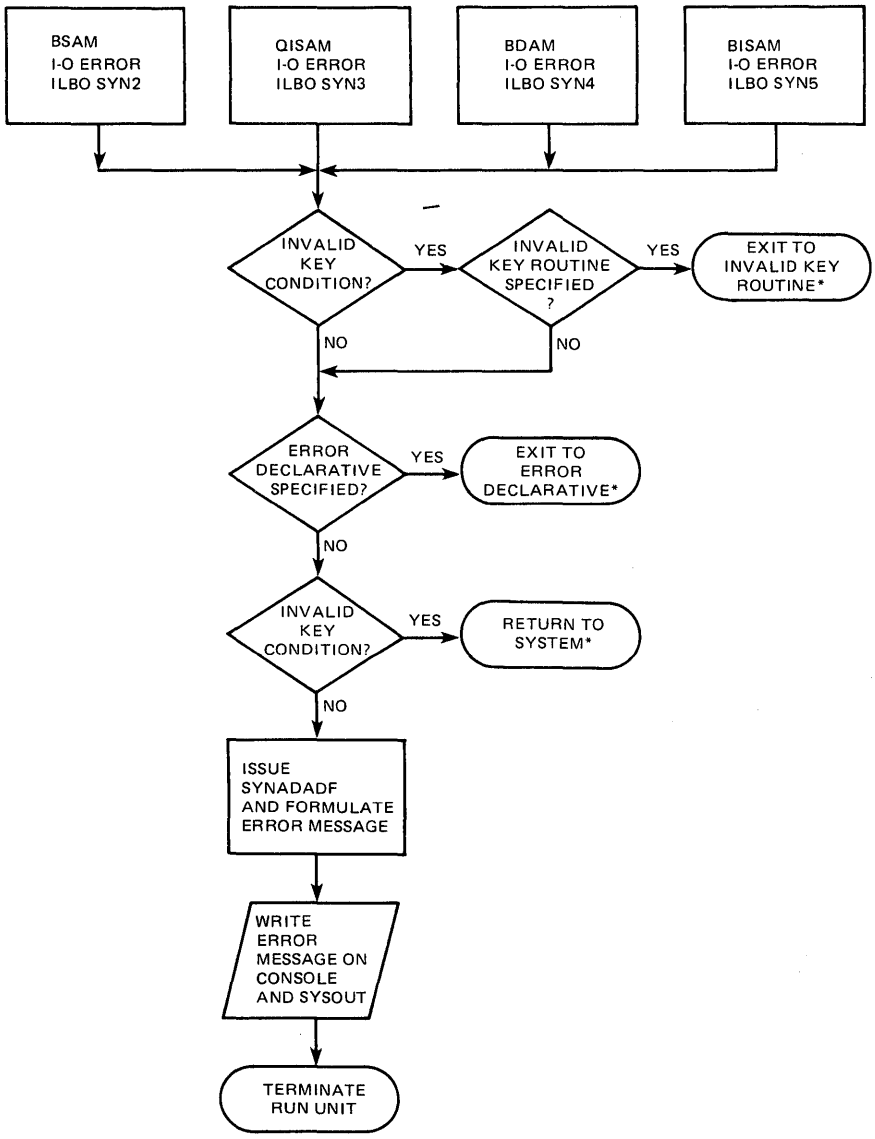
If an error declarative, a file status clause, or an invalid key clause is specified for a particular file, the DCBSYNAD field of the data control block for that file contains the address of an entry point in ILBOSYN0 (COBOL's error intercept library subroutine). ILBOSYN0 has 5 entry points (ILBOSYN1, 2, 3, 4, and 5) corresponding to each of the five access methods supported (other than VSAM); they are QSAM, BSAM, BDAM, QISAM, and BISAM. If no error declarative, file status clause, or invalid key clause is specified, no SYNAD routine is activated and the appropriate system action is taken.

In general, error handling for BSAM, QISAM, BDAM, and BISAM is identical. Figure 52 shows what happens when the access method has detected an error and given control to the COBOL SYNAD routine. The program will either continue at the next statement following the I/O statement that caused the error, or abend -- depending on the type of error and the error handling language in the program.

Error handling for QSAM files, however, is different and more complex, because of these additional factors:

1. The choice of EROPT subparameter (whether ACC, SKP, or ABE) affects some conditions
2. The presence or absence of a FILE STATUS clause affects most conditions
3. COBOL itself does some error checking, and handles conditions it finds differently

Figure 53 shows this logic flow. Note that there are three general possibilities, depending on whether the error is a QSAM-detected space problem, some other type of QSAM-detected problem, or a COBOL-detected problem. Errors in the last category are such things as OPEN and CLOSE failures, attempts to read/write/rewrite on an unopened file, attempts to read past end-of-file, and the like. (These are errors that would fall into the FILE STATUS classifications of 90 or higher.) A program encountering an error on a QSAM file will continue, abend, or terminate the job step with a return code of 12.



*EXECUTION OF COBOL PROGRAM THEN CONTINUES FOLLOWING THE I/O STATEMENT THAT RAISED THE ERROR.

Figure 52. Flow of Control in COBOL After Error Detected on BSAM/QISAM/BDAM/BISAM I/O

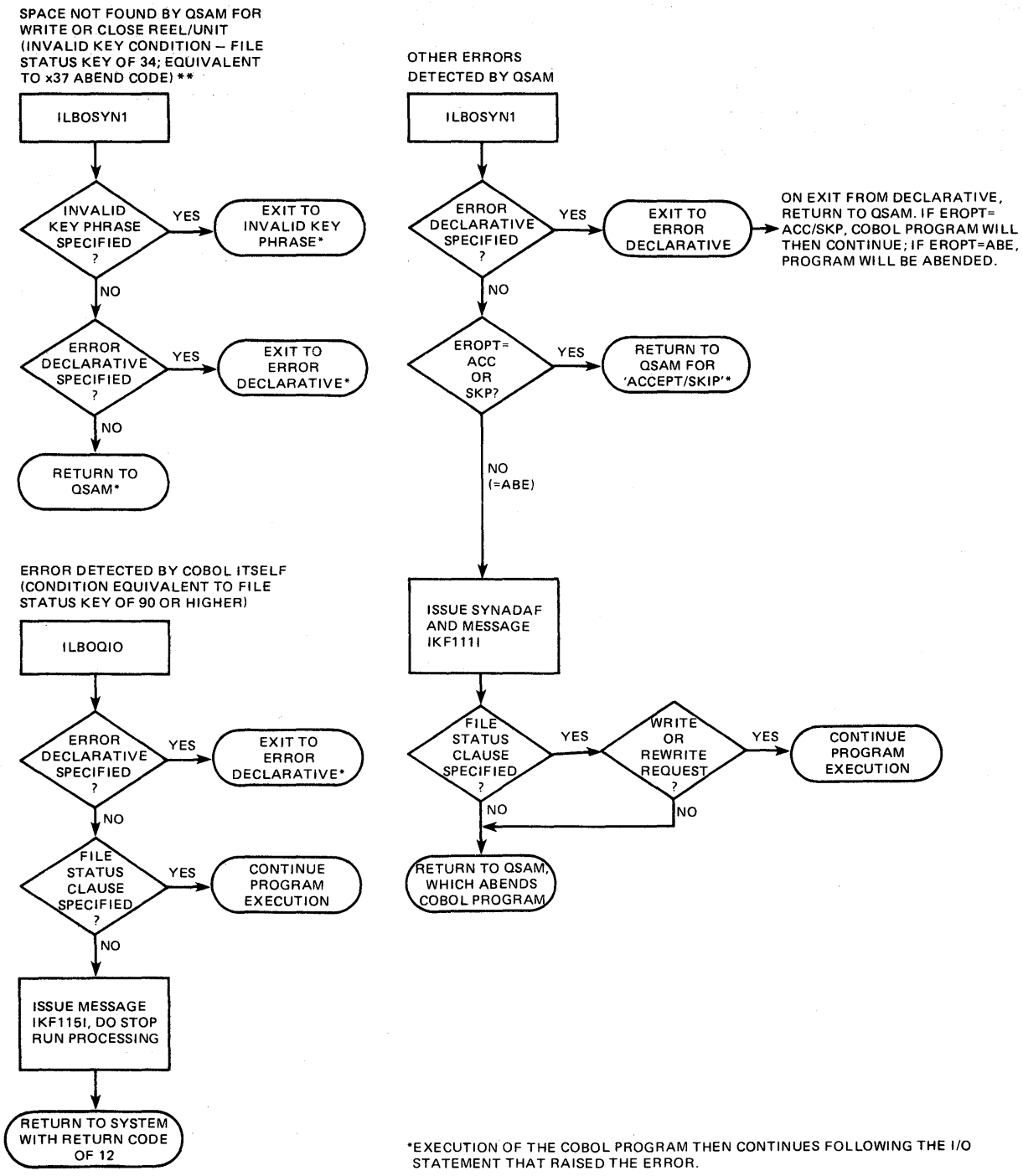


Figure 53. Flow of Control in COBOL After Error Detected on QSAM I/O

COBOL Language Features for Input/Output Error Processing

The COBOL programmer has at his disposal several language features which enable him to be notified of an input/output error and its exact nature -- information he can utilize to attempt a retry if possible.

These language features are: the FILE STATUS clause, the INVALID KEY clause, and the USE AFTER ERROR declarative. See IBM VS COBOL for OS/VS for complete details on the proper use of these features.

FILE STATUS Key

The FILE STATUS clause may be specified for QSAM files in order to provide a means of testing the success of individual I/O operations and determining more closely the specific nature of an error condition when it arises. QSAM FILE STATUS may be used alone, or in conjunction with the error declarative procedure (described below). In the latter case, the FILE STATUS key is set by COBOL before the error declarative is given control.

QSAM status key values are explained in the table below.

INVALID KEY Option

The INVALID KEY option, when specified, applies only for the specific input/output operation requested; when the operation is terminated successfully, the INVALID KEY clause is nullified. This means that the file has no constant INVALID KEY routine associated with it, but that the active INVALID KEY routine is the one specified on the current input/output verb. For files accessed randomly and for output files accessed sequentially, an INVALID KEY clause may be specified with the verbs READ, WRITE, and START.

Note: The INVALID KEY routine is given control only for input/output errors specifically attributed to an INVALID KEY condition; no return is made to the system from such a routine. Hence, this option merely informs the programmer that one of these conditions arose -- this may be all he needs to know -- and no direct return to the system is possible, nor is any further information on the nature of the error available at the COBOL source level.

INVALID KEY conditions (those which pass control to INVALID KEY clauses), as interpreted by COBOL, vary with the access method used, but may be generally described as follows: record not found, duplicate record, and sometimes space for output not found.

The Error Declarative

A single USE AFTER STANDARD ERROR declarative defines, for a certain subset of the files within a COBOL program, a series of operations to be performed for the express purpose of determining the exact nature of an input/output error and attempting to recover from that error. This series of operations may cover as broad or restrictive a subset of the files as the programmer wishes. A single declarative may be applied to all OUTPUT, all INPUT, or all I-O files, a single file, or any number of files; it is the programmer's responsibility to ensure no conflict of applicability arises.

For example, within the same program, a single file may be opened as OUTPUT, INPUT, and I-O at different states of processing. Three separate error declaratives may be coded for this file, since the proper declarative will be activated just before the file is opened. For additional information on declaratives, see IBM VS COBOL for OS/VS.

The declarative also offers a very potent and flexible tool for determining the nature of input/output errors; it is the GIVING option which may be specified for any of the subsets of files mentioned earlier. Within the GIVING clause, data-name-1 is specified and will contain a 136-byte descriptive error message after an input/output error on a READ, WRITE, REWRITE, or START verb. This field must reside in working storage. It will contain spaces for logic errors on QSAM files.

STATUS KEY 1		STATUS KEY 2	
Value	Meaning	Value	Meaning
0	Successful completion	0	No further information
1	At end (no next logical record, or an OPTIONAL file not available at OPEN time)	0	No further information
3	Permanent error (data check, parity check, transmission error)	0	No further information
		4	Space not found to add requested output record; for example, file's extents could be exhausted
9	Other errors	0	OPEN or CLOSE failed (OPEN failure could result from missing DD card)
		2	Logic error; for example, attempt to open a file already open, attempt to open a file previously closed with a lock, attempt to close a file already closed or never successfully opened, attempt to read/write/rewrite on an unopened file or a file opened in the wrong mode (e.g., WRITE on file opened INPUT), attempt to read after end-of-file has been reached

Data-name-2 can be specified optionally and will contain the block in error if data transfer actually took place; this restricts its applicability to input operations -- specifically, READs. Data-name-2 may be defined in the Working-Storage Section or in the Linkage Section. Defining it in the Working-Storage Section requires that enough storage be reserved within the program's data area to contain the block in question. The block will be moved into data-name-1 from the buffer if the system indicates data transfer did take place; otherwise, no move will be executed. Defining data-name-2 in the Linkage Section results in space being reserved within the object program only for 4 bytes per 4K bytes of the block ; the block is referenced in the buffer by means of these cells.

Besides storage and performance considerations, one reason for defining data-name-2 in the Linkage Section is that data-name-1 should be examined thoroughly before any reference to data-name-2 is made. For example, if a declarative specified for a single file, which is opened INPUT and OUTPUT, within the program is entered because an input/output error occurred on a WRITE verb for that file, an attempt to access data-name-2 will result in an abnormal termination. Data-name-1 should be checked for input/output operation, access method, block address, and so on, before data-name-2 is referenced at all. Figure 54 shows an example of this type of checking.

Figure 54 shows a single error declarative which has been specified for two files, one QSAM assigned to tape and one BISAM. For simplicity, processing of each file is kept separate in the declarative, even though some common processing could have been done. Each file has the same logical record length; if they had different lengths, a problem might arise when the entry coding of the declarative attempts to move the offending block into data-name-2.

```
00001          IDENTIFICATION DIVISION.
00002          REMARKS.
00003              THIS PROGRAM CAUSES I-O ERRORS FOR A QSAM AND ISAM FILE AND
00004              WILL DEMONSTRATE THE USE OF THE ERROR DECLARATIVE.
00005          ENVIRONMENT DIVISION.
00006          INPUT-OUTPUT SECTION.
00007          FILE-CONTROL.
00008              SELECT QSAM-FILE,
00009                  ASSIGN TO UT-S-QSAMFILE.
00010              SELECT BISAM-FILE,
00011                  ASSIGN TO DA-I-BISMFILE,
00012                  ACCESS IS RANDOM,
00013                  NOMINAL KEY IS NOMINAL-KEY,
00014                  RECORD KEY IS RECORD-KEY.
00015          DATA DIVISION.
00016          FILE SECTION.
00017          FD  QSAM-FILE,
00018              RECORDING MODE IS F,
00019              LABEL RECORDS ARE OMITTED,
00020              RECORD CONTAINS 100 CHARACTERS,
00021              DATA RECORD IS QSAM-RECORD.
00022          01  QSAM-RECORD PIC X(100).
00023          FD  BISAM-FILE,
00024              RECORDING MODE IS F,
00025              LABEL RECORDS ARE STANDARD,
00026              RECORD CONTAINS 80 CHARACTERS,
00027              DATA RECORD IS BISAM-RECORD.
00028          01  BISAM-RECORD.
00029              05  FILLER PIC X.
00030              05  RECORD-KEY PIC X(5).
00031              05  FILLER PIC X(74).
00032          WORKING-STORAGE SECTION.
00033          01  SYNAD-RECORD COPY ERRSRCD1.
00034          01  SYNAD-RECORD.
00035         *
00036         *    THIS RECORD IS FORMATTED TO SHOW ALL FIELDS IN DATA-NAME-1
00037         *    (THE SYNADAF MESSAGE). NOT ALL FIELDS WILL BE REFERENCED
00038         *    IN A PARTICULAR PROGRAM.
00039         *
00040              05  FILLER PIC X(8).
00041              05  INPUT-BUFFER-ADDRESS PIC X(4).
00042              05  NUMBER-OF-BYTES-READ PIC 9(4) USAGE COMP.
00043              05  FILLER PIC X(36).
00044              05  JOBNAME PIC X(8).
00045              05  FILLER PIC X.
00046              05  STEPNAME PIC X(8).
00047              05  FILLER PIC X.
00048              05  UNIT-ADDRESS PIC X(3).
00049              05  FILLER PIC X.
00050              05  DEVICE-TYPE PIC XX.
00051                  88  MASS-STORAGE-DEVICE  VALUE 'DA'.
00052                  88  MAGNETIC-TAPE-DEVICE VALUE 'TA'.
00053                  88  UNIT-RECORD-DEVICE   VALUE 'UR'.
00054              05  FILLER PIC X.
00055              05  DDNAME PIC X(8).
00056              05  FILLER PIC X.
00057              05  OPERATION-ATTEMPTED PIC X(6).
```

Figure 54. Example of Use of GIVING Option in Error Declarative (Part 1 of 3)

```
00058              05  FILLER PIC X.
00059              05  ERROR-DESCRIPTION PIC X(15).
00060                  88  WLR-ERROR VALUE 'WRNG.LEN.RECORD'.
00061                  88  INVLD-REQ VALUE 'INVALID REQUEST'.
00062              05  FILLER PIC X.
00063              05  ACCESS-METHOD-DATA.
00064                  10  UNIT-RECORD.
00065                      15  FILLER PIC X(15).
00066                      15  UR-ACCESS-METHOD PIC X(6).
00067                  10  MAGNETIC-TAPE REDEFINES UNIT-RECORD.
00068                      15  RELATIVE-BLOCK-NUMBER PIC 9(7).
00069                      15  FILLER PIC X.
00070                      15  TA-ACCESS-METHOD PIC X(5).
00071                      15  FILLER PIC X(8).
00072                  10  MASS-STORAGE REDEFINES UNIT-RECORD.
00073                      15  LAST-ACTUAL-ADDRESS PIC X(14).
00074                      15  FILLER PIC X.
00075                      15  DA-ACCESS-METHOD PIC X(6).
00076              05  SYSTEM-USE PIC X(8).
00077          01  NOMINAL-KEY PIC 9(5).
00078          LINKAGE SECTION.
00079          01  ERROR-DATA PIC X(100).
00080          PROCEDURE DIVISION.
00081          DECLARATIVES.
00082          ERROR-EXAMPLE SECTION.
00083              USE AFTER STANDARD ERROR PROCEDURE
00084                  ON QSAM-FILE, BISAM-FILE,
00085                  GIVING SYNAD-RECORD, ERROR-DATA.
00086              DISPLAY '** ERROR DECLARATIVE ENTERED FOR'
00087                  OPERATION-ATTEMPTED ' OPERATION **'.
00088              DISPLAY SYNAD-RECORD.
00089              IF ERROR-DESCRIPTION IS NOT EQUAL TO 'UNKNOWN COND'
00090                 AND
00091                  OPERATION-ATTEMPTED IS NOT EQUAL TO "UNKNOWN"
00092                      GO TO DECLARATIVE-1.
00093          *
00094          *    IF OPERATION OR ERROR TYPE IS UNKNOWN, CALL ASSEMBLY
00095          *    LANGUAGE ROUTINE TO CHECK MORE DEEPLY.
00096          *
00097              IF DA-ACCESS-METHOD IS EQUAL TO 'QSAM ',
00098                  CALL 'ERRANAL',
00099                      USING SYSTEM-USE, QSAM-FILE, ACCESS-METHOD-DATA;
00100                ELSE
00101                  CALL 'ERRANAL',
00102                      USING SYSTEM-USE, BISAM-FILE, ACCESS-METHOD-DATA.
00103              GO TO DECLARATIVE-EXIT.
```

Figure 54. Example of Use of GIVING Option in Error claratve (Part 2 of 3)

```
00105          DECLARATIVE-1.
00106         *
00107         *    CHECK EXPECTED QSAM OR BISAM ERROR CONDITION.
00108         *
00109              IF WLR-ERROR,
00110                  DISPLAY '** WRONG LENGTH RECORD ERROR AS EXPECTED **',
00111                  GO TO DECLARATIVE-EXIT.
00112         *
00113         *    CHECK ONE PARTICULAR ERROR POSSIBILITY, THEN DISPLAY BLOCK
00114         *    IN ERROR IF POSSIBLE, THEN EXIT.
00115         *
00116              IF DA-ACCESS-METHOD IS NOT EQUAL TO 'BISAM',
00117                  GO TO DECLARATIVE-2.
00118              IF NOT INVLD-REQ,
00119                  GO TO DECLARATIVE-EXIT.
00120         *
00121         *    BISAM INVALID REQUEST. EXIT DECLARATIVE VIA 'GO TO'.
00122         *
00123              CLOSE BISAM-FILE.
00124              GO TO PROCESS-NEXT-FILE.
00125          DECLARATIVE-2.
00126              IF INPUT-BUFFER-ADDRESS IS EQUAL TO SPACES,
00127                  CLOSE QSAM-FILE,
00128                  GO TO PROCESS-BISAM-FILE.
00129              DISPLAY ERROR-DATA.
00130          DECLARATIVE-EXIT.
00131              EXIT.
00132          END DECLARATIVES.
00133              OPEN INPUT QSAM-FILE.
00134              READ QSAM-FILE,
00135                AT END
00136                  CLOSE QSAM-FILE,
00137                  STOP RUN.
00138          PROCESS-BISAM-FILE.
00139              OPEN I-O BISAM-FILE.
00140              MOVE 100 TO NOMINAL-KEY.
00141              REWRITE BISAM-RECORD.
00142              CLOSE BISAM-FILE.
00143          PROCESS-NEXT-FILE.
00144              STOP RUN.
```

Figure 54. Example of Use of GIVING Option in Error Declarative (Part 3 of 3)

Since it is the receiving field, the length of data-name-2 is taken as the length of the move. Therefore, when using the same error declarative for files, ensure the files do not have differing attributes.

SYNAD-RECORD, a 136-byte area set aside to receive the error message provided by the system, is compatible across all access methods. An area of some variation is ACCESS-METHOD-DATA, which is device dependent. SYNAD-RECORD is shown in great detail for the purpose of showing how each field might be coded. Such exhaustive detail, however, is not necessary in many applications, and only the fields to be referenced need be explicitly described at all. This detailed map of the area would be a good skeleton for a COPY library member, once it is standardized for an installation.

When data has been transferred, INPUT-BUFFER-ADDRESS will contain the address of such data and is the source of data-name-2. Likewise, NUMBER-OF-BYTES-READ contains the actual length of the offending block. DEVICE-TYPE is a key to the actual layout of ACCESS-METHOD-DATA, which is device dependent. The possible content of OPERATION-ATTEMPTED, ERROR-DESCRIPTION, and ACCESS-METHOD-DATA are shown in some detail in Appendix G. SYSTEM-USE is an 8-byte field which is not useful to a COBOL source program but which in most cases reflects the contents of registers 0 and 1 upon entry to the SYNAD routine and are passed as arguments (shown at line 100 in Figure 54) to the assembly language program ERRANAL for further study. The user will find additional information about the contents of this field in OS/VS Data Management Macro Instructions.

Note: Data-name-2, ERROR-DATA in the example, is specified in the Linkage Section; the generated code within the declarative will do nothing but move the address of INPUT-BUFFER-ADDRESS into a base locator cell assigned to ERROR-DATA.

Immediately upon entry to the declarative, a signal message is displayed as well as the SYNAD-RECORD. Then if an unknown operation or error is indicated in the message, the system calls ERRANAL passing SYSTEM-USE, the appropriate DCB or DECB, and ACCESS-METHOD-DATA, exiting from the declarative upon return.

If both fields are known, the system does further checking. For the purpose of this example, it is assumed that a wrong-length record is expected and this causes a message to be printed and an exit from the declarative. Finally, for the BISAM file, the coding checks the invalid request condition. If invalid, the file is closed and the declarative is exited via a GO TO statement (not a normal exit). If it is a QSAM file, the field containing the address of the data represented by data-name-2 is checked for blanks before displaying it. For a QISAM file, data-name-2 can never be referenced.

This example of error declarative technique points out some basic tools: the use of fields within data-name-1 to decide processing; the checking of the address used for data-name-2 before referring to it; the use of data-name-1 as a group item; the normal and GO TO exits from the declarative; the calling of an assembly language subroutine to perform detailed analysis of system information; forcing use of a declarative for INVALID KEY conditions by not coding an INVALID KEY clause; closing of the offending file if the nature of the error suggests such an action. A course of action not described here (but often possible) is a retry of the input/output operation that caused the error.

The INVALID KEY clause and/or the error declarative may be specified for a file, and for any given input/output error, the error intercept subroutine decides which is to be given control. Figure 55 is a generalized summary of the means available for recovery from an invalid key condition or an input/output error.

It is most important that the programmer make certain of the validity of data-name-2 before referencing it. In the event that a complete message could not be formatted by the system, the entry coding in the error declarative takes steps to avoid moving the block in error into data-name-2 if it is in the Working-Storage Section or setting up the base locator address if it is in the Linkage Section.

However, it is up to the user to avoid an invalid reference. Note also that for certain conditions, the contents of data-name-1 will be invalid (blanks or asterisks). The user may find information on this under SYNADAF macro in OS/VS Data Management Macro Instructions.

Note: The programmer should also consider the following when a relatively large number of INVALID KEY exits or declarative sequences (with GO TO exits) are to be executed:

1. The distinction between error processing via an error declarative and the the INVALID KEY clause. When an input/output operation is requested, a storage area of about 40 to 100 bytes (called an input/output

block or IOB) is allocated until the request is satisfied (or, in the event of an error, until return from the user-provided error-handling routine). If the error declarative is used, a normal exit from the declarative returns control the system and frees the IOB. When the INVALID KEY routine is used, however, the system does not regain control and the IOB is not freed.

Note: If an I/O error occurs on a WRITE or REWRITE, and a FILE STATUS clause was specified but an error declarative or INVALID KEY clause was not, then the system does not regain control and the IOB is not freed.

2. The error declarative interface dynamically allocates storage for a register save area upon entry, roughly 200 bytes. This is necessary to make the declarative serially reusable in the event of another input/output error occurring within the declarative, and for which it is specified (for example, an I/O request to another file may be done within the declarative, and this second attempt may also cause an error). If a GO TO statement is used to exit from the declarative, neither this save area nor the IOB is freed.

   To make maximum efficient use of one's address space and to make the maximum space available to other users, the programmer should rely on the error declarative as much as possible, taking a normal exit from it. Otherwise, it is recommended that the programmer specify a larger address space.

VOLUME LABELING

Various groups of labels may be used in secondary storage to identify magnetic-tape and mass storage volumes, as well as the data sets they contain. The labels are used to locate the data sets and are identified and verified by label processing routines of the operating system.

There are two different kinds of labels, standard and nonstandard. Magnetic tape volumes can have standard or nonstandard labels, or they can be unlabeled. The type(s) of label processing for tape volumes to be supported by an installation is selected during the system generation process. Mass storage volumes are supported with standard labels only.

Standard labels consist of volume labels and groups of data set labels. The volume label group precedes or follows data on the volume; it identifies and describes the volume. The data set label groups precede and follow each data set on the volume, and identify and describe the data set.

- The data set labels that precede the data set are called header labels.
- The data set labels that follow the data set are called trailer labels. They are almost identical to the header labels.
- The data set label groups can optionally include standard user labels except for ISAM files.
- The volume label groups can optionally include standard user labels for QSAM files.

Specified in COBOL Program / Error	Only INVALID KEY option	Only USE AFTER STANDARD ERROR	Both	Neither on This Statement	Neither in Entire Program
Invalid key	Go to invalid key routine	Go to user's declarative routine	Go to invalid key routine	Error ignored; return to system; next sequential instruction executed	ABEND
All other types of I/O errors	Return to system	Go to user's routine	Go to user's routine	Return to system	ABEND

Figure 55. Recovery from an Invalid Key Condition or other Input/Output Error (Non-QSAM)

	X	X	Note 1	X
	X X		X Note 2	X X
			X	X
	X		Note 1	X
	X		X	X

Notes:
1. Holds only for WRITE.
2. Error cannot be caused by an invalid key.

Figure 56. Input/Output Error Processing Facilities (Non-QSAM)

Nonstandard labels can have any format and are processed by routines provided by the programmer. Unlabeled volumes contain only data sets and tapemarks. In the job control statements, a DD statement must be provided for each data set to be processed. The LABEL parameter of the DD statement is used to describe the data set's labels.

Specific information about the contents and physical location of labels is contained in the publications OS/VS Data Management Services Guide and OS/VS Tape Labels.

STANDARD LABEL FORMAT

Standard labels are 80-character records that are recorded in EBCDIC and odd parity on 9-track tape; or in BCD and even parity on 7-track tape. The first four characters are always used to identify the labels. These identifiers are:

VOL1	--	volume label
HDR1 and HDR2	--	data set header labels
EOV1 and EOV2	--	data set trailer labels (end-of-volume)
EOF1 and EOF2	--	data set trailer labels (end-of-data set)
UHL1 to UHL8	--	user header labels
UTL1 to UTL8	--	user trailer labels

The format of the mass storage volume label group is the same as the format of the tape volume label group, except one of the data set labels of the initial volume label consists of the data set control block (DSCB). The DSCB appears in the volume table of contents (VTOC) and contains the equivalent of the tape data set header and trailer information, in addition to space allocation and other control information.

STANDARD USER LABELS

Standard user labels contain user-specified information about the associated data set. User labels are optional within the standard label groups. The format used for user header labels (UHL1-8) and user trailer labels (UTL1-8) consists of a label 80 characters in length recorded in EBCDIC on 9-track tape units, or in BCD on 7-track tape units. The first three bytes consist of the characters that identify the label: UHL for a user header label (at the beginning of a data set) or UTL for a user trailer label (at the end-of-volume or end-of-data set). The next byte contains the relative position of this label within a set of labels of the same type and can be any number from 1 through 8. The remaining 76 bytes consist of user-specified information.

User labels are generally created, examined, or updated when the beginning or end of a data set or volume (reel) is reached. User labels are applicable for sequential, direct, and relative data sets. For sequentially processed data sets, end or beginning of volume exits are allowed (i.e., "intermediate" trailers and headers may be created or examined). For direct or relative data sets, user label routines will be given control only during OPEN or CLOSE condition for a file opened as INPUT, OUTPUT, or I-O. Trailer labels for files opened as INPUT or I-O are processed when a CLOSE statement is executed for the file that has reached an AT END condition. Thus, for physical sequential data sets, the user may create, examine, or update up to eight header labels and eight trailer labels on each volume of the data set, whereas for direct or relative data sets the user may create, examine, or update up to eight header labels during OPEN and up to eight trailer labels during CLOSE. (QSAM EXTEND functions in a manner identical to OUTPUT, except that the beginning of a file label is not processed.) Note that these labels reside on the initial volume of a multi-volume data set. This volume must be mounted at CLOSE if trailer labels are to be created, examined, or updated.

When standard user label processing is desired, the user must specify the label type of the standard and user labels (SUL) on the DD statement that describes the dataset. For mass storage volumes, specification of a LABEL subparameter of SUL results in a separate track being allocated for use as a user-label track when the data set is created. This additional track is allocated at initial allocation and for sequential data sets at end-of-volume (volume switch) time. The user-label track (one per volume of a sequential data set) will contain both user header and user trailer labels.

User Label Totaling
(BSAM and QSAM only)

When creating or processing a data set with user labels on a sequential file, the programmer may develop control totals to obtain exact information about each volume of the data set. This information can be stored in his user labels. For example, a control total accumulated as the data set is created, can be stored in a user label and later compared with a total accumulated while processing a volume. The user totaling facility enables the programmer to synchronize the control data that he has created while processing a data set with records physically written on a volume. In this way, he can tell exactly what records were written. This information can also be used for accurately labeling tape reels (i.e., assigning physical adhesive labels).

To request this option, specify OPTCD=T in the DCB parameter of the DD statement. The user's TOTALING area, where control data is accumulated, is provided by the user. In this area, the user can store information on each record he writes. When an input/output operation is scheduled, the control program sets up a user TOTALED save area that preserves an image of the information in the user's TOTALING area. When the output USE LABEL declarative is entered, the values accumulated in the user's TOTALING area corresponding to the last record actually written on the volume are stored in the TOTALED area. These values can be included in user labels.

When using this facility for an output data set (i.e., when creating the data set), the programmer must update his control data in the TOTALING area prior to issuing a WRITE instruction. When subsequently using this data set for input, the programmer can accumulate the same information as each record is read. These values can be compared with the ones previously stored in the user label when the records were created.

Variable length records with APPLY WRITE-ONLY or records with SAME RECORD AREA specified require special considerations when using the TOTALING option. Since the control program determines whether a variable-length record will fit in a buffer after a WRITE instruction has been issued, the values accumulated may include one more record than is actually written on the volume. In this case, the programmer must update his TOTALING area after issuing a WRITE instruction.

User label totaling is not available with S-mode records.

NONSTANDARD LABEL FORMAT

Nonstandard labels do not conform to the standard label formats. They are designed by programmers and are written and processed by programmers. Nonstandard labels can be any length less than 4096 bytes. There are no requirements as to the length, format, contents, and number of nonstandard labels, except that the first record on the volume cannot be a standard volume label. In other words, the first record cannot be 80 characters in length with the identifier VOL1 as its first four characters.

NONSTANDARD LABEL PROCESSING

To use nonstandard labels (NSL), the system programmer must first:

- Create nonstandard label processing routines for input header labels, input trailer labels, output header labels, and output trailer labels.
- Insert these routines into the operating system.

Then the COBOL programmer must code NSL in the LABEL parameter of the DD statement at execution time.

The system verifies that the tape has a nonstandard label. Then if NSL is specified in the LABEL parameter, it loads the appropriate NSL routines. These NSL routines are entered at OPEN, CLOSE, and END-OF-VOLUME conditions by the respective executors.

For a data set opened as output, the NSL routines entered include:

- At OPEN time, a header routine to check the old header and/or create the new header;
- At CLOSE time, a trailer-creation routine;
- At EOV time, a trailer-creation routine and a header routine.

For a data set opened as input essentially the same types of routines are required.

Note: The NSL routines must observe the following conventions:

1. Follow Type-IV SVC routine conventions.
2. Use GETMAIN and FREEMAIN for work areas.
3. Be reentrant load modules.
4. Use EXCP for I/O operations and XCTL for passing control among load modules and then returning to the I/O-support routines.
5. Begin with the letters NSL if the system branches to them directly. (Other user-written modules having to do with nonstandard labels must begin with the letters IGC.)
6. Have as their entry points the first byte in each load module.

In addition, the NSL routines must write their own tapemarks, do all I/O operations necessary (via EXCP), determine when all labels have been processed, and take care of data set positioning. These routines may communicate at the LABEL source level with USE BEFORE LABEL PROCEDURE declaratives by means of linkage described under "User Label Procedure."

USER LABEL PROCEDURE

The USE...LABEL PROCEDURE statement provides the user with label handling procedures at the COBOL source level to handle nonstandard or user labels. The BEFORE option indicates processing of nonstandard labels. The AFTER option indicates processing of standard user labels. The labels must be listed as data-names in the LABEL RECORDS clause in the File Description entry for the file. When the file is opened as input, the label is read in and control is passed to the USE declarative if a USE...LABEL PROCEDURE is specified for the OPEN option or for the file. If the file is opened as output, a buffer area for the label is provided and control is passed to the USE declarative if a USE...LABEL PROCEDURE is specified for the OPEN option or for the file. For files opened as INPUT or I-O, control is passed to the USE declarative to process trailer labels when a CLOSE statement is executed for the file that has reached the AT END condition.

One of the concerns of the programmer is linkage between the nonstandard label SVC routine and the USE BEFORE LABEL PROCEDURE section. Other problems related to writing nonstandard label SVC routines are discussed in the publication OS/VS Tape Labels.

When the nonstandard label SVC routine has determined that a particular DCB has nonstandard labels, the nonstandard label routine must inspect the DCB exit list for an active entry to ensure that there is a USE BEFORE...LABEL section for this DCB and for that type of label processing. The DCB field EXLST contains a pointer to this exit list. An active entry is defined as a 1-byte code other than X'00' or X'80' followed by a 3-byte address of the appropriate label section (Figure 57).

Code	Exit List
1	(USE section for header labels)
2	(USE section for trailer labels)
.	.
.	.
.	.

Notes:
1. Code 1 is set to X'01' indicating INPUT, or X'02' indicating OUTPUT.
2. Code 2 is set to 'X'OD' indicating INPUT, or X'04' indicating OUTPUT.

Figure 57. Exit List Codes

Once the nonstandard label SVC routine tests that the exit list confirms an appropriate active entry, it must pass the address of a parameter list in register 1.

The parameter list (Figure 58) must have the following format.

	1 byte	3 bytes
Byte 0	0	A(label buffer)
Byte 4	Flag byte	A(DCB)
Byte 8	Error flag	

Figure 58. Parameter List Formats

The A(label buffer) is the address of the label record on input and the address where the label will be created on output.

The A(DCB) is the address of the DCB. The DCB contains a pointer to the DEB. The nonstandard label SVC routine must test the EOF bit in the OFLGS field of the DEB (data extent block) to determine whether to return control to the EOV or CLOSE module. Control is given to the CLOSE module only at EOF.

The error flag byte will have bit 0 set to 1 if an input/output error occurs when reading or writing a label.

When the USE BEFORE LABEL PROCEDURE section returns control to the nonstandard label SVC routine, it will pass a return code that will indicate whether or not more labels are to be processed (Figure 59). This return code is set by assigning a value to the special register LABEL-RETURN.

The maximum size of the label record is stored on a halfword boundary at the EXITLIST address +46.

The user's nonstandard label routines are responsible for all tape positioning. For multifile volumes, the user may specify a file sequence number in the LABEL parameter on the DD card. The nonstandard label routines can inspect this information in the JFCB and position the files accordingly. For additional information, see the publication OS/VS Tape Labels.

Routine Type	Return Code	Applicable Note
Input header and/or trailer	0	1
	4	2
	16	3
Output header and/or trailer	4	1
	8	2
Update header and/or trailer	8	1
	12	2
	16	3

Notes:
1. For output mode, the label is written or rewritten. For input mode, normal processing is resumed; any additional user labels are ignored.
2. Another label is read (for input mode) and control is returned to the USE BEFORE LABEL PROCEDURE section. For output mode, the labels should be written and control should be returned to the USE BEFORE LABEL PROCEDURE section. When control is returned to the nondeclarative portion, either normal processing will continue or the label section will be re-entered, depending on whether the return code is 4 or 8.
3. A return code of 16 indicates that the USE BEFORE LABEL PROCEDURE section has determined that an incorrect volume was mounted. When LABEL-RETURN is set to a nonzero value, the return code is set to 16.

Figure 59. Label Routine Return Codes

ASCII File Labels

ASCII files on magnetic tape may have American National Standard labels or American National Standard and user labels, or they may have no label. Any labels on an ASCII tape must be in ASCII code. Tapes containing a combination of ASCII and EBCDIC labels are not read. All the record formats supported (i.e., fixed, undefined, and variable) are allowed on ASCII files,

regardless of whether or not the files are labeled. Spanned records are not supported under ASCII.

When American National Standard labels are being processed, the label type must be specified in the DD statement that describes the data set. The parameter for American National Standard labels is LABEL=AL. The parameter for American National Standard and user labels is LABEL=AUL. Nonstandard labels are not permitted for ASCII files. The user may indicate no labels as LABELS=NL.

ASCII Standard Label Processing

Standard label processing for ASCII files is identical to standard label processing for files coded in EBCDIC. ASCII code is translated into EBCDIC code prior to processing.

ASCII User Label Processing

All American National Standard user labels (LABEL=AUL) are optional. ASCII files may have user header labels (UHLn) and user trailer labels (UTLn), which are processed very much like the standard user labels on EBCDIC files. However, there is no limit to the number of user labels possible at the beginning and the end of a file. No check is made on the number of labels written. It is left to the user to determine how many labels he wants written.

All user labels must be 80 bytes in length, but they may contain any user information desired.

Note: USE BEFORE STANDARD LABEL procedures are not allowed, because they are nonstandard.

User Label Exits

To create or verify user labels, the programmer must code for the file a USE AFTER STANDARD LABEL procedure.

NON-VSAM RECORD FORMATS

Logical records may be in one of four formats for a non-VSAM file: fixed-length (format F), variable-length (format V), unspecified (format U), or spanned (format S). F-mode files must contain records of equal lengths. Files containing records of unequal lengths must be V-mode, U-mode, or S-mode. Files containing logical records that are longer than physical records must be S-mode.

The record format is specified in the RECORDING MODE clause in the Data Division. If this clause is omitted, the compiler determines the record format from the record descriptions associated with the file. If the file is to be blocked, the BLOCK CONTAINS clause must be specified in the Data Division.

The prime consideration in the selection of a record format is the nature of the file itself. The programmer knows the type of input his program will receive and the type of output it will produce. The selection of a record format is based on this knowledge as well as an understanding of the type of input/output devices on which the file is written and of the access method used to read or write the file.

FIXED-LENGTH (FORMAT F) RECORDS

Format F records are fixed-length records. The programmer specifies format F records by including RECORDING MODE IS F in the file description entry in the Data Division. If this clause is omitted and both of the following are true:

- All records in the file are the same size
- BLOCK CONTAINS [integer-1 TO] integer-2... does not specify integer-2 less than the length of the maximum level-01 record

the compiler determines the recording mode to be F. All records in the file are the same size if there is only one record description associated with the file and it contains no OCCURS clause with the DEPENDING ON option; or if multiple record descriptions are all the same length.

The number of logical records within a block (blocking factor) is normally constant for every block in the file. When fixed-length records are blocked, the programmer specifies the BLOCK CONTAINS clause in the file description (FD) entry in the Data Division.

In unblocked format F, the logical record constitutes the block. The BLOCK CONTAINS clause is unnecessary for unblocked records.

Format F records are shown in Figure 60. The optional control character, represented by the letter C in Figure 60 is used for stacker selection and carriage control. When carriage control or stacker selection is desired, the WRITE statement with the ADVANCING or POSITIONING option is used to write records on the output file. In this case, one character position must be included as the first character of the record (if NOADV is specified). This position will be filled in with the carriage control or stacker select character. The type of carriage control character to be used is determined by the compiler. When only AFTER is specified, ASA control characters are used. Machine control characters are used when only BEFORE or both BEFORE and AFTER are specified. The carriage control character never appears when the file is written on the printer or punched on the card punch.

Note: Illustrations of unblocked Format F records do not take into account either the key field required when direct organization is used or ASCII block prefix considerations. See "Processing ASCII Files" for more information.

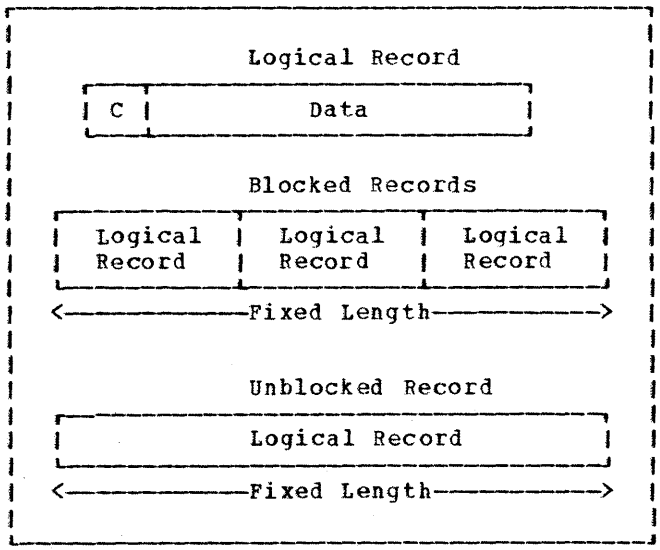


Figure 60. Fixed-length (Format F) Records

UNSPECIFIED (FORMAT U) RECORDS

Format U is provided to permit the processing of any blocks that do not conform to F, V, or S formats. Format U records are shown in Figure 61. The optional control character C, as discussed under "Fixed-Length (Format F) Records," may be used in each logical record.

The programmer specifies format U records by including RECORDING MODE IS U in the file description (FD) entry in the Data Division. U-mode records may be specified only for direct or physical sequential files.

If the RECORDING MODE clause is omitted, and BLOCK CONTAINS [integer-1 TO] integer-2... does not specify integer-2 less than the maximum level-01 record, the compiler determines the recording mode to be U if the file is direct and one of the following conditions exist:

- The FD entry contains two or more level-01 descriptions of different lengths.
- A record description contains an OCCURS clause with the DEPENDING ON option.
- A RECORD CONTAINS clause specifies a range of record lengths.

Each block on the external storage media is treated as a logical record. There are no record-length or block-length fields.

When a READ INTO statement is used for a U-mode file, the size of the longest record for that file is used in the MOVE statement. All other rules of the MOVE statement apply.

Note: Illustrations of Format U records do not take into account either the key field required when direct organization is used or ASCII block prefixes. See "Processing ASCII Files" for more information.

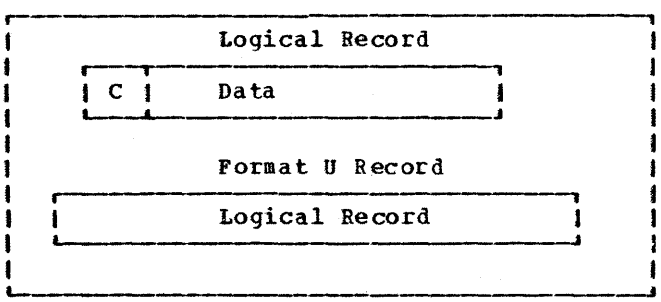


Figure 61. Unspecified (Format U) Records

VARIABLE LENGTH (FORMAT V) RECORDS

The programmer specifies format V records by including RECORDING MODE IS V in the file description entry in the Data Division. V-mode records may be specified only for direct or physical sequential files. If the RECORDING MODE clause is omitted and BLOCK CONTAINS [integer-1 TO] integer-2... does not specify integer-2 less than the maximum level-01 record, the compiler determines the recording mode to be format V if the file is physical sequential and one of the following conditions exist:

- The FD entry contains two or more level-01 descriptions of different lengths.
- A record description contains an OCCURS clause with the DEPENDING ON option.
- The RECORD CONTAINS clause specifies a range of record lengths.

V-mode records, unlike U-mode or F-mode records, are preceded by fields containing control information. These control fields are illustrated in Figures 62 and 63.

The first four bytes of each block contain control information (CC):

LL -- represents two bytes designating the length of the block (including the 'CC' field).

BB -- represents two bytes reserved for system use.

The first four bytes of each logical record contain control information (cc):

ll -- represents two bytes designating the logical record length (including the 'cc' field).

bb -- represents two bytes reserved for system use.

For unblocked V-mode records (Figure 62), the Data portion + CC + cc constitute the block.

For blocked V-mode records (Figure 63), the Data portion of each record + the cc of each record + CC constitute the block.

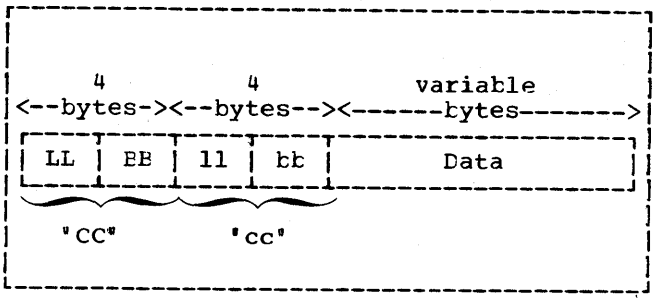


Figure 62. Unblocked V-Mode Records

Variable-length record descriptions, for input and output files, must not define space for the control bytes. Control bytes are automatically provided when a record is written and are not communicated to the user when a file is read. Although they do not appear in the descriptions of logical records, control bytes do appear in the buffer areas of main storage. The compiler automatically allocates input and output buffers that are large enough to contain the required control bytes.

When variable-length records are written on unit record devices, control bytes are neither printed nor punched. They do appear, however, on other external storage devices. V-mode records moved from an input buffer to a working storage area will be moved without the control bytes.

Note: When a READ INTO statement is used for a V-mode file, the size of the current record for that file is used in the MOVE statement. All other rules of the MOVE statement apply. For considerations when using OCCURS DEPENDING ON, see the section "Programming Techniques."

Example 1:

Consider the following physical sequential file consisting of unblocked V-mode records:

```
FD  VARIABLE-FILE-1
    RECORDING MODE IS V
    BLOCK CONTAINS 35 TO 80 CHARACTERS
    RECORD CONTAINS 27 TO 72 CHARACTERS
    DATA RECORD IS VARIABLE-RECORD-1
    LABEL RECORDS ARE STANDARD.

01  VARIABLE-RECORD-1.
    LOGICAL RECORD
    05 FIELD-A      PIC X(20).
    05 FIELD-B      PIC 99.
    05 FIELD-C  OCCURS 1 TO 10 TIMES
       DEPENDING ON
       FIELD-B      PIC 9(5).
```

The LABEL RECORDS clause is always required. The DATA RECORD(S) clause is never required. If the RECORDING MODE clause is omitted, the compiler determines the mode as V since the record associated with VARIABLE-FILE-1 varies in length depending on the contents of FIELD-B. The RECORD CONTAINS clause is never required. The compiler determines record sizes from the record description entries. The BLOCK CONTAINS clause is also not required, since the compiler assumes unblocked records if the clause is omitted. Note: Record length calculations are affected by the following:

- When the BLOCK CONTAINS clause with the RECORDS option is used, the compiler adds four bytes to the logical record length and four more bytes to the block length.
- When the BLOCK CONTAINS clause with the CHARACTERS option is used, the user must include each cc + CC in the length calculation. In the definition of VARIABLE-FILE-1, the BLOCK CONTAINS clause specifies eight more bytes than does the RECORD CONTAINS clause. Four of these bytes are the logical record control bytes and the other four are the block control bytes.

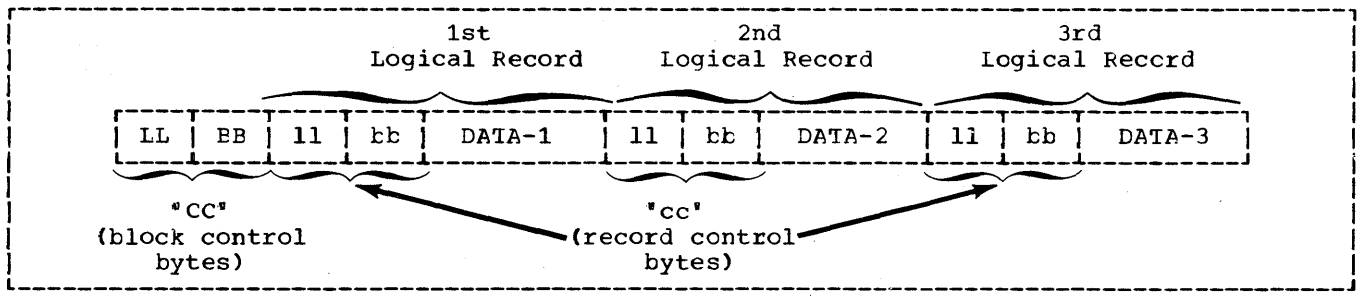


Figure 63. Blocked V-Mode Records

In Example 1, assume that FIELD-B contains the value 02 for the first record of a file and FIELD-B contains the value 03 for the second record of the file. The first two records will appear on an external storage device and in buffer areas of main storage as shown in Figure 64.

If the file described in Example 1 had a blocking factor of 2, the first two records would appear on an external storage medium as shown in Figure 65.

Example 2:

If VARIABLE-FILE-2 is blocked, with space allocated for three records of maximum size per block, the following FD entry could be used when the file is created:

```
FD VARIABLE-FILE-2
     RECORDING MODE IS V
     BLOCK CONTAINS 3 RECORDS
     RECORD CONTAINS 20 TO 100 CHARACTERS
     DATA RECORDS ARE VARIABLE-RECORD-1,
       VARIABLE-RECORD-2
     LABEL RECORDS ARE STANDARD.

01   VARIABLE-RECORD-1.
     05 FIELD-A   PIC X(20).
     05 FIELD-B   PIC X(80).

01   VARIABLE-RECORD-2.
     05 FIELD-X   PIC X(20).
```

As mentioned previously, the RECORDING MODE, RECORD CONTAINS, and DATA RECORDS clauses are unnecessary. By specifying that each block contains three records, the programmer allows the compiler to provide space for three records of maximum size plus additional space for the required control bytes. Hence, 316 character positions are reserved by the compiler for each output buffer. If this size is other than that required, the BLOCK CONTAINS clause with the CHARACTERS option should be specified. If the block size is to be specified at execution time by use of the BLKSIZE subparameter on an associated DD card, BLOCK CONTAINS 0 CHARACTERS must be specified.

Note: Blocked variable-length records are permitted only when the file processing technique is physical sequential.

In Example 2, assume that the first six records written are five 100-character records followed by one 20-character record. The first two blocks of VARIABLE-FILE-2 will appear on the external storage device as shown in Figure 66.

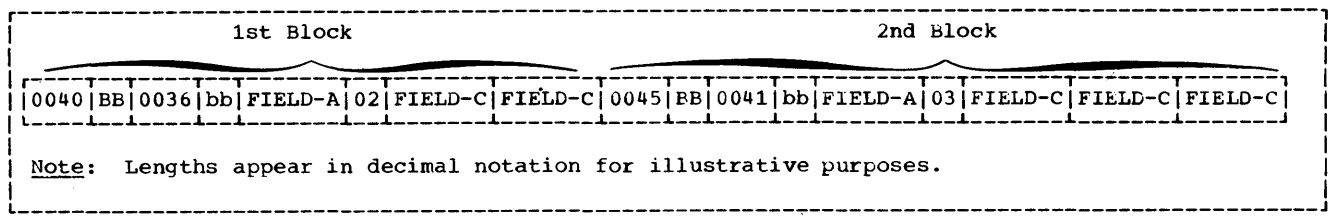


Figure 64. Fields in Unblocked V-Mode Records

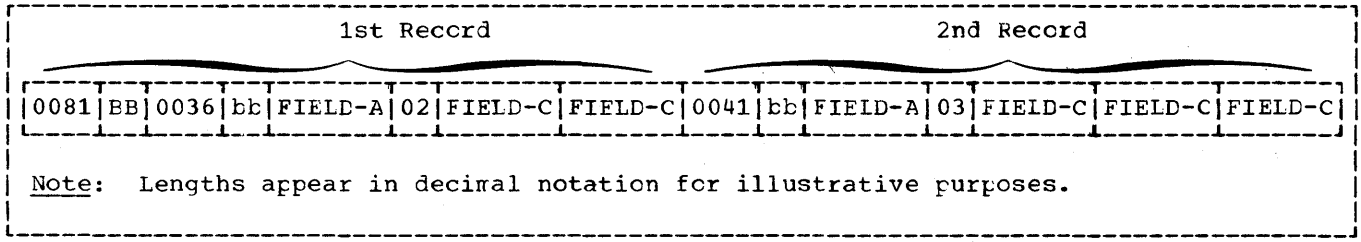


Figure 65. Fields in Blocked V-Mode Records

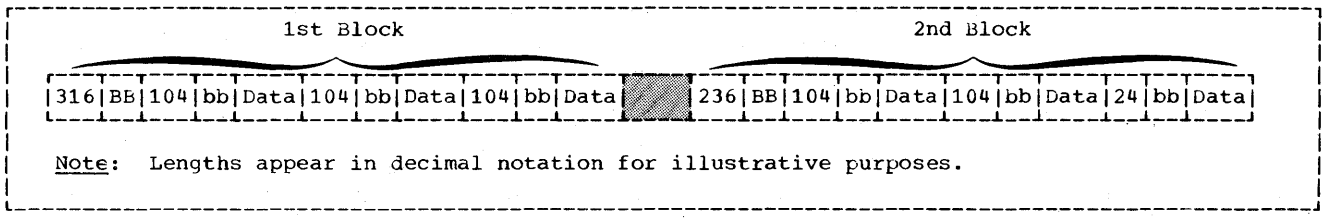


Figure 66. First Two Blocks of VARIABLE-FILE-2

The buffer for the second block is truncated after the sixth WRITE statement is executed since there is not enough space left for a maximum size record. Hence, even if the seventh WRITE to VARIABLE-FILE-2 is a 20-character record, it will appear as the first record in the third block. This condition can be eliminated by using the APPLY WRITE-ONLY clause when creating files of variable-length blocked records.

Note: Illustrations of unblocked Format V records do not take into account either the key field required when direct organization is used or ASCII block prefixes. See the description of format D records under "Processing ASCII Files".

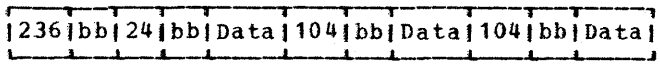
APPLY WRITE-ONLY Clause

The APPLY WRITE-ONLY clause is used to make optimum use of buffer space when creating a physical sequential file with blocked V-mode records.

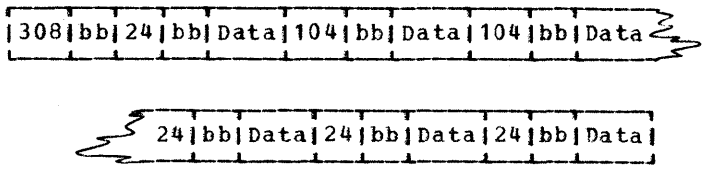
Suppose VARIABLE-FILE-2 is being created with the following file description entry:

```
FD   VARIABLE-FILE-2
     RECORDING MODE IS V
     BLOCK CONTAINS 316 CHARACTERS
     DATA RECORDS ARE VARIABLE-RECORD-1,
       VARIABLE-RECORD-2
     LABEL RECORDS ARE STANDARD.

01   VARIABLE-RECORD-1.
     05 FIELD-A   PIC X(20).
     05 FIELD-B   PIC X(80).

01   VARIABLE-RECORD-2.
     05 FIELD-X   PIC X(20).
```

The first three WRITE statements to the file create one 20-character record followed by two 100-character records. Without the APPLY WRITE-ONLY clause, the buffer is truncated after the third WRITE statement is executed since the maximum size record no longer fits. The block is written as shown below:



Using the APPLY WRITE-ONLY clause causes a buffer to be truncated only when the next record does not fit in the buffer. That is, if the next three WRITE statements to the file specify VARIABLE-RECORD-2, the block is created containing six logical records, as shown below:



Note: When using the APPLY WRITE-ONLY clause, records must not be constructed in buffer areas. An intermediate work area must be used with a WRITE FROM statement.

SPANNED (FORMAT S) RECORDS

A spanned record is a logical record that may be contained in one or more physical blocks. Format S records may be specified for direct (BDAM, BSAM) files and for physical sequential (QSAM) files assigned to magnetic tape or to mass storage devices.

When creating files with S-mode records, if a record is larger than the remaining space in a block, a segment of the record is written to fill the block. The remainder of the record is stored in the next block or blocks, as required.

When retrieving a file with S-mode records, only complete records are made available to the user.

Spanned records are preceded by fields containing control information. Figure 67 illustrates the control fields.

BDF (Block Descriptor Field):

LL -- represents two bytes designating the length of the physical block (including the block descriptor field itself).

BB -- represents two bytes reserved for system use.

SDF (Segment Descriptor Field):

ll -- represents two bytes designating the length of the record segment (including the segment descriptor field itself).

bb -- represents two bytes reserved for system use.

Note: There is only one block descriptor field at the beginning of each physical block. There is, however, one segment descriptor field for each record segment within the block.

Each segment of a record in a block, even if it is the entire record, is preceded by a segment descriptor field. The segment descriptor field also indicates whether the segment is the first, the last, or an intermediate segment. Each block includes a block descriptor field. These fields are not described in the Data Division; provision is automatically made for them. These fields are not available to the user.

A spanned blocked file may be described as a file composed of physical blocks of fixed length established by the programmer. The logical records may be either fixed or variable in length and that size may be smaller, equal to, or larger than the physical block size. There are no required relationships between logical records and physical block sizes. Records of a spanned file may only be blocked when organization is sequential (QSAM).

A spanned unblocked file may be described as a file composed of physical blocks each containing one logical record or one segment of a logical record. The logical records may be either fixed or variable in length. When the physical block contains one logical record, the length of the block is determined by the logical record size. When a logical record has to be segmented, the system always writes the largest physical block possible. The system segments the logical record when the entire logical record cannot fit on the track.

Figure 68 is an illustration of blocked spanned records of SFILE. SFILE is described in the Data Division with the following file description entry:

```
FD  SFILE
    RECORD CONTAINS 250 CHARACTERS
    BLOCK CONTAINS 100 CHARACTERS
    .
    .
    .
```

Figure 68 also illustrates the concept of record segments. Note that the third block contains the last 50 bytes of REC-1 and the first 50 bytes of REC-2. Such portions of logical records are called record segments. It is therefore correct to say that the third block contains the last segment of REC-1 and the first segment of REC-2. The first block contains the first segment of REC-1 and the second block contains an intermediate segment of REC-1.

S-MODE CAPABILITIES

Formatting a file in the S-mode allows the user to make the most efficient use of external storage while organizing data files with logical record lengths most suited to his needs.

1. Physical record lengths can be designated in such a manner as to make the most efficient use of track capacities on mass storage devices.

2. The user is not required to adjust logical record lengths to maximum physical record lengths and their device-dependent variants when designing his data files.

3. The user has greater flexibility in transferring logical records across DASD types.

Spanned record processing will not be supported on unit record devices.

SEQUENTIAL S-MODE FILES (QSAM) FOR TAPE OR MASS STORAGE DEVICES

When the spanned format is used for QSAM files, the logical records may be either fixed or variable in length and are completely independent of physical record length. A logical record may span physical records. A physical record may contain one or more logical records and/or segments of logical records.

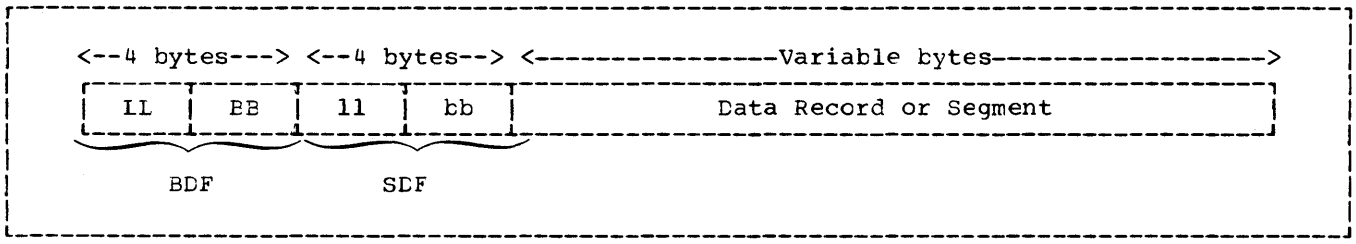


Figure 67. Control Fields of an S-Mode Record

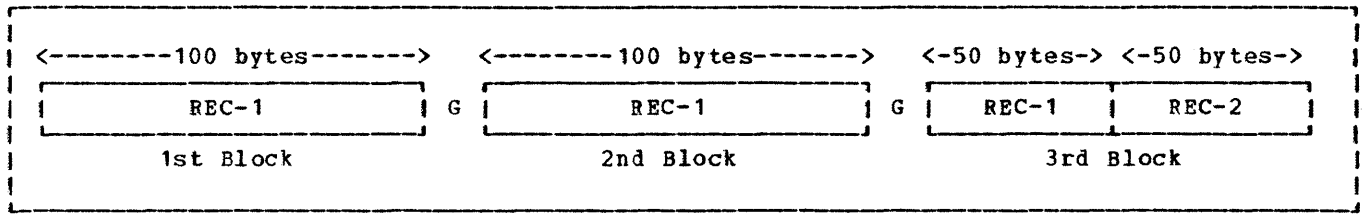


Figure 68. One Logical Record Spanning Physical Blocks

Source Language Considerations

The user specifies S-mode by describing the file with the following clauses in the file description (FD) entry of his COBOL program:

- BLOCK CONTAINS integer-2 CHARACTERS
- RECORD CONTAINS [integer-1 TO] integer-2 CHARACTERS
- RECORDING MODE IS S

The size of the physical record must be specified using the BLOCK CONTAINS clause with the CHARACTERS option. Any block size may be specified. Block size is independent of logical record size.

The size of the logical record may be specified by the RECORD CONTAINS clause. If this clause is omitted, the compiler will determine the maximum record size from the record descriptions under the FD.

Format S may be specified by the RECORDING MODE IS S clause. If this clause is omitted, the compiler will set the recording mode to S if the BLOCK CONTAINS inteter-2 CHARACTERS clause was specified and either of the following conditions exist:

- Integer-2 is less than the largest fixed-length level-01 FD entry.
- Integer-2 is less than the maximum length of a variable level-01 FD entry (i.e., an entry containing one or more OCCURS clauses with the DEPENDING ON option).

Except for the APPLY WRITE-ONLY, APPLY RECORD-OVERFLOW, WRITE BEFORE ADVANCING, WRITE AFTER ADVANCING, or WRITE AFTER POSITIONING clauses, all the options for a variable file apply to a spanned file.

Processing Sequential S-Mode Files (QSAM)

Suppose a file has the following file description entry:

```
FD  SPAN-FILE
    BLOCK CONTAINS 100 CHARACTERS
    LABEL RECORDS ARE STANDARD
    DATA RECORD IS DATAREC.

01  DATAREC.
    05  FIELD-A PIC X (100).
    05  FIELD-B PIC X (50).
```

Figure 69 illustrates the first four blocks of SPAN-FILE as they would appear on external storage devices (i.e., tape or mass storage) or in buffer areas of main storage.

Notes:

1. The RECORDING MODE clause is not specified. The compiler determines the recording mode to be S since the

block size is less than the record size.

2. The length of each physical block is 100 bytes, as specified in the BLOCK CONTAINS clause. All required control fields, as well as data, must be contained within these 100 bytes.

3. No provision is made for the control fields within the level-01 entry DATAREC.

The preceding discussion dealt with S-mode records which were larger than the physical blocks that contained them. It is also possible to have S-mode records which are equal to or smaller than the physical blocks that contain them. In such cases, the RECORDING MODE clause must specify S (if so desired) since the compiler cannot determine this by comparing block size and record size.

One advangage of S-mode records over V-mode records is illustrated by a file with the following characteristics:

1. RECORD CONTAINS 50 TO 150 CHARACTERS

2. BLOCK CONTAINS 350 CHARACTERS

3. The first five records written are 150, 150, 150, 100, and 150 characters in length.

For V-mode records, buffers are truncated if the next logical record is too large to be completely contained in the block (Figure 70). This results in more physical blocks and more inter-record gaps on the external storage device.

Note: For V-mode records, buffer truncation occurs:

1. When the maximum level-01 record is too large.

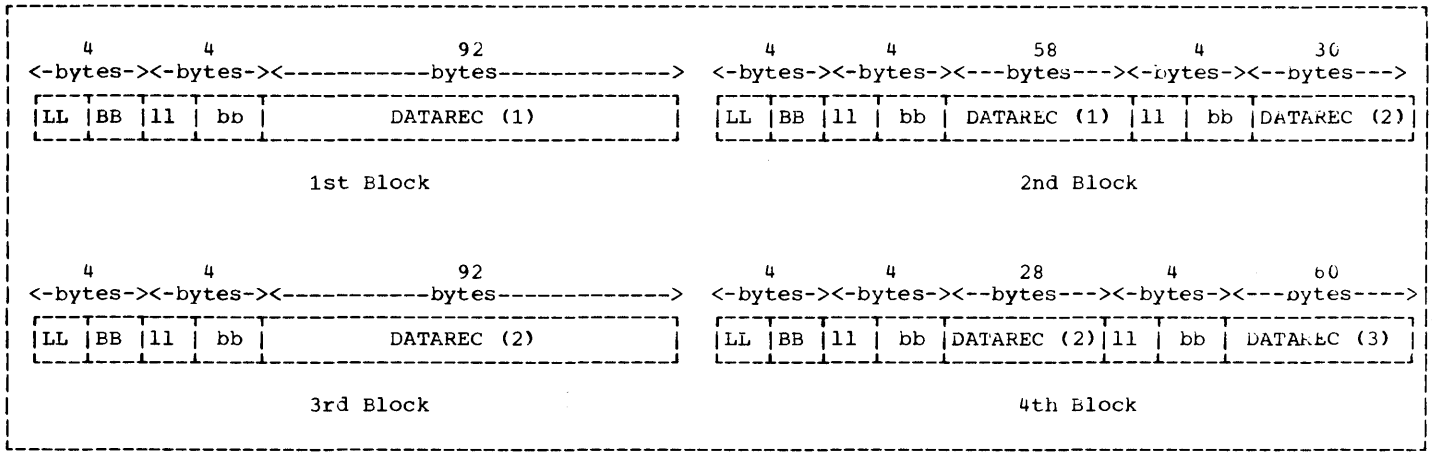


Figure 69. First Four Blocks of SPAN-FILE

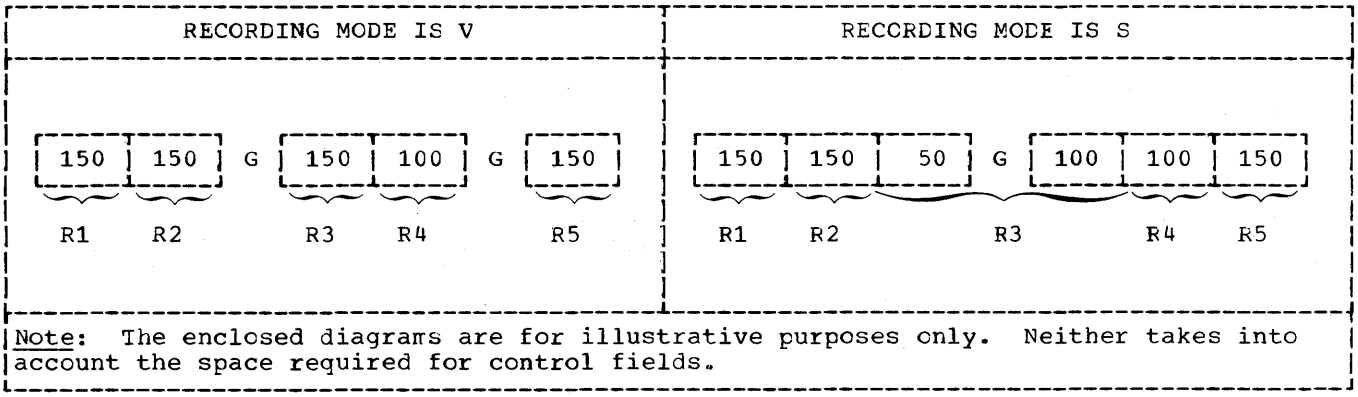


Figure 70. Advantage of S-Mode Records Over V-Mode Records

2. If APPLY WRITE-ONLY or SAME RECORD AREA is specified and the actual logical record is too large to fit into the remainder of the buffer.

For S-mode records, all blocks are 350 bytes in length and records that are too large to fit entirely into a block will be segmented. This results in more efficient use of external storage devices since the number of inter-record gaps are minimized (Figure 70).

A second advangage of S-mode processing over that of V-mode is that the user is no longer limited to a record length that does not exceed the track of the mass storage device selected. Records may span tracks, cylinders, extents, and volumes.

QSAM spanned records differ from other QSAM record formats because of an allocation of an area of main storage known as the "Logical Record Area." If logical records span physical blocks, COBOL will use this Logical Record Area to assemble complete logical records. If logical records do not span blocks (i.e., they are contained within a single physical block) the Logical Record Area is not used. Regardless, only complete logical records are made available to the user. Both READ and WRITE statements should be thought of as manipulating complete logical records not record segments.

The allocation of a Logical Record Area may be a disadvantage to the COBOL user. Additional main storage, consisting of 36 bytes + the maximum record length, will always be required. The Logical Record Area is discussed in detail in "Finding Data Records in an Abnormal Termination Dump."

DIRECTLY ORGANIZED S-MODE FILES (BDAM AND BSAM)

When S-mode is used for directly organized files, only unblocked records are permitted. Logical records may be either fixed or variable in length. A logical record will span physical records if, and only if, it spans tracks. A physical record will contain only one logical record or a segment of a logical record. A track may contain a segment of a logical record, or segments of two logical records and/or whole logical records. Records may span tracks, cylinders, and extents, but not volumes.

Source Language Considerations

The user specifies S-mode by describing the file with the following clauses in the file description (FD) entry of his COBOL program:

- BLOCK CONTAINS integer-2 CHARACTERS
- RECORD CONTAINS [integer-1 TO] integer-2 CHARACTERS
- RECORDING MODE IS S

The size of a logical record may be specified by the RECORD CONTAINS clause. If this clause is omitted, the compiler

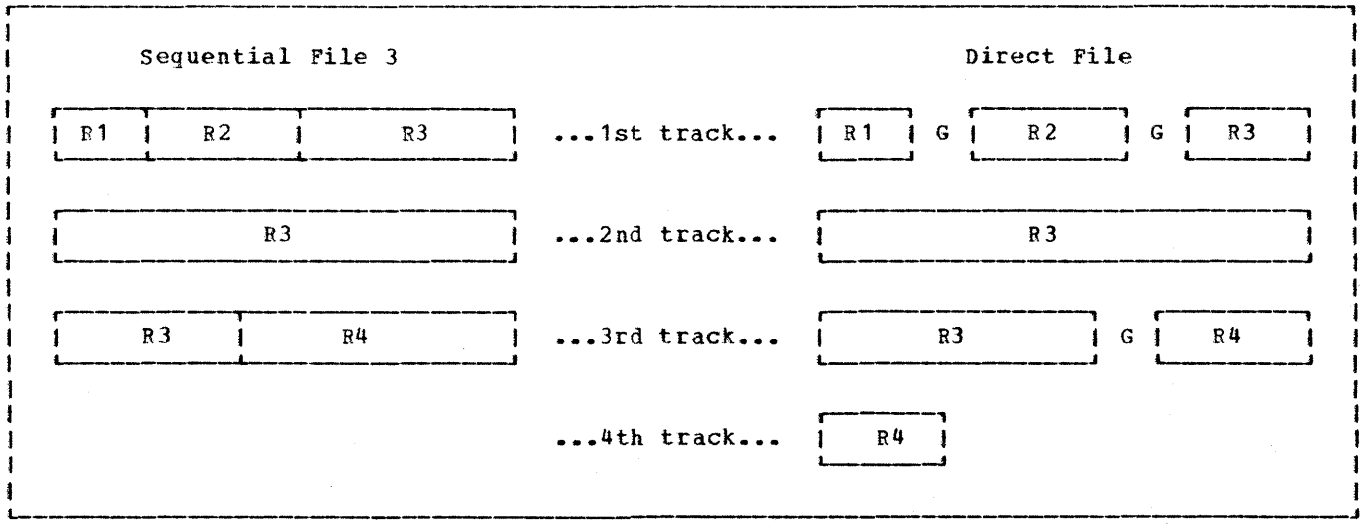


Figure 71. Direct and Sequential Spanned Files on a Mass Storage Device

will determine the maximum record size from the record descriptions under the FD.

The spanned format may be specified by the RECORDING MODE IS S clause. If this clause is omitted, the compiler will set the recording mode to S if the BLOCK CONTAINS integer-2 CHARACTERS clause was specified and integer-2 is less than the greatest logical record size. This is the only use of the BLOCK CONTAINS clause. It is otherwise treated as comments.

The physical block size is determined by either:

1. The logical record length.

2. The track capacity of the device being used.

If, for example, the track capacity of a mass storage device is 7294 characters, any record smaller than 7294 characters may be written as a single physical block. If a logical record is greater than 7294 characters, the record is segmented. The first segment may be contained in a physical block of up to 7294 bytes, and the remaining segments must be contained in succeeding blocks. In other words, a logical record will span physical blocks if, and only if, it spans tracks.

Figure 71 illustrates four variable-length records (R1, R2, R3, and R4) as they would appear in direct and sequential files on a mass storage device. In both cases, control fields have been omitted for illustrative purposes. For both files, assume:

1. BLOCK CONTAINS 7294 CHARACTERS (track capacity = 7294)

2. RECORD CONTAINS 500 TO 8000 CHARACTERS

In the sequential file, each physical block is 7294 bytes in length and is completely filled with logical records. The file consists of three physical blocks, occupies three tracks, and contains no inter-record gaps.

In the direct file, the physical blocks vary in length. Each block contains only one logical record or one record segment. Logical record R3 spans physical blocks only because it spans tracks. The file consists of seven physical blocks, occupies more than three tracks, and contains three inter-record gaps.

Processing Directly Organized S-Mode Files (BDAM and BSAM)

When processing directly organized files, there are two advantages spanned format has over the other record formats:

1. Logical record lengths may exceed the length restriction of the track capacity of the mass storage device. If, for example, the track capacity of a mass storage device is 2000 bytes, this does not represent the maximum length of the logical record that can be specified (even when the device does not have a Track Overflow feature).

   Note: Even when the spanned format is used, the COBOL restriction on the length of logical records must be adhered to (i.e., a maximum length of 32,767 characters).

2. S-mode records give the user the same facility as the Track Overflow feature. If neither RECORDING MODE IS S nor APPLY RECORD-OVERFLOW is specified, only complete logical records can be written on any single track. This means that when a track has only 900 unoccupied bytes and a record of 1000 bytes is to be added, it will be written on the next available track. This is inefficient, since a 900 byte segment could be added to the current track by means of either APPLY RECORD-OVERFLOW or RECORDING MODE IS S.

   Note: If a choice exists between Track Overflow and S-mode records, neither has any particular advantage over the other with regard to the efficient use of storage space.

The disadvantage of BSAM and BDAM spanned records is similar to that mentioned for QSAM. A segment work area is always allocated which occupies additional main storage.

Like QSAM, the processing of BSAM and BDAM spanned records relies on an interaction between buffers, segment work areas, and Logical Record Areas. For QSAM, input-output buffers are used as the segment work area and complete logical records are assembled in a Logical Record Area before being made available to the user if the record is segmented. If the record is not segmented, the logical recordis made available to the user within the buffer unless the SAME AREA clause is specified. For BSAM and BDAM, input-output buffers are used as a Logical Record Area and a separate segment work area must be

allocated. Segment work areas and Logical Record Areas are described fully in "Finding Data Records in an Abnormal Termination Dump."

OCCURS CLAUSE WITH THE DEPENDING ON OPTION

If a record description contains an OCCURS clause with the DEPENDING ON option,the record length is variable. This is true for records described in an FD as well as in the Working-Storage section. The previous sections discussed four different record formats. Three of them, V-mode, U-mode, and S-mode, may contain one or more OCCURS clauses with the DEPENDING ON option.

The following section discusses some factors that affect the manipulation of records containing OCCURS clauses with the DEPENDING ON option. The text indicates whether the factors apply to the File (FD) or Working-Storage sections, or both.

The compiler calculates the length of records containing an OCCURS clause with the DEPENDING ON option at two different times, as follows (the first applies to FD entries only, the second to both FD and Working-Storage entries):

1. When a file is read and the object of a DEPENDING ON option is within the record.

2. When the object of the DEPENDING ON option is changed as a result of a move to it or to a group that contains it. (The length is not calculated when a move is done to an item which redefines or renames it.)

Consider the following example:

```
WORKING-STORAGE SECTION.

77  CONTROL-1     PIC 99.
77  WORKAREA-1    PIC 9(6)V99.
        .
        .
        .
01  SALARY-HISTORY.
    05  SALARY OCCURS 1 TO 10 TIMES
        DEPENDING
        ON CONTROL-1   PIC 9(6)V99.
```

The Procedure Division statement MOVE 5 TO CONTROL-1 will cause a recalculation of the length of SALARY-HISTORY. MOVE SALARY (5) TO WORKAREA-1 will not cause the length to be recalculated.

The compiler permits the occurrence of more than one level-01 record, containing the OCCURS clause with the DEPENDING ON option, in the same FD entry (Figure 72). If the BLOCK CONTAINS clause is omitted, the buffer size is calculated from the longest level-01 record description entry. In Figure 72, the buffer size is determined by the description of RECORD-1 (RECORD-1 need not be the first record description under the FD).

During the execution of a READ statement, the length of each level-01 record description entry in the FD will be calculated (Figure 72). The length of the variable portions of each record will be the product of the numeric value contained in the object of the DEPENDING ON option and the length of the subject of the OCCURS clause. In Figure 72, the length of FIELD-1 is calculated by multiplying the contents of CONTROL-1 by the length of FIELD-1; the length of FIELD-2, by the product of the contents of CONTROL-2 and the length of FIELD-2; the length of FIELD-3 by the contents of CONTROL-3 and the length of FIELD-3.

Since the execution of a READ statement makes available only one record type (i.e., RECORD-1 type, RECORD-2 type, or RECORD-3 type), two of the three record descriptions in Figure 72 will be inappropriate. In such cases, if the contents of the object of the DEPENDING ON option does not conform to its picture, the length of the corresponding record will not be calculated. For the contents of an item to conform to its picture:

- An item described as USAGE DISPLAY must contain decimal data.
- An item described as USAGE COMPUTATIONAL-3 must contain internal decimal data.
- An item described as USAGE COMPUTATIONAL must contain binary data.
- An item described as USAGE DISPLAY or USAGE COMPUTATIONAL-3 must conform to the rules for the numeric class test condition:
  - If the PICTURE for the item does not contain an operational sign, the item being tested conforms only if the contents are unsigned numeric.
  - If the PICTURE clause for the item does contain an operational sign, the identifier being tested conforms only if the item is elementary signed numeric.

  In the EBCDIC collating sequence for signed items, valid embedded operational signs are hexadecimal C, D, and E; for items described with the SIGN IS SEPARATE clause, valid operational signs are + (hexadecimal 4E) and - (hexadecimal 60).

```
FD  INPUT-FILE
    .
    .
    .
    DATA RECORDS ARE RECORD-1 RECORD-2 RECORD-3.

01  RECORD-1.
    02  CONTROL-1                PIC 99.
    02  FIELD-1 OCCURS 1 TO 10 TIMES DEPENDING ON CONTROL-1  PIC 9(5).

01  RECORD-2.
    02  CONTROL-2                PIC 99.
    02  FIELD-2 OCCURS 1 TO 5 TIMES DEPENDING ON CONTROL-2   PIC 9(4).

01  RECORD-3.
    02  FILLER                   PIC XX.
    02  CONTROL-3                PIC 99.
    02  FIELD-3 OCCURS 1 TO 10 TIMES DEPENDING ON CONTROL-3  PIC X(4).
```

Figure 72. Calculating Record Lengths When Using the OCCURS Clause with the DEPENDING ON Option

The following example illustrates the length calculations made by the system when a READ statement is executed:

```
FD
    .
    .
    .
01  RECORD-1.
    05  A  PIC  99.
    05  B  PIC  99.
    05  C  PIC  99  OCCURS 1 TO 5 TIMES
        DEPENDING ON A.

01  RECORD-2.
    05  D  PIC  XX.
    05  E  PIC  99.
    05  F  PIC  99.
    05  G  PIC  99  OCCURS 1 TO 5 TIMES
        DEPENDING ON F.


WORKING-STORAGE SECTION.
    .
    .
    .
01  TABLE-3.
    05  H OCCURS 1 TO 10 TIMES DEPENDING
        ON B.


01  TABLE-4.
    05  I OCCURS 1 TO 10 TIMES DEPENDING
        ON E.
```

When a record is read, lengths are determined as follows:

1. The length of RECORD-1 is calculated using the contents of field A.
2. The length of RECORD-2 is calculated using the contents of field F.
3. The length of TABLE-3 is calculated using the contents of field B.
4. The length of TABLE-4 is calculated using the contents of field E.

The user should be aware of several additional factors that affect the successful manipulation of variable-length records. The following example illustrates a group item (i.e., REC-1) whose subordinate items contain an OCCURS clause with the DEPENDING ON option and the object of that DEPENDING ON option.

```
WORKING-STORAGE SECTION.
01  REC-1.
    05  FIELD-1              PIC S9.
    05  FIELD-2 OCCURS 1 TO 5 TIMES
        DEPENDING ON FIELD-1  PIC X(5).

01  REC-2.
    05  REC-2-DATA           PIC X(50).
```

The results of executing a MOVE to the group item REC-1 will be affected by the following:

- The length of REC-1 may have been calculated at some time prior to the execution of this MOVE statement. The user should be sure that the current length of REC-1 is the desired one.
- The length of REC-1 may never have been calculated at all. In this case, the result of the move will be unpredictable.
- After the move, since the contents of FIELD-1 have been changed, an attempt

will be made to recalculate the length of REC-1. This recalculation, however, will be made only if the new contents of FIELD-1 conform to its picture. In other words, if FIELD-1 does not contain an external decimal item, the length of REC-1 will not be recalculated.

Note: According to the COBOL description, FIELD-2 can occur a maximum of five times. If, however, FIELD-1 contains an external decimal item whose value exceeds five, the length of REC-1 will still be calculated. One possible consequence of this invalid calculation will be encountered if the user attempts to initialize REC-1 by moving zeros or spaces to it. This initialization would inadvertently delete part of the adjacent data stored in REC-2.

The following example applies to updating a record containing an OCCURS clause with the DEPENDING ON option and at least one other subsequent entry. In this case, the subsequent entry is another OCCURS clause with the DEPENDING ON option.

```
WORKING-STORAGE SECTION.
01  VARIABLE-REC.
    05  FIELD-A      PIC X(10).
    05  CONTROL-1    PIC S99.
    05  CONTROL-2    PIC S99.
    05  VARY-FIELD-1 OCCURS 1 TO 10 TIMES
        DEPENDING ON CONTROL-1 PIC X(5).
    05  GROUP-ITEM-1.
        10  VARY-FIELD-2 OCCURS 1 TO 10
            TIMES DEPENDING ON CONTROL-2
            PIC X(9).
01  STORE-VARY-FIELD-2.
    03  GROUP-ITEM-2.
        05  VARY-FLD-2 OCCURS 1 TO 10
            TIMES DEPENDING ON CONTROL-2
            PIC X(9).
```

Assume that CONTROL-1 contains the value 5 and VARY-FIELD-1 contains 5 entries.

In order to add a sixth field to VARY-FIELD-1, the following steps are required:

```
MOVE GROUP-ITEM-1 TO GROUP-ITEM-2.
ADD 1 TO CONTROL-1.
MOVE 'additional field' TO
  VARY-FIELD-1 (CONTROL-1).
MOVE GROUP-ITEM-2 TO GROUP-ITEM-1.
```

Note: When an FD contains multiple 01-level entries, the following restriction applies: The object of an OCCURS DEPENDING ON clause must be in the fixed portion of all the records within that FD (not just in the fixed portion of the record it is described in). Stated another way, the minimum size of each 01-level record must exceed the highest position of the object of an OCCURS DEPENDING ON clause appearing in any 01-level record in that FD. Thus, the following coding would be wrong, and could cause an abend:

```
FD
   01 RECORD-1.
      02 FIELDA PIC X(200).
      02 FIELDB PIC 99.
      02 FIELDC PIC XX OCCURS 1 TO 25 TIMES
                      DEPENDING ON FIELDB.
   01 RECORD-2.
      02 FIELDD PIC X(100).
```

For a discussion of the use of the OCCURS DEPENDING ON clause in a sort program, see "Sorting Variable-Length Records."

VSAM is a high-performance access method of OS/VS for use with direct-access storage. VSAM provides high-speed retrieval and storage of data, ease of use (including simplified JCL statements), data protection against unauthorized access, central control of data management functions, cross-system compatibility, and device independence. VSAM data sets can be processed by the COBOL programmer after they have been defined through use of the VSAM multifunction utility program known as Access Method Services. This utility (described in the publications OS/VS1 Access Method Services and OS/VS2 Access Method Services) will describe data sets, load records into them if desired, and perform numerous other tasks--such as converting existing ISAM and SAM data sets to VSAM format.

TYPES OF VSAM DATA SETS

COBOL allows access to the three major types of VSAM data sets: entry-sequenced (ESDS), key-sequenced (KSDS) and relative record (RRDS). The primary difference between the three is the order in which their data records are stored and retrieved.

Records are stored in an entry-sequence data set without respect to the contents of the records. The sequence is determined by the order in which the records are presented for inclusion in the data set: that is, their entry sequence. New records are stored at the end of the data set. Records can be retrieved sequentially only, that is, in the order they were stored in the data set.

Records are stored in a key sequenced data set in key sequence: that is, in the order defined by the collating sequence of the primary key field in each record. Each record has a unique value in the primary key field, such as employee number or invoice number. VSAM uses the key associated with each record to insert or retrieve a record in the data set. The order can be random or sequential.

Records are stored in a relative record data set in relative record number sequence. The data set may be described as a string of fixed-length slots, each of which is identified by a number which gives its position relative to the first such slot. new records are inserted either sequentially in the next available slot, where they assume that relative record number, or according to a relative record number that the programmer specifies. Records may be retrieved either sequentially or by specific relative record number.

ENTRY-SEQUENCED DATA SETS

An entry-sequenced data set has no index associated with it. Records are added at the end of the data set. The data set can be accessed sequentially only; the access is similar to QSAM except that tape storage or unit record devices cannot be used with VSAM.

KEY-SEQUENCED DATA SETS

The index and distributed free space are the most distinctive features of a key-sequenced data set.

The index relates key values to the data records in the data set. A record's key field and position are the same for every record in the data set; its value cannot be altered. VSAM uses the index to locate a record for retrieval and to locate the collating position for insertion.

Multiple indexing is also available. This means that a record may have both a primary key and up to 254 alternate, or secondary, keys. An alternate key may be any field in the data record that has a fixed length and position. (In spanned records, the alternate key must be in the first control interval.) Alternate keys serve the same function in accessing data, but allow the user additional flexibility in his processing. In contrast to the primary key, values of these alternate keys need not be unique.

When a key-sequenced data set is defined (through Access Method Services), free space can be distributed in two ways: by leaving some space at the end of all the

used control intervals[1] and by leaving some control intervals completely empty. Space becomes available within a control interval when a record is shortened or deleted. This space can then be used by VSAM when a record is lengthened in place or inserted into the control interval.

RELATIVE RECORD DATA SETS

A relative record data set has no index. In its string of fixed-length slots, only the relative record number--a number from 1 to n, where n is the maximum number of records that can be stored in the data set--identifies the record. Each record occupies one slot, and is stored and retrieved according to the relative record number of that slot. The record's contents and entry sequence are unimportant.

Records in a relative record data set are grouped together in control intervals, just as they are in a entry-sequenced or key-sequenced data set. Each control interval contains the same number of slots. The size of each slot is the record length specified by the user when he defined the data set.

ACCESS METHOD SERVICES

Access Method Services is a utility program that must be used before any COBOL program can process VSAM files. A number of user-entered commands initiate the Access Method Services programs. There are two types of commands: functional and modal. The functional commands invoke the desired Access Method Services function, while the modal commands control the sequence of execution of the functional commands. For more information on modal commands as well as for complete details on the specification and uses of Access Method Services, see the publication OS/VS Access Method Services.

There are a number of important functional commands: DEFINE, ALTER, DELETE, LISTCAT, REPRO, PRINT, IMPORT, EXPORT, BLDINDEX, and VERIFY. The commands DEFINE, ALTER, DELETE, and LISTCAT are used to create, modify, remove and list entries in the VSAM catalog. The REPRO and PRINT commands reproduce data sets either as new data sets or as printed output. The IMPORT and EXPORT commands provide a way to transfer data sets from one system to another. The BLDINDEX command builds alternate indexes for VSAM KSDS data sets. The VERIFY command provides a data set recovery service for VSAM data sets by verifying that the end of the data set indicated in the catalog is the same as the real data set end.

---

[1] A control interval is the unit of information that VSAM transfers between virtual and auxiliary storage.

THE DEFINE COMMAND

The DEFINE command must be used to define:

1. Master catalog: catalog in which all VSAM data sets must be entered.
2. User catalog: optional catalog in which VSAM data sets may be entered.
3. Data space: space which is to be used by VSAM.
4. VSAM data set(s): data sets that are to be processed by VSAM.
5. Any alternate indexes and alternate paths.

In order to process a VSAM data set, the above must be done in the order indicated. Therefore it is necessary to fully understand the DEFINE command, its functions, and its specification.

Functions of the DEFINE Command

An object, in VSAM terminology, is:

- A VSAM catalog
- A VSAM data set (KSDS, RRDS, or ESDS)
- A VSAM data space
- A VSAM KSDS alternate index and path

Data sets must be introduced to the system and defined as entries in either the master or user catalog. Non-VSAM data sets may also be cataloged in a VSAM catalog. All VSAM and non-VSAM data sets are introduced to the system with the DEFINE command.

There are two steps in the creation of an object: (1) defining the object to the system and (2) generating its contents. The DEFINE command does not generate the contents of any object except for the entries in the master and user catalogs.

The process of defining a VSAM object includes allocation of storage space, initialization of catalog entry contents, and, in the case of defining catalogs, possible initialization of the object's storage space.

Specification of the DEFINE Command

The DEFINE command has the following format:

DEFINE object parameters

The definable objects are as follows:

- MASTERCATALOG: specifies that the VSAM master catalog is to be defined.
- USERCATALOG: specifies that a VSAM user catalog is to be defined.
- SPACE: specifies that a VSAM data space is to be defined.
- CLUSTER: specifies that a data set is to be defined.
- ALTERNATEINDEX: specifies that an alternate index for a KSDS is to be defined and have space allocated.
- PATH: establishes the relationship between an alternate index and its data set (base cluster).
- NONVSAM: specifies that a data set not having the VSAM data set organization is to be cataloged in a VSAM catalog.

For each data set there is an associated valid parameter list. See the publication OS/VS Access Method Services for the specification rules for DEFINE and its associated objects and parameters.

Defining a Master Catalog: DEFINE MASTERCATALOG

The DEFINE MASTERCATALOG command is used in VS1 to set up the master catalog. (In VS2, this command only creates a user catalog. The VS2 master catalog is created at system-generation time.) It is the first Access Method Services command ever used since without a master catalog other objects cannot be defined. Defining a master catalog automatically creates the data space necessary to contain the catalog. Entries for both the master catalog itself and the volume containing the data space automatically created are placed in the master catalog.

The following is an example of defining a VSAM master catalog:

```
//MYJOB1     JOB   .........
//STEP       EXEC  PGM=IDCAMS
//SYSPRINT   DD    SYSOUT=A
//VOL        DD    DISP=OLD,UNIT=2314,
                   VOL=SER=MYVOL

//SYSIN      DD    *
      DEFINE     MASTERCATALOG(NAME     +
                 (MASTCATL) FILE(VOL)   +
                 VOLUME(myvol) RECORDS  +
                 (300 10)    +
                 MASTERPW(111111)    +
                 UPDATEPW(222222)    +
                 READPW(333333))
/+
```

Note the following concerning the JCL used.

- PGM=IDCAMS is required to invoke Access Method Services.
- MYVOL is the volume serial number of the volume on which the master catalog is to reside. //VOL is the DD statement identifying that volume.

In this example the following parameters were used:

- NAME (objectname). This is a required parameter. The objectname is cataloged and is the name that must be used in all future references to the master catalog.
- FILE (dname). This is a required parameter identifying the JCL statement that specifies the device and volume which are to contain the master catalog. The associated JCL statement should specify DISP=OLD. For dname, the name of the JCL statement that specifies the device and volume to contain the master catalog is substituted.
- VOLUME (volser). This is a required parameter that specifies the volume to contain the master catalog.
- RECORDS (primary[secondary]). This is a required parameter that specifies the amount of space to be allocated in terms of the number of records the space is to hold. The capacity of the allocated space is device independent. The size of the primary extent is specified by primary. Once the primary extent is filled, the data space can expand to include a maximum of 15 secondary extents if the size of

secondary extents is specified through secondary.

For primary and secondary, the number of records the master catalog is to be able to hold is substituted. Note that every KSDS requires three catalog entries: one each for the cluster, data component, and index component. Every ESDS requires two catalog entries: one for the cluster and one for the data component.

- MASTERPW (password). This optional parameter specifies a master level password for the master catalog being defined. The master level password is the highest level that can be specified; it allows all operations. If a master level password is not supplied but other levels are, the highest level supplied password becomes the password for all higher levels including the master level. The master catalog must be password protected if any VSAM clusters are to be protected.

- UPDATEPW (password). This optional parameter specifies an update level password. The password permits read and write operations against the master catalog. The master catalog must be password protected if any VSAM clusters are to be protected.

- READPW (password). This optional parameter specifies a read level password for the master catalog being defined. The read level password permits read operations against the master catalog. The master catalog must be password protected if any VSAM clusters are to be protected.

Defining a User Catalog: DEFINE USERCATALOG

The DEFINE USERCATALOG command is used to set up user catalogs. When a user catalog is defined, a data space to containthe catalog is automatically created. An entry for the volume containing the data space is placed in the user catalog being defined. Entries for the user catalog being defined are placed in the master catalog and in the user catalog itself.

The parameters that may be used with DEFINE USERCATALOG are the same as those described for DEFINE MASTERCATALOG with one exception. There is an additional parameter that may be used with DEFINE USERCATALOG as follows:

- CATALOG (mastercatname/password). This parameter specifies the name and password of the master catalog that contains the entry for the user catalog being defined. This parameter is required only when the master catalog is password protected.

  For mastercatname, the name of the master catalog is substituted. For password, the update or higher level password is substituted.

Defining a VSAM Data Space: DEFINE SPACE

The DEFINE SPACE command is used to define VSAM data spaces or to reserve volumes for future use by VSAM. A VSAM data space is space on a direct access volume that is owned and managed by VSAM. Clusters cannot be defined without the UNIQUE attribute unless a VSAM data space is defined to contain them. A VSAM data space may include several extents on the same volume but it cannot span volumes. The volume containing the data space is owned by the catalog containing the entry for the space. Data spaces on several volumes can be defined at one time.

The following is an example of defining a VSAM data space:

```
//MYJOB2     JOB   ......
//STEP       EXEC  PGM=IDCAMS
//SYSPRINT   DD    SYSOUT=A
//SPACDD     DD    DISP=OLD,UNIT=2314,
                   VOL=SER=MYVOL
//SYSIN      DD    *
       DEFINE      SPACE(VOLUME(MYVOL) FILE+
                   (SPACDD)    +
                   CYLINDERS(30 5)  CATALOG+
                   (MASTCATL/222222)
 *
```

In this example the following parameters were used:

- VOLUMES(volser[...]). This required parameter specifies the volumes to contain the data spaces. If more than one volume is specified, each volume will contain a data space of the same size.

- FILE(dname). This required parameter identifies the JCL statement that specifies the devices and volumes to be used for space allocation. [For dname, substitute the name of the JCL statement that specifies the devices and volumes to be used for space allocation.] All volumes must be of the same device type.

- CYLINDERS(primary[secondary]). This parameter specifies the amount of space to be allocated in terms of cylinders. The capacity of the allocated space is device dependent. Either this parameter or the TRACKS or RECORDS parameter must be specified. The size of the primary extent is specified by primary. Once the primary extent is filled, the data space can expand to include a maximum of 15 secondary extents if the size of secondary extents is specified through secondary. For primary and secondary, the number of cylinders to be allocated to the data space is substituted.

- CATALOG(catname[/password][dname]). This parameter specifies the name and password of the catalog in which the data space is to be defined. [For catname, substitute the name of the catalog that is to contain the entry for the data space.] This parameter is specified if the desired catalog is not the default catalog (see Defaults below). If the catalog is password protected, the password must be specified.

  If the desired catalog is neither the master catalog nor a catalog identified by a JOBCAT or STEPCAT DD statement, the name of the DD statement identifying the catalog must be specified. For dname, the name of the DD statement that identifies the desired catalog is substituted.

  Defaults: The data space is defined in the catalog identified as STEPCAT. If STEPCAT is not provided, the data space is defined in the catalog identified as JOBCAT. If JOBCAT is not provided, the data space is defined in the master catalog.

Defining a KSDS

Figure 73 is an example of defining an indexed VSAM data set. DEFINE CLUSTER is used to define the attributes of all VSAM data sets and to catalog the data sets in a VSAM catalog. This command does not put any records into the VSAM data set. COBOL permits reference only to a KSDS cluster; in other words, the KSDS's data and index components cannot be defined separately.

The DEFINE CLUSTER command establishes the primary keys for the records. If only primary keys are to be used, then only this DEFINE CLUSTER command is needed. If alternate keys are also to be used (as in this example), they are established with the DEFINE ALTERNATEINDEX and DEFINE PATH commands. In addition (after the base cluster is filled with records), a follow-on job must be run to specify the BLDINDEX command (see the Access Method Services manual).

In this example the following parameters were used:

- NAME(objectname). This is a required parameter. It must be specified at the cluster level.

- FILE identifies the DD statement specifying the device and volume that are to contain the VSAM object being defined.

- VOLUME(volser [...]). This is a required parameter that specifies the volumes to contain the object. More than one volume can be specified; the volumes actually allocated are listed.

- RECORDS(primary [secondary]) specifies the amount of space to be allocated in terms of the number of records the space is to hold.

```
//MYJOB3      JOB      ......
//STEP        EXEC     PGM=IDCAMS
//SYSPRINT    DD       SYSOUT=A
//MYDD        DD       VOL=SER=MYVOL,UNIT=SYSDA,DISP=OLD
//SYSIN       DD       *
       DEFINE       CLUSTER(NAME(SAMPLE) FILE(MYDD) VOLUME(MYVOL)    +
                    RECORDS(500 50) RECORDSIZE(45 80)    +
                    FREESPACE(25 10) SUBALLOCATION INDEXED    +
                    KEYS(8 2) UPDATEPW(RDANDWRT) READPW(READONLY)    +
                    ATTEMPTS(0)),CATALOG(MASTCATL/222 222)
       DEFINE       ALTERNATEINDEX(NAME(ALTX) RELATE(SAMPLE)  +
                    FILE(MYDD1)  VOLUME(MYVOL)  RECORDS (500  50)  +
                    KEYS(6  15) UNIQUEKEY  UPDATEPW(RDANDWRT)  READPW(READONLY)+
                    ATTEMPTS(0))  CATALOG(MASTCATL/222  222)  UPGRADE
       DEFINE       PATH(NAME(PATHX)  PATHENTRY(ALTX)  UPDATE  +
                    CATALOG(MASTCATL/222  222)
 *
```

Figure 73. Defining a VSAM Indexed Data Set (KSDS) with Both Primary and Alternate Keys

- RECORDSIZE(average maximum) specifies the average and maximum lengths of the records in the data component of the cluster. This is a required parameter. The size specified can be from 1 to 32,761.

  The number substituted for average should be the number of bytes that is the average length of all logical records. The number substituted for maximum should be the maximum length of the largest logical record.

- FREESPACE(cipercent[capercent]) specifies the amount of space that is to be left free after any allocation and after any split of control intervals and control areas. The amount is specified as a percentage.

  For cipercent, the percentage of unused space desired in each control interval is specified. For capercent, the percentage of unused space desired in each control area is specified.

- SUBALLOCATION specifies that a portion of an already defined VSAM data space is to be suballocated to the object. Objects with the SUBALLOCATION attribute do not appear in the VTOC. Only the name of the data space that contains the object appears there.

- INDEXED specifies that the cluster being defined is for a KSDS.

- KEYS(length position) specifies the length of the keys in a KSDS and their position within the records. The length of the keys is specified by length; the displacement of the keys within the record is specified by position. The first character in a record is at displacement 0.

- UPDATEPW(password) specifies an update level password for the data set being defined. The update level password permits input and update (READ, START, DELETE, WRITE, REWRITE) operations against the logical records of the data set.

- READPW(password) specifies a read level password for the object being defined. The read level password permits input (READ, START) operations against the logical records of the data set.

- ATTEMPTS(number) specifies the maximum number of times the operator can try to enter a correct password in response to a prompting message. This parameter should always specify 0 as the number.

- CATALOG(catname[/password][dname]) specifies the name of the catalog into which the cluster is to be defined. The name of the catalog is substituted for catname. If the catalog is password protected, the password must also be supplied.

  The name of the DD statement identifying the catalog must be specified if the catalog is neither the master catalog nor a catalog identified by a JOBCAT or STEPCAT DD statement or if the catalog obtained through the default is not the desired catalog. For dname, substitute the name of the DD statement that identifies the catalog.

- RELATE specifies the name of the base cluster, as given in the (NAME(name)) field of the DEFINE CLUSTER for this

data set. This is a required parameter.

- UNIQUEKEY/NONUNIQUEKEY specifies whether each alternate key points to only one data record or to more than one. If to more than one, then NONUNIQUEKEY must be specified and the COBOL program must contain the WITH DUPLICATES phrase in the associated ALTERNATE RECORD KEY. A specification of UNIQUEKEY requires that the COBOL program not have such a WITH DUPLICATES phrase.

- UPGRADE specifies that this alternate index is to be kept up to date when its base cluster is modified. This is a required parameter.

- PATHENTRY specifies the name of the alternate index, as given in the (NAME(name)) field for the related DEFINE ALTERNATEINDEX. This is a required parameter.

- UPDATE specifies that the base cluster's upgrade set is to be allocated when the path is opened. This allows updating of alternate indexes (see UPGRADE above), and is a required parameter.

ADDITIONAL PARAMETERS: Additional parameters are valid for DEFINE CLUSTER, ALTERNATEINDEX, and PATH. Complete details on the use of these parameters are in the publication OS/VS Access Method Services.

Defining an RRDS

Defining an RRDS is quite similar to defining a KSDS. With the following modifications, the DEFINE CLUSTER portion of Figure 73 could be used to define an RRDS:

1. Change INDEXED to NUMBERED.
2. Remove the KEYS parameter.
3. Remove the FREESPACE parameter.
4. Change the RECORDSIZE parameter so that the average and maximum value specifications are the same.

Defining an ESDS

Defining an ESDS is quite similar to defining a KSDS. With the following modifications, the DEFINE CLUSTER portion of Figure 73 could be used to define an ESDS:

1. Change INDEXED to NONINDEXED.
2. Remove the KEYS parameter.
3. Remove the FREESPACE parameter.

Reusable Data Sets

If a COBOL program wishes to use a VSAM data set for a workfile (that is, use the data set again and again during the course of processing), the REUSE parameter must be included in the DEFINE CLUSTER specification, and the data set must be opened OUTPUT. (Its status is then "unloaded.") ESDS and RRDS data sets and KSDS data sets without alternate indexes can be reused in this manner.

Miscellaneous DEFINE Cluster Considerations

The control interval is the unit of transmission of data to and from main storage. VSAM determines the size of the control interval based upon the amount of BUFFERSPACE specified and the size of the RECORDSIZE specified. If BUFFERSPACE is not specified and if the size of the records permits, VSAM uses the optimum size for the data component control intervals and 512 as the size of the index component control intervals.

COBOL FILE PROCESSING CONSIDERATIONS

The file processing considerations of importance to the COBOL programmer are: the file processing techniques available, the current record pointer, the START statement, and the error processing options available.

FILE PROCESSING TECHNIQUES

The COBOL user has three different file processing techniques available to him: sequential, random, and dynamic (a combination of sequential and random). The technique to be used is specified through the ACCESS clause of the SELECT statement.

ESDS Processing

An ESDS can only be processed sequentially. Therefore, the ACCESS clause need not be specified since the default is sequential.

KSDS and RRDS Processing

A KSDS or an RRDS can be processed sequentially, randomly, or both sequentially and randomly. To process sequentially, ACCESS IS SEQUENTIAL is specified. To process randomly, ACCESS IS RANDOM is specified. To process both sequentially and randomly, ACCESS IS DYNAMIC is specified.

ACCESS IS DYNAMIC provides the greatest flexibility since most of the capabilities of both sequential and random processing are available. Subsequent to an OPEN statement processing can be switched from sequential to random and vice-versa, as many times as desired.

PASSWORD USAGE

The following procedures must be used when password support is employed with the VSAM data sets:

- Through Access Method Services (at DEFINE time), the programmer must password-protect the base cluster (as opposed to the data and its index separately). This is the password specified with the RECORD or RELATIVE KEY. If the data set is a KSDS with alternate keys, then the programmer must also password-protect either the path to the base cluster via an alternate index, or the alternate index itself. This is the password specified with the ALTERNATE RECORD KEY.

- In the COBOL program, the user must specify the correct level of the password: read-only, update, and so on. Failure to do so will cause a rejection of the action request which violates the protection.

- In the COBOL program, the password (if present for the data set) must be specified for every ALTERNATE RECORD KEY defined in the program--regardless of whether any accessing will ever actually be done using them. (This requirement does not apply if the file is opened only for OUTPUT and the user does not request a dynamic invocation of Access Method Services via the AIXBLD option.)

- All required passwords must be correctly specified for the file before the COBOL OPEN will succeed.

CURRENT RECORD POINTER

The current record pointer (CRP), a conceptual pointer, is applicable only to sequential requests for ESDS, RRDS, and KSDS. Basically, the current record pointer indicates the next record to be accessed by a sequential request; the CRP has no meaning for random processing or output operations. The CRP is affected only by the OPEN, START and READ statements; it is not used or affected by the WRITE, REWRITE, or DELETE statements.

In general, the last request on a file that establishes the CRP (OPEN, READ, or START) must have been successful in order for the sequential READ to be successful.

Example 1:

If the sequence of I/O operations on a file with ACCESS IS DYNAMIC and opened I-O is:

```
READ        (After setting record key to 10)
WRITE       (After setting record key to 44)
READ NEXT
```

the READ NEXT will get record 11 if the previous READ was successful. If the previous READ was not successful, the STATUS KEY will be set to 94 (No Current Record Pointer) when the READ NEXT is attempted. This occurs independently of the success of the intervening WRITE.

Generally, a READ NEXT must be preceded by a request that establishes the CRP (OPEN, START, READ, READ NEXT). If the request that establishes the CRP is unsuccessful, the READ NEXT causes the STATUS KEY to be set to 94.

Example 2:

In this example, ACCESS IS SEQUENTIAL is specified for a KSDS; therefore, records are retrieved in ascending key sequence starting at the position indicated by the CRP.

```
OPEN INPUT                (CRP at record with
                           lowest key in file)
(Set record key to 10)
START                     (CRP at record 10)
READ                      (Read record 10)
(Set record key to 5)
START                     (CRP at record 5)
READ                      (Read record 5; CRP
                           set to record 6)
READ                      (Read record 6; CRP
                           set to record 7)
READ                      (Read record 7:CRP
                           set to record 8)
```

Note that the CRP can be moved around randomly through the use of the START statement but all reading is done sequentially from that point.

If the START request for record key 5 had failed with no record found, the subsequent READ requests would have failed because there would have been no current record pointer.

Example 3:

In this example ACCESS IS DYNAMIC is specified. Therefore, records are accessed randomly if a READ is specified and sequentially if READ NEXT is specified. The highest key is 44.

```
OPEN INPUT              (CRP is set to
                         lowest key on file)
(Set record key to 5)
READ                    (Read record 5; set
                         CRP to record 6)
READ NEXT               (Read record 6; set
                         CRP to record 7)
READ NEXT               (Read record 7; set
                         (CRP to record 8
(Set record key to 43)
START                   (Position CRP to
                         record 43)
READ NEXT               (Read record 43; set
                         CRP to 44)
(Set record Key to 47)
START
READ NEXT               (Fails - no CRP)
```

The last READ NEXT failed because the preceding START was unsuccessful; in this data set there is no record 47.

Example 4:

In this updating example, ACCESS IS DYNAMIC is specified; the REWRITE statement does not affect the CRP.

```
OPEN I-O                (CRP is set to
                         first record on
                         file)
(Set record Key to 10)
READ                    (Read record 10; set
                         CRP to record 11)
REWRITE                 (Updates record 10;
                         CRP remains at
                         record 11)
READ NEXT               (Read record 11;
                         set CRP to record
                         12)
(Set record key to 74)
REWRITE                 (Fails - record not
                         found in this data
                         set
READ NEXT               (Read record 12; set
                         CRP to record 13)
```

Note that although the last REWRITE failed, the following READ NEXT was successful. The REWRITE failed because record 74 was not read before the REWRITE was attempted.

Example 5:

In this example, ACCESS IS DYNAMIC is specified for a KSDS with an alternate record key, AIXKEY, defined. Assume that the file contains eight records whose primary and alternate key values are as follows:

Record	Primary Key	Alternate Key
1st	5	100
2nd	10	70
3rd	15	80
4th	20	85
5th	25	75
6th	30	50
7th	35	95
8th	40	55

```
OPEN I-O                (CRP is set to the
                         first record of file
                         and the key of
                         reference is the
                         primary key)
(set record key to 10)
READ (without KEY clause)
                        Read second record;
                         set CRP to third
                         record)
(set alternate key to 50)
READ KEY IS AIXKEY      (the key of
                         reference is the
                         alternate key; read
                         sixth record; set
                         CRP to eighth
                         record)
READ NEXT               (the key of
                         reference remains
                         the alternate key;
                         read eighth record;
                         set CRP to second
                         record)
(set primary key to 45
and alternate key to 90)
WRITE                   (write ninth record;
                         CRP remains at
                         second record; the
                         key of reference
                         also remains the
                         alternate key)
READ NEXT               (read second record;
                         CRP is set to fifth
                         record)
(set alternate key to 100)
START KEY = AIXKEY      (CRP is set to first
                         record; the key of
                         reference is the
                         alternate key)
READ NEXT               (read first record;
                         CRP is set so that
                         the next sequential
                         read results in the
                         AT END condition)
READ NEXT               (The AT END
```

condition is raised; CRP is undefined)

USE OF THE START VERB

In some of the preceding examples, the START verb was used to position the CRP. Then the READ (for ACCESS IS SEQUENTIAL) and READ NEXT (for sequential processing when ACCESS IS DYNAMIC) retrieves the record pointed to by the CRP as established by the START.

Example:

```
05  RECORD-KEY.
    10  GEN11.
        15  GEN12   PIC 99.
        15  GEN13   PIC 99.
    10  GEN14       PIC 9.
```

In this example, GEN12, GEN11, or RECORD-KEY could be used as the data-name in the "KEY IS relational data-name" option of the START statement. The lengths would be 2, 4, and 5 respectively. GEN13 and GEN14 could not be used as they are not in the leftmost part of RECORD-KEY.

Assume that the value of RECORD-KEY is 01472:

- START file-name KEY=GEN11 would position the CRP to the first record on the file whose key has 0147 in the first 4 bytes.
- START filename KEY > GEN12 would position the CRP to the first record in the file whose key has the first two bytes greater than 01.

ERROR PROCESSING OPTIONS

The error processing options available to the COBOL programmer are INVALID KEY, EXCEPTION/ERROR procedure, and STATUS KEY. These options can be used in combination with each other.

The Importance of Status Key

All errors in processing a VSAM file, whether a logic error on the part of the COBOL programmer or an I/O error on the external storage media, return control to the COBOL program. Upon return to the COBOL program, the Status Key will indicate the status of the last request on the file. Figure 74 indicates the possible value of the Status Key and their associated general meanings.

Warning: It is essential that all VSAM files have a Status Key associated with them and that the programmer always check the contents of the Status Key after each I/O request. If Status Key is not used (and an EXCEPTION/ERROR procedure is not present), serious errors will go undiscovered by the program--which does not abend. The continued processing in such a situation may produce results that are not only destructive but difficult to detect.

Invalid Key

If the INVALID KEY option is specified in the statement causing an invalid key condition, control is transferred to the INVALID KEY imperative-statement. Any EXCEPTION/ERROR declarative procedure specified for this data set is not executed. If the FILE-STATUS clause is specified, a value is placed into the Status Key to indicate INVALID KEY condition.

EXCEPTION/ERROR Procedure

The EXCEPTION/ERROR procedure is invoked only when a file is in the open status.

Status Key 1 Value	Meaning	Status Key 2 Value	Meaning
0	Successful Completion	0	No Further Information
		2	Duplicate Key Found, And Program Specified the DUPLICATES Option
1	At End (no next logical record, or an OPTIONAL file not available at OPEN time)	0	No Further Information
2	Invalid Key	0	No Further Information
		1	Sequence Error
		2	Duplicate Key Found, But Duplicate Keys Not Allowed
		3	No Record Found
		4	Boundary Violation on WRITE to VSAM indexed or relative file (space not found to add requested record)
3	Permanent Error (data check, parity check, transmission error)	0	No Further Information
		4	Boundary Violation on WRITE to sequential VSAM file (space not found to add requested record)
9	Other Errors	0	No Further Information
		1	Password Failure
		2	Logic Error
		3	Resource Not Available
		4	No Current Record Pointer For Sequential Request
		5	Invalid Or Incomplete File Information
		6	No DD Card
		7	The data set was not properly closed; an implicit VERIFY has therefore been issued, and the file then successfully opened.

Figure 74. Status Key Values And Their Meanings

Error Handling Considerations

Figure 75 is a table of actions taken for all the combinations of AT END, INVALID KEY and EXCEPTION/ERROR Procedure based on the first character of the Status Key return. Note that the return is always to the next sentence unless the request that caused the error contained on AT END or INVALID KEY clause. Note also that the EXCEPTION/ERROR Procedure is executed only if the file is in the open status.

First Digit of Status Key	No EXCEPTION/ERROR Procedure		With EXCEPTION/ERROR Procedure	
	AT END or INVALID KEY	No AT END or INVALID KEY	AT END or INVALID KEY	No AT END or INVALID KEY
0	Return to next sentence.	Return to next sentence.	Return to next sentence.	Return to next sentence.
1	Return to AT END/INVALID KEY address.	Return to next sentence.	Return to AT END/INVALID KEY address.	Return to next sentence after execution of EXCEPTION/ERROR Procedure.
2	Return to AT END/INVALID KEY address.	Return to next sentence.	Return to AT END/INVALID KEY address.	Return to next sentence after execution of EXCEPTION/ERROR Procedure.
3	Return to next sentence.	Return to next sentence.	Return to next sentence after execution of EXCEPTION/ERROR Procedure.	return to next sentence after execution of EXCEPTION/ERROR Procedure.
9	Return to next sentence.	Return to next sentence.	Return to next sentence after execution of EXCEPTION/ERROR Procedure.	Return to next sentence after execution of EXCEPTION/ERROR Procedure.

Figure 75. Error Handling Actions Based on COBOL Program Coding.

OPENING A VSAM FILE

VSAM files can be opened INPUT, OUTPUT, EXTEND, or I-O.

Opening an Unloaded File

An unloaded file is a file that has never contained records. It is normally opened OUTPUT or EXTEND.

While certain types of unloaded files may also be successfully opened INPUT or I-O, the following conditions will occur if attempts are then made to access it.

If unloaded file is opened INPUT and operation is:

- READ sequential--will fail, with status key set to 10
- READ random--will fail, with status key set to 23
- START--will fail, with status key set to 23
- Any other request--will fail, with status key set to 92

If unloaded file is opened I-O and first operation is:

- WRITE--will succeed, if a valid request, and any subsequent request will be treated as if made to a loaded file.
- NOTE: A WRITE to a sequentially-accessed file is not a valid request when opened I-O.
- READ sequential (if first request)--will fail, with status key set to 10
- READ random (if first request)--will fail, with status key set to 23
- START (if first request)--will fail, with status key set to 23
- REWRITE/DELETE sequential (if first request)--will fail, with status key set to 92
- REWRITE/DELETE random or dynamic (if first request)--will fail with status key set to 23

Opening an Empty File

An empty file is a previously created file from which all records have been deleted. It can be opened EXTEND, INPUT or I-O. After opening, the first READ request will cause an AT END condition (Status Key = 10) or an INVALID KEY condition (Status Key = 23), depending on the access mode of the file. An empty file cannot be opened OUTPUT.

Opening a File Containing Records

A file containing records can be opened INPUT, EXTEND or I-O. If a KSDS is opened EXTEND, the first record to be added must have its record key higher than the highest record key on the file when it was opened. If the record key is not higher the status key will be set to 92. For an ESDS, the records are added after the last record on the file. An RRDS cannot be opened EXTEND.

OPEN Status Key Values

If any of the OPEN rules are violated the file is not opened and the Status Key is set to the appropriate value. See Figure 76 for the OPEN Status Key values and their meanings.

Dynamic Invocation of Access Method Services for KSDS and RRDS Data Sets

(Note: The following feature is provided only to assure compliance with the 1974 ANS COBOL standard X3.23-1974. Use of the feature will necessarily adversely affect execution-time performance.)

As described earlier, the user must employ Access Method Services to define all VSAM files and their indexes ahead of time--outside of the COBOL program. Normally at this time, the user specifies the size of the file's records, and (for KSDS) the lengths and offsets of primary and any alternate keys. He also builds the actual alternate indexes.

However, if he wishes, the user may choose to omit these elements of the definition procedure and have COBOL automatically perform them later when the file is opened for OUTPUT processing. COBOL does this by obtaining the correct record and key specifications from the program's source statements, and then issuing the Access Method Services ALTER and BLDINDEX commands.

If this is the user's choice, he must make it known to COBOL by including the object-time (GO-step) option AIXBLD. (He must also ensure that the Access Method Services program is available for his COBOL program to invoke.)

For an RRDS, COBOL will then fill in the record size information. For a KSDS, COBOL will fill in record size, and length and displacement for primary keys. If the KSDS has alternate keys, COBOL also fills in their length and displacement, and issues the BLDINDEX command. (Because the BLDINDEX command can be issued only after a base cluster is loaded, COBOL first fills it with dummy records, then issues the BLDINDEX command, and then erases them.)

Because the large Access Method Services program must be present when the COBOL program is run with this feature, substantial extra main storage is required. Use of the AIXBLD feature also requires the user to provide a SYSPRINT DD card for Access Method Services messages. If this card is missing, OPEN failure will result (Status Key = 95).

If the Status Key given to the user was...	either COBOL itself detected one of these conditions...	...or VSAM found an error and returned one of the following VSAM error codes*
00	None	100 (if AIXBLD was specified) 128 (if file is optional) 160 (if file was to be opened for input)
30	None	132,144,164,176, 184
90	Failure of an attempt to write a dummy record to or delete a dummy record from the file. Such a failure may occur in the following cases: -- an indexed file is opened for OUTPUT and the access mode is either RANDOM or DYNAMIC. -- a file to be opened I-O has just been created. -- the object-time option AIXBLD was specified and the file has at least one alternate record key. Failure of attempt to use a CBMM macro. Failure of attempt to use a BLDL system macro; this macro is used when the programmer has specified the object-time option AIXBLD.	96,108,116,192, 196,200,204,208, 212,216,220,224, 236,240,244 (plus any other VSAM codes not appearing elsewhere in this table)
91	None	152
92	The file to be opened is already open.	4

* COBOL's VSAM-processing subroutines retrieve the VSAM return code from the ACBERFLG field of the Access Control Block (ACB) by issuing a SHOWCB macro, translate it into one of the status key values above (as prescribed by the ANS standard), and move that value into the STATUS KEY field where it becomes accessible in the user's program. An explanation of the meanings of the VSAM return codes can be found in the VSAM Programmer's Guide.

Figure 76. (Part 1 of 2) Status Key Values for OPEN Requests

If the Status Key given to the user was...	either COBOL itself detected one of these conditions...	...or VSAM found an error and returned one of the following VSAM error codes*
93	COBOL cannot obtain sufficient virtual storage for: -- the general work area used by the COBOL VSAM-interface modules. -- the Access Control Block (ACB) address list area during the OPEN process. -- the work area required for the invocation of Access Method Services. -- processing of the user declarative.	136 168
95	The ENDRBA of a file to be opened OUTPUT is not zero. The length and/or offset of the key of each cluster do not match those in the catalog. A KSDS cluster is to be opened as a COBOL sequential (ESDS) file. An attempt was made to either alter the record size and/or key information of a cluster or build alternate indexes, and Access Method Services returned a non-zero return code.	100 (if AIXBLD was not specified) 104,108,148, 160 (if a file is not to be opened for INPUT) 180,188,232
96	No DD card is present for a path to be opened.	128 (if a file is not optional)
97	OPEN is successful for a data set with alternate keys opened I-O, EXTEND, or OUTPUT, and an implicit VERIFY has occurred.	118

* COBOL's VSAM-processing subroutines retrieve the VSAM return code from the ACBERFLG field of the Access Control Block (ACB) by issuing a SHOWCB macro, translate it into one of the status key values above (as prescribed by the ANS standard), and move that value into the STATUS KEY field where it becomes accessible in the user's program. An explanation of the meanings of the VSAM return codes can be found in the VSAM Programmer's Guide.

Figure 76. (Part 2 of 2) Status Key Values for OPEN Requests

INITIAL LOADING OF RECORDS INTO A FILE

Initial loading refers to writing records into a file after it is opened for the first time; this is distinctly different than writing records into an empty file (a previously created file from which all records have been deleted).

When the file is unloaded and is opened EXTEND, it is processed exactly the same as it would be had it been opened OUTPUT.

It is recommended that initial loading of records into a KSDS always be done sequentially. This assists in optimizing performance for the initial loading process as well as for any subsequent processing on the file. Loading records randomly does not conserve any free space in the file and, as a result, any future inserts require the file to be dynamically reorganized.

WRITING RECORDS INTO A VSAM FILE

The COBOL WRITE statement adds a record to a file; it does not replace an existing record on the file. The record to be written must not be larger than the maximum record size specified when the file was defined.

ESDS Considerations

Records are written sequentially.

KSDS Considerations - (ACCESS IS SEQUENTIAL)

The records must be written in ascending key sequence. If the file is opened EXTEND, the record keys of the records to be added must be higher than the highest record key on the file when the file was opened.

For example, a file containing records whose records keys are 2, 4, 6, 8 and 10 is opened EXTEND; the following actions take place for the sequence of operations shown:

```
WRITE   (record key = 8)     SK = 92
WRITE   (record key = 9)     SK = 92
WRITE   (record key = 12)    SK = 00
WRITE   (record key = 11)    SK = 21
```

If many records are to be added to the end of a file, it is recommended that sequential processing be used. It assists in optimizing processing for both the addition of records as well as later retrieval of them.

KSDS Considerations - (ACCESS IS RANDOM/DYNAMIC)

When a file has alternate keys, the records must be written using their primary keys.

RRDS Considerations

For a sequential request, the first record written will have relative record number one, the second two, the third three, and so on. If a RELATIVE KEY data item was included by the user in the file control entry statement, the relative record number of the record just written will be placed in the data item.

REWRITING RECORDS ON A VSAM FILE

The COBOL REWRITE statement is used to replace an existing record in the file.

ESDS Considerations

The file must be opened I-O; if not, the record is not rewritten and the Status Key is set to 92. The record to be rewritten must first be read by the COBOL program; the record may then be rewritten. If there was no preceding READ, or if the preceding READ was not successful, the record is not rewritten and the Status Key is set to 92. If an attempt is made to change the length of the record to be rewritten, the Status Key is set to 92.

KSDS Considerations

The file must be opened I-O; if not, the record is not rewritten and the Status Key is set to 92. The length of the record can be changed; the value of the record key cannot be changed.

For ACCESS IS SEQUENTIAL, or files containing spanned records, the record to be rewritten must first be read by the COBOL program. The REWRITE then updates the record that was read. If the REWRITE is not preceded by a successful READ of the record to be rewritten, the rewrite is not done and the Status Key is set to 92.

For ACCESS IS RANDOM/DYNAMIC, and for records that are not spanned, the record to be rewritten need not be read by the COBOL program. To update a record, the key of the record to be updated is moved to the RECORD KEY data-name and then the REWRITE is issued.

Rewiting must always be done by the primary key. COBOL does, however, allow a user to change the alternate key contents while rewriting the record.

READING RECORDS ON A VSAM FILE

The COBOL READ statement is used to access records on a file. The file must be opened INPUT or I-O; if not, the record is not read and the Status Key is set to 92.

ESDS Considerations

The records are read in the sequence in which they were written.

KSDS Considerations - (ACCESS IS SEQUENTIAL)

Records are read sequentially beginning at the position of the Current Record Pointer. If the Current Record Pointer is not defined at the time the READ is issued, the READ fails and the Status Key is set to 94.

The Current Record Pointer is undefined if a START is unsuccessful. For example:

OPEN I-O filename	CRP set to first record on file.
READ	First record is read.
(Set Record Key to 10)	
START	Fails--no record found. SK=23.
READ	Fails--no CRP. SK = 94.
(Set Record Key to 20)	
START	Successful.
READ	Record 20 is read.
READ	EOF encountered; SK = 10.
READ	Logic error;SK=92
(Set Record Key to 8)	
START	Successful.
READ	Record 8 is read.

KSDS Considerations - (ACCESS IS RANDOM)

Records are read in the order specified in the COBOL program. For example, to readthe record whose Record Key is 10, the RECORD KEY field must be set to 10 and then a READ is issued.

KSDS Considerations - (ACCESS IS DYNAMIC)

Records can be read sequentially or randomly. The READ NEXT statement is used for sequential accessing while the READ statement is used for random accessing. Within any given program, both sequential and random processing may be performed.

SEQUENTIAL PROCESSING: Records are read sequentially beginning at the position of the Current Record Pointer. If the Current Record Pointer is undefined when the READ NEXT is issued, the record is not read and the Status Key is set to 94. The Current Record Pointer is undefined if the previous START or READ was unsuccessful. See the discussion of Current Record Pointer for more details and examples of the effect of different COBOL statements on the positioning of the Current Record Pointer.

RANDOM PROCESSING: Records are read randomly according to the value placed in the record key field.

RRDS Considerations

If a RELATIVE KEY data item was specified for a sequential READ, the relative record number of the record just read will be placed in the data item.

DELETING RECORDS ON A FILE

The COBOL DELETE statement is used to remove an existing record on a KSDS. DELETE cannot be used with an ESDS.

The file must be opened I-O; if not, the record is not deleted and the Status Key is set to 92.

For ACCESS IS SEQUENTIAL, or files containing spanned records, the record to be deleted must first be read by the COBOL program. The DELETE then removes the record that was read. If the DELETE is not preceded by a successful READ of the record to be deleted, the deletion is not done and the Status Key is set to 92.

For ACCESS IS RANDOM/DYNAMIC, and for records that are not spanned, the record to be deleted need not be read by the COBOL program. To delete a record, the key of the record to be deleted is moved to the RECORD KEY data-name and the DELETE is issued.

STATUS KEY SETTINGS FOR ACTION REQUESTS

Figure 77 is a summary of the Status Key values that can occur for action requests. Status Key 92 has numerous possible causes as described below.

Status Key 92 can be caused by:

- Any request against a file that is not open.
- Any request that is not allowed for the option that was specified with the OPEN statement. For example, an attempt is made to read a file that was opened as OUTPUT or an attempt is made to rewrite on a file opened as INPUT.
- Any attempt to write or rewrite a record longer than the maximum record size specified when the file was defined.

- Any attempted action on a file after the end-of-file condition has occurred. This is applicable to ESDS, RRDS, and KSDS; however, on an RRDS or KSDS a START or READ can be issued to set the Current Record Pointer to another point in the file so that processing may continue. For example:

  1. ACCESS IS SEQUENTIAL

```
OPEN
READ     Successful
READ     EOF encountered
READ     Logic error
START    To reset Current Record
           Pointer
READ     Successful
```

  2. ACCESS IS DYNAMIC

```
OPEN
READ  NEXT   Successful
READ  NEXT   EOF encountered
READ  NEXT   Logic error
READ         Random READ to reset
               CRP
READ  NEXT   Successful
```

- An attempt to rewrite, when access is sequential, after an unsuccessful READ.

- An attempt to delete, when access is sequential, after an unsuccessful READ. This applies to KSDS and RRDS only, since DELETE is not legal for ESDS.

CLOSING A FILE

If the user attempts to close a file which has already been closed, COBOL returns a status key value of 92. When performing a CLOSE request, VSAM itself may detect an error and return one of the following codes to COBOL: 132, 144, 164, 176, or 184. COBOL will translate this VSAM code into a STATUS KEY of 30.

If the Status Key given to was...	either COBOL itself discovered one of these conditions...	or VSAM found an error and returned one of the following VSAM error codes*
02	Permissible duplicate key follows for READ, or permissible duplicate key is created on one or more alternate indexes for WRITE or REWRITE. (This is the case when the feedback field of the RPL contains 8 but the LERAD exit is not taken.)	None
10	READ is issued for the first time to an optional file. Sequential READ is issued to an empty file opened for INPUT.	4 (if the request was not START)
21	None	12,96
22	None	8
23	Random READ or START issued to an empty file opened for INPUT. Relational operator GREATER THAN was specified in a START and the key contains HIGH-VALUE. Current record pointer failed for sequential READ because key used in the previous READ contained HIGH-VALUE.	4 (if the request was START) 16 192
24	Relative record key contains a value larger than allowed.	28 (if the file is not ESDS) 148
30	SYNAD exit taken due to an I/O error.	140
34	None	28 (if the file is ESDS)
90	Failure of attempt to use a CBMM macro.	32,64,68,72,76, 80,84,104,112, 116,132,136,144, 196,200 (plus any VSAM codes not appearing elsewhere in this table)

* COBOL's VSAM-processing subroutines retrieve the VSAM return code from the RPLFDBK field of the Request Parameter List (RPL), translate it into one of the status key values above (as prescribed by the ANS standard), and move that value into the STATUS KEY field where it becomes accessible in the user's program. An explanation of the meanings of the VSAM return codes can be found in the VSAM Programmer's Guide.

Figure 77. (Part 1 of 2) Status Key Values for Action Requests

If the Status Key given to was...	either COBOL itself discovered one of these conditions...	or VSAM found an error and returned one of the following VSAM error codes*
92	Impermissible request (action does not match file's open mode). File is not open. End-of-file condition had been raised by the previous operation, and a sequential READ is issued, or a REWRITE is issued when access mode is sequential, or a DELETE is issued and the access mode is sequential. Access mode is sequential, and the last I/O request for the file (prior to a REWRITE or DELETE) was not a successful READ. READ issued to an optional file is not the first READ request. The key value of a record to be added to an indexed file opened EXTEND is not the highest among record key values in the file.	36,44,92,100, 108,152,204
93	Insufficient virtual storage for the user declarative processing.	20,24,40
94	The current record pointer (maintained by ILBOVIO) is undefined for this sequential READ.	88

* COBOL's VSAM-processing subroutines retrieve the VSAM return code from the RPLFDBK field of the Request Parameter List (RPL), translate it into one of the status key values above (as prescribed by the ANS standard), and move that value into the STATUS KEY field where it becomes accessible in the user's program. An explanation of the meanings of the VSAM return codes can be found in the VSAM Programmer's Guide.

Figure 77. (Part 2 of 2) Status Key Values for Action Requests

COBOL LANGUAGE USAGE WITH VSAM

The COBOL language statements which are directly related to VSAM processing are in the publication IBM VS COBOL for OS/VS. The following paragraphs are intended only to highlight and summarize the basic language statements used in writing a VSAM-file-processing COBOL program.

The COBOL programmer can use VSAM in three basic ways: to write, to retrieve, and to update records in a data set. However, prior to processing a VSAM data set, it is an absolute necessity that the previously discussed Access Method Services functions are performed. Most significant to the COBOL programmer is whether the data set is defined as ESDS, KSDS or RRDS.

WRITING A VSAM DATA SET

The COBOL language statements frequently used to fill in a VSAM data set are summarized in Figure 78. Examples 1 and 2 illustrate the creation of an ESDS and a KSDS respectively.

	ESDS	KSDS	RRDS
Environment Division	SELECT ASSIGN FILE STATUS PASSWORD ACCESS MODE	SELECT ASSIGN ORGANIZATION IS INDEXED RECORD KEY ALTERNATE RECORD KEY FILE STATUS PASSWORD ACCESS MODE	SELECT ASSIGN ORGANIZATION IS RELATIVE RELATIVE KEY FILE STATUS PASSWORD ACCESS MODE
Data Division	FD entry LABEL RECORDS	FD entry LABEL RECORDS	FD entry LABEL RECORDS
Procedure Division	OPEN OUTPUT OPEN EXTEND WRITE CLOSE	OPEN OUTPUT OPEN EXTEND WRITE CLOSE	OPEN OUTPUT WRITE CLOSE

Figure 78. COBOL Statements Frequently Used for Writing into a VSAM Data Set

Example 1:

This example shows the creation of a COBOL ESDS. The FILE STATUS facility is used to monitor all I/O operations in the program.

```
IDENTIFICATION DIVISION.
        .
        .
        .
ENVIRONMENT DIVISION.
        .
        .
        .
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT INREC
      ASSIGN TO UR-2540R-S-INFILE.
    SELECT  OUTREC
      ASSIGN TO AS-OUTFILE
      FILE STATUS IS CHK.
        .
        .
        .

DATA DIVISION.
FILE SECTION.
FD  INREC LABEL RECORDS ARE OMITTED
    DATA RECORD IS INMASTER.
01  INMASTER  PIC X(80).
FD  OUTREC LABEL RECORDS ARE STANDARD
    DATA RECORD IS OUTMASTER.
01  OUTMASTER PIC X(80).
WORKING-STORAGE SECTION.
77  CHK PIC 99 VALUE ZEROS.
PROCEDURE DIVISION.
PARA1.
    OPEN INPUT INREC OUTPUT OUTREC.
    IF CHK IS NOT = 00 GO TO CHKRTN.
PARA2.
    READ INREC INTO OUTMASTER
      AT END GO TO PARA4.
PARA3.
    WRITE OUTMASTER.
    IF CHK IS NOT = 00 GO TO CHKRTN.
    GO TO PARA2.
PARA4.
    CLOSE  INREC  OUTREC.
    IF CHK IS NOT = 00  GO TO CHKRTN.
FINIT.
    STOP RUN.
CHKRTN.
    DISPLAY 'I/O ERROR. STATUS KEY VALUE
      IS' CHK.
    GO TO FINIT.
```

Note that in this example any Status Key return other than 00 causes transfer of control to paragraph CHKRTN. This user-created routine can determine the exact cause of the error by checking the Status Key. Once the cause is determined, instructions can be issued according to the user's desired response to each type of error.

Example 2:

This example shows the creation of a COBOL KSDS; this program performs the same function as Example 1 except that now a KSDS is being created.

```
IDENTIFICATION DIVISION.
        .
        .
        .
ENVIRONMENT DIVISION.
        .
        .
        .
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT INREC
      ASSIGN TO UR-2540R-S-INFILE.
```

```
    SELECT  OUTREC
      ASSIGN TO DA-2319-OUTFILE
      ORGANIZATION IS INDEXED
      RECORD KEY IS ARG-1
      FILE STATUS IS CHK.
        .
        .
        .
DATA DIVISION.
FILE SECTION.
FD  INREC LABEL RECORDS ARE OMITTED
    DATA RECORD IS INMASTER.
01  INMASTER  PIC  X(80).
FD  OUTREC LABEL RECORDS ARE STANDARD
    DATA RECORD IS OUTMASTER.
01  OUTMASTER.
    05  FILLER  PIC X.
    05  ARG-1  PIC XXX.
    05  REM  PIC X(76).
WORKING-STORAGE SECTION
77  CHK  PIC 99 VALUE ZERO.
PROCEDURE DIVISION.
PARA1.
    OPEN INPUT INREC  OUTPUT OUTREC.
    IF CHK IS NOT = 00 GO TO CHKRTN.
PARA2.
    READ INREC INTO OUTMASTER.
        AT END GO TO PARA4.
PARA3.
    WRITE OUTMASTER.
    IF CHK IS NOT = 00 GO TO CHKRTN.
    GO TO PARA2.
PARA4.
    CLOSE INREC OUTREC.
    IF CHK IS NOT = 00 GO TO CHKRTN.
FINIT.
    STOP RUN.
CHKRTN.
    DISPLAY 'I/O ERROR. STATUS KEY VALUE
      IS' CHK.
    GO TO FINIT.
```

Note that in this example any Status Key return other than 00 causes transfer of control to paragraph CHKRTN. This user-created routine can determine the exact cause of the I/O error by checking the Status Key. Once the cause is determined, instructions can be issued according to the user's desired response to each type of error.

Example 3:

This example also shows the creation of a COBOL KSDS, but with the addition of an alternate key; this program serves the same function as Example 2.

```
IDENTIFICATION DIVISION.
        .
        .
        .
ENVIRONMENT DIVISION.
        .
        .
        .
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT INREC
      ASSIGN TO UR-2540R-S-INFILE.
    SELECT  OUTREC
      ASSIGN TO DA-2319-OUTFILE
      ORGANIZATION IS INDEXED
      RECORD KEY IS ARG-1
      ALTERNATE RECORD KEY IS ARG-2
      FILE STATUS IS CHK.
        .
        .
        .
DATA DIVISION.
FILE SECTION.
FD  INREC LABEL RECORDS ARE OMITTED
    DATA RECORD IS INMASTER.
01  INMASTER PIC X(80).
FD  OUTREC LABEL RECORDS ARE STANDARD
    DATA RECORD IS OUTMASTER.
01  OUTMASTER.
    05  FILLER  PIC X.
    05  ARG-1  PIC XXX.
    05  ARG-2  PIC XXXXX.
    05  REM  PIC X(71).
WORKING-STORAGE SECTION
77  CHK  PIC 99 VALUE ZERO.
PROCEDURE DIVISION.
PARA1.
    OPEN INPUT INREC  OUTPUT OUTREC.
    IF CHK IS NOT = 00 GO TO CHKRTN.
PARA2.
    READ INREC INTO OUTMASTER.
        AT END GO TO PARA4.
PARA3.
    WRITE OUTMASTER.
    IF CHK IS NOT = 00 GO TO CHKRTN.
    GO TO PARA2.
PARA4.
    CLOSE INREC OUTREC.
    IF CHK IS NOT = 00 GO TO CHKRTN.
FINIT.
    STOP RUN.
CHKRTN.
    DISPLAY 'I/O ERROR. STATUS KEY VALUE
      IS' CHK.
    GO TO FINIT.
```

Note that in this example any Status Key return other than 00 causes transfer of control to paragraph CHKRTN. This user-created routine can determine the exact cause of the error by checking the Status Key. Once the cause is determined, instructions can be issued according to the user's desired response to each type of error.

RETRIEVING RECORDS FROM A VSAM DATA SET

The COBOL language statements frequently used to retrieve records from a VSAM data set are summarized in Figure 79. Examples 3 and 4 illustrate the retrieval of records from an ESDS and KSDS, respectively.

	ESDS	KSDS	RRDS
Environment Division	SELECT ASSIGN FILE STATUS PASSWORD ACCESS MODE	SELECT ASSIGN ORGANIZATION IS INDEXED RECORD KEY ALTERNATE RECORD KEY FILE STATUS PASSWORD ACCESS MODE	SELECT ASSIGN ORGANIZATION IS INDEXED RELATIVE KEY ALTERNATE RECORD KEY FILE STATUS PASSWORD ACCESS MODE
Data Division	FD entry LABEL RECORDS	FD entry LABEL RECORDS	FD entry LABEL RECORDS
Procedure Division	OPEN INPUT READ...AT END CLOSE	OPEN INPUT READ CLOSE	OPEN INPUT READ CLOSE

Figure 79. COBOL Statements Frequently Used for Retrieving Records From a VSAM Data Set

Example 4.

This example shows the retrieval of records from the ESDS created in example 1. The records are then printed.

```
IDENTIFICATION DIVISION.
        .
        .
        .
ENVIRONMENT DIVISION.
        .
        .
        .
INPUT-OUTPUT SECTION.
FILE-CONTROL.
   SELECT  INREC
   ASSIGN TO AS-OUTFILE
   FILE STATUS IS CHK.
   SELECT PREC
   ASSIGN TO UR-1403-S-PFILE.
        .
        .
        .
DATA DIVISION.
FILE SECTION.
FD INREC  LABEL RECORDS ARE STANDARD
   DATA RECORD IS INMASTER.
01 INMASTER  PIC X(80).
FD PREC LABEL RECORDS ARE OMITTED
   DATA RECORD IS POUT.
01 POUT  PIC X(80).
WORKING-STORAGE SECTION.
77 CHK  PIC 99 VALUE ZERO.
PROCEDURE DIVISION.
PARA1.
   OPEN INPUT INREC OUTPUT PREC.
IF CHK IS NOT = 00 GO TO CHKRTN.
PARA2.
READ INREC INTO POUT AT END GO TO PARA4.
   IF CHK IS NOT = 00 GO TO CHKRTN.
PARA3.
   WRITE POUT.
   GO TO PARA2.
PARA4.
   CLOSE  OUTREC PREC.
   IF CHK IS NOT = 00 GO TO CHKRTN.
FINIT.
   STOP RUN.
CHKRTN.
   DISPLAY 'I/O ERROR. STATUS KEY VALUE IS'
     CHK.
   GO TO FINIT.
```

Note that in this example any Status Key return other than 00 causes transfer of control to paragraph CHKRTN. This user-created routine can determine the exact cause of the error by checking the Status Key. Once the cause is determined, instructions can be issued according to the user's desired response to each type of error.

Example 5:

This example shows the retrieval of records from the KSDS created in example 2. Note that in the Procedure Division there is a switch from sequential processing to random processing; this is permitted since ACCESS IS DYNAMIC is specified in the Environment Division.

```
IDENTIFICATION DIVISION.
        .
        .
        .
ENVIRONMENT DIVISION.
        .
        .
        .
INPUT-OUTPUT SECTION.
FILE-CONTROL.
   SELECT  INREC
      ASSIGN TO OUTFILE
      ORGANIZATION IS INDEXED
      ACCESS IS DYNAMIC
      RECORD KEY IS ARG-1
      FILE STATUS IS CHK.
   SELECT PREC
      ASSIGN TO UR-1403-S-PFILE.
```

```
        .
        .
        .
DATA DIVISION.
FILE SECTION.
FD INREC  LABEL RECOPDS ARE STANDARD
   DATA RECORD IS INMASTER.
01 INMASTER.
   05   FILLER  PIC X.
   05   ARG-1  PIC XXX.
   05   ARG-2  PIC XX.
   05   ARG-3  PIC XX.
   05   FILLER  PIC X(72).
FD PREC LABEL RECORDS ARE OMITTED
   DATA RECORD IS POUT.
01 POUT  PIC X(80).
WORKING-STORAGE SECTION.
77 CHK  PIC 99 VALUE ZERO.
PROCEDURE DIVISION.
PARA1.
   OPEN INPUT INREC OUTPUT PREC.
   IF CHK IS NOT = 00 GO TO CHKRTN.
PARA2.
   MOVE 003 TO ARG-1.
   START INREC.
PARA3.
   READ INREC NEXT AT END GO TO PARA4.
   IF CHK IS NOT = 00 GO TO CHKRTN.
   IF ARG-2 IS = 02 GO TO PARA4.
   IF ARG-3 IS NOT = 73 GO TO PARA3.
   WRITE POUT FROM INMASTER.
   GO TO PARA3.
PARA4.
   MOVE 101 TO ARG-1.
   READ INREC INVALID KEY GO TO CHKRTN.
   WRITE POUT FROM INMASTER.
   MOVE 103 TO ARG-1.
   READ INREC INVALID KEY GO TO CHKRTN.
   WRITE POUT FROM INMASTER.
PARA5.
   CLOSE INREC PREC.
   IF CHK IS NOT = 00 GO TO CHKRTN.
FINIT.
   STOP RUN.
CHKRTN.
   DISPLAY 'I/O ERROR. STATUS KEY VALUE
     IS ' CHK.
   GO TO FINIT.
```

Note that in this example any Status Key return other than 00 causes transfer of control to paragraph CHKRTN. This user created routine can determine the exact cause of the I/O error by checking the Status Key. Once the cause is determined, instructions can be issued according to the users desired response to each type of error.

Example 6:

This example shows the retrieval of records from the KSDS created in example 3. Since ACCESS IS RANDOM is specified in the Environment Division, random processing of the file is done in the Procedure Division.

```
IDENTIFICATION DIVISION.
        .
        .
        .
ENVIRONMENT DIVISION
        .
        .
        .
INPUT-OUTPUT SECTION.
FILE-CONTROL.
SELECT INREC
      ASSIGN TO OUTFILE
      ORGANIZATION IS INDEXED
      ACCESS IS RANDOM
      RECORD KEY IS ARG-1
      ALTERNATE RECORD KEY IS ARG-2
      FILE STATUS IS CHK.
SELECT PREC
      ASSIGN TO UR-1403-S-PFILE.
        .
        .
        .
DATA DIVISION.
FILE SECTION.
FD INREC LABEL RECORDS ARE STANDARD
   DATA RECORD IS INMASTER.
01 INMASTER.
   05      FILLER  PIC X.
   05      ARG1  PIC XXX.
   05      ARG-2  PIC XXXXX.
   05      ARG-3 PIC XX.
   05      ARG-4  PIC XX.
   05      FILLER  PIC X(67).
FD PREC LABEL RECORDS ARE OMITTED
   DATA RECORD IS POUT.
01 POUT PIC X(80).
WORKING-STORAGE SECTION.
77 CHK PIC 99 VALUE ZERO.
PROCEDURE DIVISION.
PARA1.
   OPEN INPUT INREC OUTPUT PREC.
   IF CHK IS NOT = 00 GO TO CHKRTN.
PARA2.
   MOVE 003 TO ARG-1.
PARA3.
        READ INREC INVALID KEY GO TO CHKRTN.
        IF CHK IS NOT = 00 GO TO CHKRTN.
        IF ARG-3 IS = 02 GO TO PARA5.
        IF ARG-4 IS NOT = 73 GO TO PARA4.
        WRITE POUT FROM INMASTER.
        ADD 010 TO ARG-1.
        GO TO PARA3.
PARA4.
        SUBTRACT 001 FROM ARG-1.
        GO TO PARA3.
PARA5.
        MOVE 101 TO ARG-2.
        READ INREC KEY IS ARG-2 INVALID KEY
        GO TO CHKRTN.
        WRITE POUT FROM INMASTER.
        MOVE 103 TO ARG-2.
        READ INREC KEY IS ARG-2 INVALID KEY
        GO TO CHKRTN.
        WRITE POUT FROM INMASTER.
PARA6.
        CLOSE INREC PREC.
        IF CHK IS NOT = 00 GO TO CHKRTN.
FINIT.
```

```
        STOP RUN.
CHKRTN.
        DISPLAY 'I/O ERROR. STATUS KEY VALUE
          IS ' CHK.
        GO TO FINIT.
```

Note that in this example any Status Key return other than 00 causes transfer of control to paragraph CHKRTN. This user created routine can determine the exact cause of the I/O error by checking the Status Key. Once the cause is determined, instructions can be issued according to the user's desired response to each type of error.

UPDATING A VSAM DATA SET

The COBOL language statements frequently used to update a VSAM data set are summarized in Figure 80. Examples 7 and 8 illustrate the updating of an ESDS and KSDS respectively.

	ESDS	KSDS	RRDS
Environment Division	SELECT ASSIGN FILE STATUS PASSWORD ACCESS MODE	SELECT ASSIGN ORGANIZATION IS INDEXED RECORD KEY ALTERNATE RECORD KEY FILE STATUS PASSWORD ACCESS MODE	SELECT ASSIGN ORGANIZATION IS RELATIVE RELATIVE KEY FILE STATUS PASSWORD ACCESS MODE
Data Division	FD entry LABEL RECORDS	FD entry LABEL RECORDS	FD entry LABEL RECORDS
Procedure Division	OPEN EXTEND WRITE CLOSE or OPEN I-O READ ... AT END REWRITE CLOSE	For ACCESS IS SEQUENTIAL: OPEN EXTEND WRITE CLOSE or OPEN I-O READ ... AT END REWRITE DELETE CLOSE	For ACCESS IS SEQUENTIAL: OPEN I-O READ .. AT END REWRITE DELETE CLOSE
		For ACCESS IS RANDOM: OPEN I-O READ WRITE REWRITE DELETE CLOSE	For ACCESS IS RANDOM: OPEN I-O READ WRITE REWRITE DELETE CLOSE
		For ACCESS IS DYNAMIC with Sequential Processing OPEN I-O READ NEXT ... AT END WRITE REWRITE START DELETE CLOSE	For ACCESS IS DYNAMIC with Sequential Processing OPEN I-O READ NEXT ... AT END WRITE REWRITE START DELETE CLOSE
		For ACCESS IS DYNAMIC with Random Processing OPEN I-O READ WRITE REWRITE DELETE CLOSE	For ACCESS IS DYNAMIC with Random Processing OPEN I-O READ WRITE REWRITE DELETE CLOSE

Figure 80. COBOL Statements Frequently Used for Updating a VSAM Data Set

Example 7:

This example shows the updating of records from the ESDS data set created in Example 1.

```
IDENTIFICATION DIVISION.
        .
        .
        .
ENVIRONMENT DIVISION.
        .
        .
        .
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT INREC
        ASSIGN TO AS-OUTFILE
        FILE STATUS IS CHK.
        .
        .
        .
DATA DIVISION.
FILE SECTION.
FD INREC LABEL RECORDS ARE STANDARD
    DATA RECORD IS INMASTER.
01  INMASTER.
    05  FLD1  PCI X(3)
    05  FLD2  PIC X(3).
    05  FLD3  PIC X(74).
WORKING-STORAGE SECTION.
77  CHK PIC 99 VALUE ZERO.
PROCEDURE DIVISION.
PARA1.
    OPEN I-O OUTREC.
    IF CHK IS NOT = 00 GO TO CHKRTN.
PARA2.
    READ INREC AT END GO TO PARA4.
    IF CHK IS NOT = 00 GO TO CHKRTN.
PARA3.
    IF FLD2 IS NOT = 373 GO TO PARA2.
    MOVE 374 TO FLD2.
    REWRITE INMASTER.
    IF CHK IS NOT = 00 GO TO CHKRTN.
    GO TO PARA2.
PARA4.
    CLOSE INREC.
    IF CHK IS NOT = 00 GO TO CHKRTN.
FINIT.
    STOP RUN.
CHKRTN.
    DISPLAY 'I-O ERROR.  STATUS KEY
      VALUE IS'  CHK.
    GO TO FINIT.
```

Note that in this example any Status Key return other than 00 causes transfer of control to paragraph CHKRTN. This user-created routine can determine the exact cause of the I/O error by checking the Status Key. Once the cause is determined, instructions can be issued according to the users desired response to each type of error.

Example 8:

This example shows the updating of selected records in the KSDS created in Example 2; the records to be updated by the contents of CARDFILE. Note the use of the DELETE statement; this could not be used with an ESDS.

```
IDENTIFICATION DIVISION.
        .
        .
        .
ENVIRONMENT DIVISION.
        .
        .
        .
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT CARDFILE
        ASSIGN TO UR-2540R-S-INREC.
    SELECT INREC
        ASSIGN TO INFILE
        ORGANIZATION IS INDEXED
        ACCESS IS RANDOM
        RECORD KEY IS ARG-1
        FILE STATUS IS CHK
        .
        .
        .
DATA DIVISION.
FILE SECTION.
FD  CARDFILE LABEL RECORD IS OMITTED
    DATA RECORD IS INCARD.
01  INCARD
    05  CARDKEY  PIC XXX.
    05  FILLER  PIC X(77).
FD  INREC LABEL RECORDS ARE STANDARD
    DATA RECORD IS INMASTER.
01  INMASTER.
    05  FILLER  PIC X.
    05  ARG-1  PIC XXX.
    05  ARG-2  PIC XX.
    05  ARG-3  PIC XX.
    05  FILLER  PIC X(72).
WORKING-STORAGE SECTION.
77  CHK  PIC XX VALUE ZEROS.
        .
        .
        .
PROCEDURE DIVISION.
PARA1.
    OPEN INPUT CARDFILE I-O INREC.
    IF CHK IS NOT == 00 GO TO CHKRTN.
PARA2.
    READ CARDFILE AT END GO TO PARA3.
    MOVE CARDKEY TO ARG-1.
    READ INREC.
    IF CHK IS NOT = 00 GO TO CHKRTN.
    IF ARG-2 = 01 DELETE INREC RECORD
    GO TO PARA2.
    IF ARG-3 = 75 MOVE 74 TO ARG-3
    REWRITE INMASTER.
    GO TO PARA2
PARA3.
    CLOSE CARDFILE INREC.
    IF CHK IS NOT = 00 GO TO CHKRTN.
```

```
INIT.
   STOP RUN.
HKRTN.
   DISPLAY 'I/O ERROR. STATUS KEY
   VALUE IS'  CHK
       'LAST RECORD PROCESSED IS' CARDKEY.
   GO TO FINIT.
```

lote that in this example any Status Key :eturn other than 00 causes transfer of :ontrol to paragraph CHKRTN. This user-:reated routine can determine the exact :ause of the I/O error by checking the ;tatus Key. Once the cause is determined, .nstructions can be issued according to the isers desired response to each type of :rror.

JOB CONTROL LANGUAGE FOR VSAM FILE PROCESSING

JCL is greatly simplified for VSAM since all VSAM data sets must be cataloged through Access Method Services. In most cases, specification of the following DD statement will suffice:

```
//ddname   DD   DSNAME=dsname,DISP={OLD|SHR}
```

The dsname must be the same as the one specified for this data set through Access Method Services.

If the user specifies the COBOL option AIXBLD, then the DD statement must also include the parameter AMP='AMORG'.

DD STATEMENTS FOR ALTERNATE INDEXES

When alternate indexes are used in the COBOL program, the user must specify not only a DD statement for the base cluster, but also one DD statement for each alternate path. The ddname for the base cluster is the one declared in the COBOL program. However, no language mechanism exists to explicitly declare ddnames for alternate paths in the program. Therefore, the following convention has been established and must be adhered to by the user.

The ddname for each alternate path is to be formed by concatenating its base cluster ddname with an integer--beginning with 1 for the path associated with the first alternate record defined for that file in the COBOL program, and being incremented by 1 for each path associated with each successive alternate record definition for that file. For example, if a base cluster's ddname were ABCD, then the ddname for the alternate path of the first alternate record key defined would have to be ABCD1. The ddname for the alternate path of the second alternate record key defined would have to be ABCD2, and so on.

If the combination of base cluster ddname and sequence number exceeds eight characters, the base cluster portion of the ddname must be truncated at the right to reduce the concatenated result to eight characters. For example, if a base cluster's ddname is ABCDEFGH, then the first alternate path's ddname should be ABCDEFG1, the tenth should be ABCDEF10, and so forth.

The following example shows the connection between a program using two alternate indexes and the required DD statements. The base cluster is named XYZ, and the first alternate index' pathname is PATHONE and the other's PATHTWO.

```
//ABCD   DD  DSN=XYZ,DISP=OLD
//ABCD1  DD  DSN=PATHONE,DISP=OLD
//ABCD2  DD  DSN=PATHTWO,DISP=OLD

FILE-CONTROL.
   SELECT filename ASSIGN TO ABCD
   RECORD KEY IS whatever
   ALTERNATE RECORD KEY IS CITY
   ALTERNATE RECORD KEY IS PRICE
```

The key CITY relates to the alternate index whose pathname is PATHONE, and the key PRICE relates to the alternate index whose pathname is PATHTWO.

DD STATEMENT FOR A USER CATALOG

If a data set in a job step is defined in a user catalog, it is also necessary to identify the user catalog by means of either a JOBCAT (Example 1) or STEPCAT (Example 2) DD statement.

Example 1:

```
//EX1      JOB    ...
//JOBCAT   DD     DSNAME=usercatalogname,
                  DISP=SHR
//         EXEC   ...
```

Example 2:

```
//EX2      JOB    ...
//         EXEC   ...
//STEPCAT DD      DSNAME=usercatalogname,
                  DISP=SHR
```

DD PARAMETERS USED WITH VSAM

Although the operating system does not disallow OS/VS DD parameters and subparameters that do not apply to a VSAM data set, the COBOL programmer should be

aware that some of the DD parameters and subparameters have certain additional meanings when used with VSAM. For complete information on the meanings of the OS/VS DD parameters and subparameters, as well as the potential problems which exist if care is not taken, see OS/VS Virtual Storage Access Method (VSAM) Programmer's Guide.

VSAM-ONLY JCL PARAMETERS

VSAM has one JCL parameter of its own: AMP. AMP, and its associated subparameters, is used mainly in conjunction with specifications made through Access Method Services. The AMP parameter takes effect when the data set defined by the DD statement is opened. For details on the use and specification of AMP, see the VSAM Programmer's Guide.

CONVERTING NON-VSAM FILES TO VSAM FILES

Both SAM and ISAM data sets can be converted to VSAM data sets so that they may be processed by a COBOL program using VSAM. The conversion is done through Access Method Services.

Essentially, the conversion process consists of defining a VSAM data set as the target for the data set being converted. Then through the appropriate JCL and the REPRO command, the conversion is accomplished.

For a complete description of the conversion process, see OS/VS VSAM Access Method Services.

USING COBOL ISAM PROGRAMS WITH VSAM FILES

Existing COBOL programs written to process ISAM files can be used to process VSAM files by going through VSAM's ISAM interface. To do this, the programmer need only make some JCL changes in the COBOL ISAM program.

The EXEC card should specify the desired processing program, as usual; the DD card should be changed to a VSAM DD card as described above under "Job Control Language for VSAM File Processing."

Certain AMP subparameters might be used for running an ISAM processing program with the ISAM interface. For complete details on the conversion process, see the VSAM Programmer's Guide.

VSAM FEATURES NOT AVAILABLE THROUGH COBOL

Not all of VSAM's facilities can be used directly through a COBOL program. These unavailable features include:

- Alternate indexing for ESDS
- Multiple string processing
- Skip-sequential processing (key-ordered sequential/direct)
- Addressed-direct processing
- Control-interval (low-level) processing
- Journaling support
- Alternate index as end-use object (as base cluster instead of path) processing
- GET-previous processing
- Asynchronous processing

  It is possible to open a VSAM data set concurrently under two separate FDs in a COBOL program if the assign clauses of the SELECT statements refer to the same DDNAME. When the ACB is generated, COBOL takes the GENCB default of 'DDN', which indicates DDNAME sharing to VSAM. In such a case, VSAM will share the same buffers and control blocks for the two ACBs, and data integrity is preserved.

  If the user program attempts to access records within the same control interval by using the two separate file definitions, lock-out may occur. If the program contains two SELECT statements for the file but uses different DDNAMEs, the file can still be opened. However, data set names are not shared, and VSAM will use two completely separate buffer pools and control block structures. In this case, the integrity of the data set is not preserved, and updates to the file may be lost.

Note. Even when the spanned format is used, the COBOL restriction on the length of logical records must be adhered to (that is, a maximum length of 32,767 characters).

The lister feature of the IBM OS/VS COBOL Compiler can be used by the COBOL programmer to produce a COBOL source listing that is reformatted and cross-referenced to increase intelligibility and conserve space. Optionally, a reformatted source deck can be produced. The lister output can be produced either with or without compilation occurring.

OPERATION OF THE LISTER FEATURE

The lister accepts source programs written in OS/VS COBOL, analyzes the source statements, and produces the reformatted and cross-reference source program. The output is either in the form of a listing or as a listing and a punched deck.

This reformatted source output follows indenting conventions imposed by the lister to increase readability, and contains cross-references between data items and Procedure Division statements, between PERFORM statements and paragraph names, etc. Optionally, the lister produces a new source deck that matches the output listing except that cross-reference information is omitted.

Thus the lister can be used to process source decks for uniformity of indenting and for highlighting of statements such as IF, GOTO, etc., or it can be used to obtain a cross-referenced source listing as permanent documentation of a production program, or for use as an aid in program analysis and debugging. Various options permit printing the Procedure Division listing in two columns to conserve space and the inclusion of BASIS and COPY statements.

Notes: Lister ignores the carriage control statements SKIP and EJECT. When LISTER is in effect, the NUM option has no meaning.

PROGRAMMING CONSIDERATIONS

The lister is designed to operate most efficiently on syntactically correct COBOL source, and does not have the expanded error handling of the full compiler. It is therefore highly recommended that the user programs first be compiled using the SYNTAX option, and syntax errors corrected before invoking the lister feature. If the lister function is used and there are syntactical errors, lister processing will be terminated. The syntax checking in the lister feature is different from the checking done by the standard compiler. Syntax checking is usually more stringent in the lister than in the compiler. Some syntax errors that are recognized but corrected by the standard compiler may be flagged as errors when using the lister.

The listing produced by LISTER will be reformatted for that portion of the program that was syntactically correct. If LSTCOMP was specified, the SOURCE option will be forced on.

Further notes: Since lister reformats the user's COBOL program, compilation of the program, if LSTCOMP is in effect, will be different from a non-lister compilation of the same program. For example:

1. Lister sequence numbers may be different.
2. SKIP/EJECT cards will have no functional value with lister.
3. BASIS card will be dropped from the lister listings.
4. FIPS messages will be based on the reformatted lister listings.
5. Suppress option of COPY will have no effect.
6. Sequence checking will not take place for a lister sum.
7. Source statements copied from a user-created library as a result of a COPY statement are not reformatted. However, statements which begin in columns 8-11 will be indented to column 8 in the lister output, and those which begin in columns 12-72 will be indented to column 12.
8. Lister terminates upon detecting a syntax error in the COBOL source program. When such an error is detected, lister issues an error flag to signal that the following source cards are to be passed on without processing. Lister then treats the balance of the program as comment cards.

In addition to the condition mentioned above, unusual termination of lister can occur if the source program contains:

- Too many (approximately 80 or more) consecutive *-comments cards.
- Too many (approximately 100 or more) consecutive blank cards.
- Too many (approximately 100 or more) consecutive cards for a single data item.

If one of the above three conditions occurs, the file written on SYSUT2 is incomplete.

THE LISTING

The reformatted output listing is divided into four parts:

1. **A one-page introduction which summarizes briefly lister codes, conventions, uses.**
2. **The Identification and Environment Divisions.**
3. **Detailed, cross-referenced, reformatted Data and Procedure Divisions.**
4. **The Summary listing.**

These (except the introduction) are described briefly below, and in greater detail in subsequent sections.

THE OUTPUT DECK

The deck produced optionally by the lister may be saved either in card form or in a COPY/BASIS library. This output reflects the reformatted source program. The output deck is described in detail later in this chapter.

REFORMATTING OF IDENTIFICATION AND ENVIRONMENT DIVISIONS

The lister reformats the Identification Division statements only by imposing indenting conventions. Statements are indented two positions, and continuations, if any, are indented six additional positions.

Environment Division statements are reformatted by imposing indenting conventions and by appending cross-reference information to SELECT statements in the FILE CONTROL section. Thus, in reading the FILE CONTROL section, there are direct references to the file description statements in the Data Division.

Figure 81 is an example of reformatted Identification and Environment Divisions. (The note shown is not produced by the lister program.)

DATA DIVISION REFORMATTING

The lister reformats the Data Division principally by imposing indenting conventions. In addition, it aligns PICTURE, VALUE and other clauses vertically to improve readability and facilitate visual checking. Such clauses as REDEFINES and OCCURS are highlighted as a result of the alignment. All indenting is with respect to the left margin, which contans the statement number.

FDs and level-77 items are indented zero, level-01 items are indented two and level-02 items are indented four. Level 03 and lower are each indented two from the last higher level item, up to seven levels of indentation. Use of this convention, makes the overall structure of each file and group data item immediately apparent to the reader of the listing.

The most striking change in the appearance for the Data Division listing is

```
 1 IDENTIFICATION DIVISION.
 2   PROGRAM-ID. TESTRUN.
 3   AUTHOR. PROGRAMMER NAME.
 4   INSTALLATION. NEW YORK PROGRAMMING CENTER.
 5   DATE-WRITTEN. JULY 12, 1968.
 6   DATE-COMPILED. AUG 20,1976.
 7   REMARKS. THIS PROGRAM HAS BEEN WRITTEN AS A SAMPLE PROGRAM FOR
           COBOL USERS. IT CREATES AN OUTPUT FILE AND READS IT BACK
           AS INPUT.
10 ENVIRONMENT DIVISION.
11 CONFIGURATION SECTION.
12   SOURCE-COMPUTER. IBM-370-168.
13   OBJECT-COMPUTER. IBM-370-168.
14 INPUT-OUTPUT SECTION.
15   FILE-CONTROL.
16     SELECT FILE-1                                          21 )
           ASSIGN TO UT-2400-S-SAMPLE.                        29 ) (1)
17     SELECT FILE-2
           ASSIGN TO UT-2400-S-SAMPLE.
```

Note:

(1) Refers to FD statement numbers in the Data Division.

Figure 81. Sample Identification and Environment Division Output Listing

the addition, at the right of each statement, of cross-references that identify the statement number of each Data Division or Procedure Division statement that redefines, changes, reads, tests, or otherwise refers to the data item. When the number of such references is too great to fit on the line, the lister prints as many as there are room for, on the line, and prints the remainder as a footnote at the bottom of the page.

The eight codes used in the Data Division are:

C Data item changed (such as by ADD or MOVE)

E Data item referred to by Environment Division statement (SELECT) or by some Procedure Division input/output operation (OPEN, CLOSE, INITIATE, etc.)

D Data item changed by REDEFINE or RENAME

Q Queried by IF, WHEN, or UNTIL

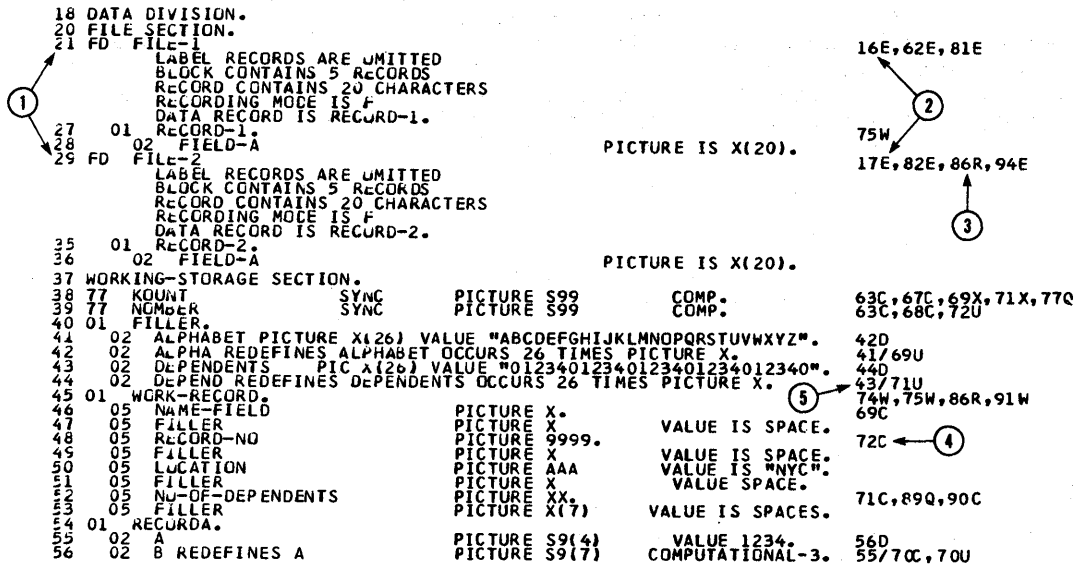
R Referred to by a READ, ACCEPT, or similar statement

U Data item unchanged (used as a source field)

W Referred to by a WRITE, GENERATE, DISPLAY, or similar statement

X Used as an index, subscript, or object of a DEPENDING ON statement

Use of these codes is depicted in Figure 82, which is an example of a reformatted Data Division. (The notes shown in the figure are not produced by the lister.)



Notes:

(1) FD referred to by SELECT statement in Environment Division.

(2) Associated SELECT statement; E denotes Environment Division reference (or OPEN/CLOSE from Procedure Division).

(3) Procedure Division statement; R denotes that this statement reads this file.

(4) Procedure Division statement; C denotes that this statement changes this data item.

(5) 43/ indicates that DEPENDENTS is defined in statement 43.
The 44D following statement 43 indicates the same relationship between these two statements.

Figure 82. Sample Data Division Output Listing

PROCEDURE DIVISION REFORMATTING

The lister reformats the Procedure Division by applying indenting conventions to nested IFs, GOTOs, etc., and by appending cross-references to sections and paragraphs to indicate that the statement is arrived at from either a GO TO or a PERFORM. It also appends references to the Data Division so that the data item being acted on can be found quickly. The five codes used in the Procedure Division are:

A	ALTER
B	(ALTER) to PROCEED TO
G	GO TO
P	PERFORM
T	(PERFORM) THRU

Use of the codes G, P, and T is depicted in Figure 83. The A and B in Figure 83 are examples of lister's footnoting of elementary 01- and 77-level data items (not of ALTER and PROCEED TO). If additional such data items were present, they would be identified by footnotes lettered C, D, E, and so forth.

```
 57 PROCEDURE DIVISION.

 58   BEGIN.
 59     NOTE THAT THE FOLLOWING OPENS THE OUTPUT FILE TO BE
            CREATED AND INITIALIZES COUNTERS.

 61   STEP-1.
 62     OPEN OUTPUT FILE-1.                                          21
 63     MOVE ZERO TO KOUNT NOMBER.                                  A,B
 64     NOTE THAT THE FOLLOWING CREATES INTERNALLY THE RECORDS
            TO BE CONTAINED IN THE FILE, WRITES THEM ON TAPE,
            AND DISPLAYS THEM ON THE CONSOLE.

 66   STEP-2. 77P
 67     ADD 1 TO KOUNT.                                               A
 68     ADD 1 TO NOMBER.                                              B
 69     MOVE ALPHA (KOUNT) TO NAME-FIELD.                      42,A,46
 70     COMPUTE B = B + 1.                                      56,56
 71     MOVE DEPEND (KOUNT) TO NO-OF-DEPENDENTS.               44,A,52
 72     MOVE NUMBER TO RECORD-NO.                                B,48

 73   STEP-3. 77T
 74     DISPLAY WORK-RECORD UPON CONSOLE.                            45
 75     WRITE RECORD-1 FROM WORK-RECORD.                      27,45

 76   STEP-4.
 77     PERFORM STEP-2 THRU STEP-3 UNTIL KOUNT IS EQUAL  66,73,A
            TO 26.
 78     NOTE THAT THE FOLLOWING CLOSES OUTPUT AND REOPENS IT AS
            INPUT.

 80   STEP-5.
 81     CLOSE FILE-1.                                                21
 82     OPEN INPUT FILE-2.                                           29
 83     NOTE THAT THE FOLLOWING READS BACK THE FILE AND SINGLES
            OUT EMPLOYEES WITH NO DEPENDENTS.

 85   STEP-6. 92G
 86     READ FILE-2 RECORD INTO WORK-RECORD AT END           29,45
 87       GO TO STEP-8.                                             93

 88   STEP-7.
 89     IF NO-OF-DEPENDENTS IS EQUAL TO "0"                          52
 90       MOVE "Z" TO NO-OF-DEPENDENTS.                              52
 91     EXHIBIT NAMED WORK-RECORD.                                   45
 92     GO TO STEP-6.                                                85

 93   STEP-8. 87G
 94     CLOSE FILE-2.                                                29
 95     STOP RUN.

A-38 77 KOUNT              SYNC      PICTURE S99        COMP.        63C,67C,69X,71X,77Q
B-39 77 NOMBER             SYNC      PICTURE S99        COMP.        63C,68C,72U

 96 -- COBOL SPECIAL-REGISTERS --
 97      TALLY
 98      PRINT-SWITCH
 99      MORE-LABELS
100      RETURN-CODE
101      LABEL-RETURN
102      SORT-RETURN
103      SORT-CORE-SIZE
104      SORT-FILE-SIZE
105      SORT-MODE-SIZE
106      SORT-MESSAGE
107      CURRENT-DATE
108      TIME-OF-DAY
109      WHEN-COMPILED
110      DEBUG-CONTENTS
111      DEBUG-ITEM
112      DEBUG-LINE
113      DEBUG-NAME
114      DEBUG-SUB-1
115      DEBUG-SUB-2
116      DEBUG-SUB-3
```

Notes:

(1) Lister footnotes all occurrences of 77-level and elementary 01-level data items.

(2) Statement 66 is arrived at through PERFORM in statement 77.

(3) Statement 77 contains the PERFORM of statement 66.

(4) Statement 85 is arrived at through GO TO in statement 92.

Figure 83. Sample Procedure Division Output Listing

SUMMARY LISTING

The summary listing provides an overall view of the relationship among FDs, RDs, SDs, and SECTIONS. The entry for each of these major parts of the program consists of a title line showing the statement number and the name of the file, record, or section and a series of counts (by reference type) for each of the categories "from" and "to." Intra references are also shown for WORKING-STORAGE and PROCEDURE; these are references within the section, file, or record, such as REDEFINES and PERFORM operations. "From" are the references from other parts of the program to this part and "to" are the references to other parts from this part. The other parts are identified by the numbers of their first statements; these numbers appear in the column just to the right of the words INTRA, FROM, and TO.

Figure 84 is an example of a summary listing.

```
  1 IDENTIFICATION DIVISION.
 10 ENVIRONMENT DIVISION.
 11 CONFIGURATION SECTION.
 14 INPUT-OUTPUT SECTION.
TO     21      1 E-1 }
       29      1 E-1 } (1)
 18 DATA DIVISION.
 20 FILE SECTION.
 21 FILE-1
FROM   14      1 E-1
       57      3 E-2,W-1 } (2)
 29 FILE-2
FROM   14      1 E-1
       57      3 E-2,R-1
 37 WORKING-STORAGE SECTION.
INTRA          3 D-3
FROM   57     21 C-9,Q-2,R-1,U-4,W-3,X-2
 57 PROCEDURE DIVISION.
INTRA          4 G-2,P-1,T-1 <---(3)
TO     21      3 E-2,W-1
       29      3 E-2,R-1
       37     21 C-9,G-2,R-1,U-4,W-3,X-2
 96 -- COBOL SPECIAL-REGISTERS --

  55 A
  42 ALPHA
  41 ALPHABET
  56 B
  58 BEGIN
 107 CURRENT-DATE
 110 DEBUG-CONTENTS
 111 DEBUG-ITEM
 112 DEBUG-LINE
 113 DEBUG-NAME
 114 DEBUG-SUB-1
 115 DEBUG-SUB-2
 116 DEBUG-SUB-3
  44 DEPEND
  43 DEPENDENTS
  28 FIELD-A
  36 FIELD-A
  21 FILE-1
  29 FILE-2
  38 KOUNT
 101 LABEL-RETURN
  50 LOCATION
  99 MORE-LABELS
  46 NAME-FIELD
  52 NO-OF-DEPENDENTS
  39 NUMBER
  98 PRINT-SWITCH
  48 RECORD-NO
  27 RECORD-1
  35 RECORD-2
  34 RECORDA
 100 RETURN-CODE
 103 SORT-CORE-SIZE
 104 SORT-FILE-SIZE
 106 SORT-MESSAGE
 105 SORT-MODE-SIZE
 102 SORT-RETURN
  61 STEP-1
  66 STEP-2
  73 STEP-3
  76 STEP-4
  80 STEP-5
  65 STEP-6
  68 STEP-7
  93 STEP-8
  97 TALLY
 108 TIME-OF-DAY
 109 WHEN-COMPILED
  45 WORK-RECORD
```

Notes:

(1) The INPUT-OUTPUT section contains references to one data item in FILE-1 and one data item in FILE-2.

(2) FILE-1 is referenced once from the INPUT-OUTPUT section and three times from the Procedure Division.

(3) The Procedure Division contains four intra references.

Figure 84. Sample Summary Listing

THE SOURCE LISTING

The source listing of the Identification, Environment, and Data Divisions may be considered as having three "columns." The leftmost contains a statement number, or is blank if the line is either a comment or a continuation of the preceding statement or line. The second contains the reformatted COBOL statements. The third (not present as an independent column in the Procedure Division) contains references to or from other statements in the source program. Thus, each line of the output listing contains a numbered source statement or its unnumbered continuation, and a reference or series of references to all other statements in the source program that refer to it. If the series of references is too long to fit on the line, the lister prints as many as will fit, followed by a letter indicating a footnote. The footnote contains the remainder of the references.

The source listing of the Procedure Division is normally printed in double-column format, with each column divided as described above. This format approximately doubles the span of logic that can be seen on one page or one facing-page spread.

Regardless of whether the source code follows indentation conventions, the lister indents statements according to their type, and according to hierarchy where applicable. This feature of the lister makes file and record structure immediately visible and also helps to identify groups of related statements such as IF/ELSE and nesting of IFs.

Note: If blank lines are present in the original source, the lister eliminates them and renumbers the statements accordingly.

Format Conventions

New statements are indented from the left margin, which contains the statement number. The lister treats the following as new statements:

- Division and Section headers
- Paragraph names
- Level numbers or level indicators (FD, RD, etc.)
- Verbs
- ELSE/OTHERWISE
- AT END (only when following SEARCH).

Indentation of the new statement is made according to the following rules:

1. Data Division
   - FDs and level 77 items are indented zero.
   - Level 01 items are indented zero in the Linkage and Working Storage sections and two in the File and Report sections
   - Each subsequent lower level within an 01 item is indented two more than the preceding higher level, up to a maximum of 14 character positions or 7 levels.
2. Procedure Division
   - Section names are indented zero
   - Paragraph names are indented two
   - Unconditionally executed verbs are indented four
   - Verbs executed under a single condition, such as IF or AT END, are indented six
   - The first IF in a nest is indented four, subsequent nested IFs are indented an additional two at each level up to a maximum of 14 character positions or 6 levels.
   - ELSEs are indented to the same position as the IF to which they correspond.
3. Continuation lines in all divisions are indented six with respect to the first line of the continued statement.

Word spacing within a statement and on continuation lines is usually one space. Within the Data Division, however, PICTURE and VALUE are aligned as nearly as possible into columns so that they may be found and compared easily. Words are never split at the end of a line unless the word to be split is a nonnumeric literal that will not fit on a single continuation line.

References appear to the right of the statement or continuation line. References following paragraph or section names appear immediately to the right of the name, separated by a blank. References following other types of statements appear as far to the right as possible depending on the number of blanks available on the line. Each reference consists of a statement number and a type indicator.

When references are in series, they are separated by commas and are in ascending order.

Within the Data Division, a reference series may end with an alphabetic footnote indicator. The footnote contains the remaining references to that data item.

In the Procedure Division, the reference may also be a footnote indicator, but the footnote is different in appearance. In the Procedure Division-, the footnote is actually an on-page replica of the Data Division statement referred to by the footnoted statement. This replica is complete with all other references to the data item from other portions of the program. To conserve space in the listing, the lister does not repeat a footnote if it appears at the bottom of either of the two preceding pages but instead reuses the same footnote letter in the new reference.

Type Indicators

As mentioned above, a reference consists of a statement number and a type indicator. The type indicator provides immediate information as to what is being done by the statement referred to.

Two sets of type indicators are used by the lister: one for the Data Division and one for the Procedure Division. Within the Data Division, the type indicators are:

- C Data item changed (such as by ADD or MOVE)
- D Data item REDEFINED or RENAMED
- E Data item referred to by Environment Division statement (SELECT) or by some Procedure Division input/output operation (OPEN, CLOSE, INITIATE, etc.)
- Q Queried by IF, WHEN, or UNTIL
- R Referred to by a READ, ACCEPT or similar statement
- U Data item unchanged (used as a source field)
- W Referred to by a WRITE, GENERATE, DISPLAY, or similar statement
- X Used as an index, subscript, or object of a DEPENDING ON statement

Within the Procedure Division, the type indicators are:

- A ALTER
- B (ALTER) TO PROCEED TO
- E INPUT or OUTPUT PROCEDURE (SORT Feature)
- G GO TO
- P PERFORM
- T (PERFORM) THRU

THE SUMMARY LISTING

The summary listing is useful both as an analysis and as a troubleshooting aid. Using the summary listing, the user can ascertain quickly which data areas are most referred to, which procedures refer to them most often, and the nature of those references. The number of references to undefined symbols and the number of incorrectly coded COBOL statements can also be ascertained.

General Appearance

Each division or section header, and each FD, RD, or SD begins a new entry in the summary listing. The entry consists of the header line, and beginning on the next line, the total number of each kind of reference to that section from within itself (INTRA), and from outside itself (FROM). These are followed by similar information for references the section makes to others outside itself (TO). The type indicators used for references are the same as those used in the source listing.

In large programs, with either no sections or very large sections in the Procedure Division, the lister summary may not be very helpful. This can be remedied by adding SECTION statements to the source program at appropriate points. If SECTION statements are being added to a program that already contains some, it is very important to make certain that both implicit and explicit reference qualifiers are not invalidated.

THE OUTPUT DECK

Optionally, the lister can produce a new COBOL source deck that reflects the reformatted source listing. This deck may be saved in a BASIS library, used directly as input to the compiler, or punched into cards. As a result of reformatting, the new deck may contain more cards than the original, but the difference is not great enough to cause any appreciable increase in compilation time. The output deck differs from the listing as follows:

1. References, footnotes, and blank lines are omitted.

2. Literals will be repositioned if needed to assure proper continuation.

3. Statement numbers are converted to card numbers.

   a. The statement number is multiplied by 10, and leading zeros added as necessary to fill columns 1 through 6.

   b. Comment and continuation cards are numbered one higher than the preceding card.

SPECIFYING THE LISTER

The lister feature is specified in the PARM field of the EXEC card through five compiler options. The combination of options that are selected determine both the format and contents of the lister output. Either LSTONLY or LSTCOMP must be specified for the other options to have meaning, unless BATCH is specified. In a batch compilation, if some or all of the programs are to be compiled using the lister feature, L120 or L132 must be specified in the PARM field of the EXEC card--even if LSTCOMP or LSTONLY are specified on the CBL card.

The five lister options are described below. Note that the IBM-supplied defaults are indicated by an underscore; they can be changed when the compiler is installed. The lister options are as follows:

LSTONLY
LSTCOMP
NOLST
: indicates whether the lister feature is to be used. LSTONLY specifies that a reformatted listing is to be produced but that no compilation is to occur. LSTCOMP specifies that both a reformatted listing is to be produced and compilation is to occur in the same job step.

FDECK
NOFDECK
: indicates whether a copy of the reformatted source program is to be written on the SYSPUNCH data set. Since FDECK has meaning only with either LSTONLY or LSTCOMP, the lister output will be both a reformatted listing and a reformatted deck. COPY statements within the source program will be produced as COPY statements, or, if CDECK is in effect, the expansion of the COPY statement will be produced.

CDECK
NOCDECK
: indicates whether or not COPY statements are to be converted to comment statements in the output listing and the COPY members are to be expanded. CDECK may be specified with FDECK or NOFDECK. With FDECK, the source deck produced will contain the expansion of COPY statements; with NOFDECK, only the expansions of COPY statements are produced.

LCOL1
LCOL2
: indicates whether the Procedure Division part of the listing is to be in single-or double- column format.

L120
L132
: indicates whether the length of each line of the reformatted listing is to be 120 or 132 characters long.

A programmer using the IBM OS/VS COBOL Compiler under the IBM Operating System, has several methods available to him for testing and debugging his programs. Use of the symbolic debugging features are described in detail in this chapter.

"Appendix A: A Sample Program" contains an example of a program run without the symbolic debugging features. The chapter "Program Checkout" contains information useful for finding the instruction that causes the abnormal termination and then correcting the problem. The chapters "Output" and "Using the Checkpoint/Restart Feature" include a discussion of compiler output and a description of taking checkpoints and restarting programs, respectively.

Note: The program product IBM OS COBOL Interactive Debug (Program Number 5734-CB4) enables the user to debug his COBOL programs from a TSO terminal. To be acceptable for Interactive Debug, a program must be compiled with the TEST compiler option. TEST overrides FLOW, STATE, SYMDMP and COUNT. However, note that TEST may not be specified with BATCH, since BATCH overrides TEST. TEST will also be cancelled if the program contains USE FOR DEBUGGING statements. Interactive Debug is described in greater detail in the "Program Checkout" part of this publication.

USE OF THE SYMBOLIC DEBUGGING FEATURES

As an aid to debugging, compiler options can be requested that provide additional diagnostic information for an abnormal termination other than one caused by "Canceled by Operator" or exceeding the system-state time slice. Three user options are available for object-time debugging -- the statement number option (STATE), the flow trace option (FLOW), and the symbolic dump option (SYMDMP).

The STATE option causes the number of the card for the last verb executed before termination to be printed out. The FLOW option causes a trace of the last user-specified number of procedures executed to be printed out (with a default of 99). Both STATE and FLOW cause the PROGRAM-ID, the completion code, and the last problem PSW to be printed out. The SYMDMP option enables the user to request a symbolic formatted dump of the data area of the object program for an abnormal termination, or to request dynamic dumps of data areas at strategic points during execution.

Use of these features requires no source language coding; rather the user specifies these options at compile time, through job control language. Operation of the SYMDMP option is dependent on execution-time control cards. Figure 86 illustrates the output generated for each of these features.

When any of the debugging options is specified, the programmer must:

- Request the option at compile time by specifying it in the PARM field or, if a cataloged procedure is used, in the PARM.COB field.
- Include a //SYSDBOUT DD card for the debug output data set at execution time.
- Make the COBOL library available at execution time by specifying the following DD statement:

```
//STEPLIB DD DSN=subr-libname,DISP=SHR,
                VOL=SER=volser,UNIT=unit
```

  (This is necessary because certain COBOL library subroutines are loaded dynamically from the subroutine library only as needed; they are not link-edited into the COBOL object program.)

- If the COBOL program being debugged is to be invoked from a higher-level non-COBOL program, the programmer must ensure that the non-COBOL program calls the COBOL library subroutine ILBOSTP0 before calling any COBOL program. For further information on this point, see the section "Calling and Called Programs" in this manual.

STATE Option

If the STATE option is in effect and an abnormal termination occurs, the printed output includes the compiler-generated card number or, if NUM is in effect, the card sequence number for the last verb executed. Violation of the rule against mixing RES and NORES programs in a single run unit may result in erroneous information from STATE.

FLOW Option

If the FLOW option is in effect, a formatted list containing the PROGRAM-ID and either the compiler-generated card number or the line number (if NUM is in effect) of the last n executed procedures is printed on SYSDBOUT. The number of procedures traced can vary from 1 to 99 and is specified by the programmer.

The number of procedures to be traced may be specified at compile time via either the PARM parameter or, if a cataloged procedure is used, the PARM.COB field. This number may be overridden at execution time via the PARM parameter or, if a cataloged procedure is used, the PARM.GO parameter. If the number of procedures traced is specified at neither compile time nor execution time, either the default value of 99 or the value specified at program product installation will be employed. When using FLOW or NOFLOW at execution time, the option must be preceded by a slash "/". (See Figure 8 for an example.)

If batch compilation is used, FLOW can be specified at compile time and remain in effect for every program in the batch. To suppress a trace for a particular program within the batch, the programmer should specify NOFLOW at execution time as the last parameter in the PARM field for that program, or change the CBL card. For more information, see the sections "Options for the Compiler" and "Options for Execution."

Note: The FLOW option is completely independent of the READY/RESET TRACE feature of the debugging language.

SYMDMP Option

If the SYMDMP option is in effect, a symbolic formatted dump of the object program's data area is produced when the program abnormally terminates. (The SYMDMP option cannot be used if the source program contains USE FOR DEBUGGING and WITH DEBUGGING MODE.) This option also enables the programmer to request dynamic dumps of specified data-names at strategic points during program execution. If two or more COBOL programs are link-edited together and one of them terminates abnormally, a formatted dump is produced for all programs in the calling sequence compiled with the SYMDMP option, up to and including the main program in the reverse order of their calling sequence. (The terminating program itself need not have been compiled with the SYMDMP option.)

By specifying a //SYSDTERM DD card in addition to the //SYSDBOUT DD card, dynamic dump output will be written onto SYSDTERM while the abend dump output will go to SYSDBOUT. SYMDMP output will be formatted at 55 lines to the page.

Note: The TSO programmer should assign SYSDTERM to the terminal since dynamic dump output is interruptable. SYSDBOUT should be assigned to a direct access data set which could be listed at the terminal after the ABEND is complete.

The abnormal termination dump consists of the following parts:

1. An abnormal termination message, including the number of the statement and of the verb being executed at the time of an abnormal termination.
2. Selected areas in the Task Global Table.
3. A formatted dump of the Data Division including:
   (a) For an SD -- the card number, the sort-file-name, the type, and the sort record.
   (b) For an FD -- for VSAM: OPEN/CLOSE status, card number, organization, access mode, last I/O operation, file status, and the fields of the record. For non-VSAM: the card number, the file-name, the type, the ddname, the DECB and/or DCB status, the contents of the DECB and/or DCB in hexadecimal, and the fields of the record; also, for QSAM, the file status.
   (c) For an RD -- the card number, the report-name, the type, the report line, and the contents of PAGE-COUNTER and LINE-COUNTER if present.
   (d) For a CD -- the CD itself in its implicit format, as well as the area containing the message data currently being buffered.
   (e) For an index name -- the name, the type, and the contents in decimal which represents an actual displacement from the beginning of the table that corresponds to an occurrence number in the table. The value is calculated as the occurrence number minus one, multiplied by the length of the

entry that is indexed by this index-name.

The symbolic dump option is requested at compile time via the SYMDMP option, through the PARM parameter of the EXEC card. Operation of the symbolic dump option is dependent on object-time control cards placed in the SYSDBG data set (see also the "Default SYSDBG Data Set" section that follows). This data set must consist of unblocked 80-byte records. If the object-time control cards are not present, SYMDMP is canceled at execution time. These cards are discussed below.

Object-Time Control Cards

The operation of the SYMDMP option is determined by two types of control cards:

Program-control card -- required if abnormal termination and/or dynamic dumps are requested.

Line-control card -- required only if dynamic dumps are requested.

Syntax Rules: The fields of both the program-control card and the line-control card must conform to the following rules:

1. Control cards are essentially free form, i.e., parameters coded on these cards can start in any column. However, parameters may not extend beyond column 71.
2. Each parameter except the last must be immediately followed by a comma or a blank.
3. No commas are needed to account for optional parameters that are not specified.
4. All upper-case letters in IBM documentation represent specifications that are to appear in the actual statement exactly as shown.
5. All lower-case letters represent generic terms that are to be replaced in the actual statement.
6. Brackets are used to indicate that a specification is optional and is not always required in the statement.
7. Brackets enclosing stacked items indicate that a choice of one item may, but need not, be made by the programmer.
8. Braces enclosing stacked items indicate that a choice of one item must be made by the programmer.
9. All punctuation marks and special characters shown in the statement formats other than hyphens, brackets, braces, and underscores, must be punched exactly as shown. This includes commas, parentheses, and the equal sign.

   Note: Blanks may be substituted for commas.

Continuation Cards: To continue either the program-control card or the line-control card, the programmer must code a nonblank character in column 72 of the continued card. Individual keywords and data-names cannot be split between cards.

Control Statement Placement: If a main program is compiled with the SYMDMP option, or if at least one subprogram called by the main program is a COBOL program compiled with the SYMDMP option, the control cards may either follow or precede the programmer's data, if any, in the input stream:

```
//GO          EXEC   PGM=
//GO.SYSDBG   DD     *

  {user's control cards}

/*
//GO.SYSIN    DD     *

  {user's data cards, if any}

/*
```

For an example of the control statements used to compile a program with the SYMDMP option, see Figure 86.

Program-Control Cards: A program-control card must be present at execution time for any program requesting a SYMDMP service. Program-control cards have the following format:

```
                                  [,ENTRY  ] [,(HEX)  ]
program-id,ddname                 [,NOENTRY] [,(NOHEX)] [,PDS]
```

where:

program-id
: is a 1- to 8-character program-name of a COBOL program compiled with the SYMDMP option. This parameter is required and must appear first on the program-control card.

ddname
: is the execution-time ddname of the file that was produced at compile time on

SYSUT5. This parameter is required and must follow the program-id.

ENTRY
NOENTRY
ENTRY is used to provide a trace of a program-name when several programs are link-edited together. Each time the program whose PROGRAM-ID matches the "program-id" parameter is entered, its name is displayed.

HEX
NOHEX
is optional and refers to the format of the Data Division area in the abnormal termination dump. If HEX is specified, level-01 items are provided in hexadecimal. Items subordinate to level-01 items are printed in EBCDIC, if possible. Level-77 items are provided both in EBCDIC and hexadecimal. If HEX is not specified, items subordinate to level-01 items and level-77 items are provided in EBCDIC. If these items are unprintable, hexadecimal notation is provided.

PDS
is optional and allows the user to specify that the debug file, which was produced at compile time on SYSUT5 and whose name is ddname, is a partitioned data set. In this case, SYMDMP assumes that program-id is the name of a member in that debug file. This option is intended to reduce the number of execution-time DD cards required for debug files, when many programs compiled with the SYMDMP option are executed together in a COBOL run-unit. Since each such program requires a unique debug file, each program-control card could contain a unique program-id (member), the same ddname, and PDS.

Note: The user should be aware that the debug file produced at compile time contains device-dependent relative block addresses embedded in the data blocks and is, therefore, unmovable. The only way to alter a member in an existing partitioned debug file is through recompilation replacement. User attempts to compress the data set through IEBCOPY or move the data set to another data set through IEHMOVE will be rejected by these utilities. Further, the user should not create a partitioned data set from several compiler-created sequential debug files. SYMDMP will produce message IKF164I and will cancel debug output for any program whose debug file has been moved. The only exception is that a sequential debug file can be moved to another sequential data set on a device of the same type.

Line-Control Cards: Line-control cards have the following format:

```
line-num[,(verb-num)][,ON n[,m[,k]]]

{ [,(HEX)  ]                                  }
{ [,(NOHEX)] [,ALL]                           }
{ [,(HEX)  ]                                  }
{ [,(NOHEX)] ,name1 [THRU name2]...           }
```

line-num
indicates the card number associated with the point in the Procedure Division at which the dynamic dump is to be taken. The card number is either the compiler-generated number or, if NUM is in effect, the user's number in card columns 1 through 6. The number must be that of a card containing a section name, procedure name, conditional verb, or imperative verb.

verb-num
indicates the position of the verb in the card indicated by "line-num" before whose execution a dynamic dump is taken. When "verb-num" is not specified, the value 1 is assumed; when specified, "verb-num" must follow "line-num" and may not exceed 15.

ON n[,m[,k]]
is equivalent to the COBOL statement ON n AND EVERY m UNTIL k... This option limits the requested dynamic dumps to specified times. For example, "ON n" would result in one dump, given the nth time "line-num" is reached during execution. "ON n,m" would result in a dump the first time at the nth execution of "line-num" and thereafter at every mth execution until end-of-job. K limits the number of dumps to the kth occurrence of "line-num".

HEX
NOHEX
refers to the format of the Data Division areas provided in the dynamic dump. If HEX is specified, level-01 items are provided in hexadecimal. Items subordinate to level-01 items are printed in EBCDIC, if possible. Level-77 items are printed in both EBCDIC and hexadecimal. If HEX is not specified, items subordinate to level-01 items and level-77 items are provided in EBCDIC. If the items are unprintable, hexadecimal notation is provided. Note that if "name1" is specified and it represents a group item and HEX has not been specified,

neither the group nor the elementary items in the group will be provided in hexadecimal.

name1 [THRU name2]
represents selected areas of the Data Division to be dumped. With the THRU option, a range of data-names appearing consecutively in the Data Division is dumped. "name1" and "name2" may be qualified but not subscripted. If the programmer wishes to see a subscripted item, specifying the name of the item without the subscript results in a dump of every occurrence of that item.

ALL
results in a dump of everything that would be dumped in the event of an abnormal termination for the program specified in the "program-id" parameter in the preceding program-control card. One use of ALL allows the programmer to receive a formatted dump at normal return from the program. To do this, the programmer must ensure that the generated statement number of the line on which a STOP RUN, EXIT PROGRAM, or GOBACK statement appears is specified as the "line-num" parameter.

DEFAULT SYSDBG DATA SET

If the programmer fails to define a SYSDBG data set, the SYMDMP routines generate a default SYSDBG data set equivalent to the following job control language:

```
//SYSDBG DD *
  prog-id SYSUT5
/*
```

where:

prog-id
is the name of the first program compiled with the SYMDMP option encountered in the run-unit. If the programmer has provided a SYSUT5 DD statement referring to the file produced during the compilation of prog-id on SYSUT5, the effect of this default data set is to produce normal SYMDMP output on an abend.

If a run-unit includes one or more programs that have been compiled with the SYMDMP option and the programmer desires to suppress the normal SYMDMP output on an abend, either of the following methods may be used:

- omit the SYSUT5 DD statement from the execution step. This will cause the following message to be produced and SYMDMP output to be cancelled:

  IKF168I UNSUCCESSFUL OPEN OF DEBUG FILE

- define the SYSDBG data set as

  //SYSDBG DD DUMMY

  This will cause the following message to be produced and SYMDMP to be cancelled:

  IKF174I SYMDMP CANCELLED. NO CONTROL CARDS FOUND.

SYMBOLIC DEBUGGING UNDER INFORMATION MANAGEMENT SYSTEM (PP5734-XX6, 5740-XX2)

Execution of a COBOL program compiled with the options STATE, FLOW, SYMDMP, or COUNT under IMS requires the COBOL programmer to write an explicit CALL statement to subroutine ILBOSPI0 in his source program, i.e., CALL 'ILBOSPI0'.

- There should be one CALL statement written at the beginning of the Procedure Division and following each ENTRY statement in the program.

- There should be one CALL statement written at each exit point in the program, i.e., preceding each GOBACK, EXIT PROGRAM or STOP RUN statement.

- These CALL statements are effective only in a COBOL program compiled with debugging or COUNT, i.e., FLOW, STATE, SYMDMP or COUNT options. They must be executed as a logical pair only once per COBOL run unit. If COBOL program A calls COBOL program B, either A or B or both can be compiled with debugging, but only the highest level program compiled with debugging or COUNT options should contain CALL 'ILBOSPI0' statements. The first execution of ILBOSPI0 issues a SPIE macro instruction to trap the old program PSW in the event of a program check before STAE gets control at abnormal termination. The second execution of ILBOSPI0 resets any previous SPIE at task normal termination. At abnormal termination, ILBODBG0 will reset the previous SPIE.

- Finally, any CALL 'ILBOSPI0' statements written in a COBOL program compiled without any of the above options cause the subroutine to return control with no action (SPIE is not issued).

If IMS will link to a COBOL load module many times in a job step, the ENDJOB compiler option should be specified. For additional information, see the discussion of the ENDJOB option in the section "Options for the Compiler" in this manual.

SAMPLE PROGRAM -- TESTRUN

Figure 86 contains selected portions of output from a program that utilizes the Symbolic Debugging feature. In the following description of the program and its output, letters identifying the text correspond to letters in the program listing. (SYMDMP itself provides no page headings or numberings on its output.)

Ⓐ Because the SYMDMP option is requested in the PARM parameter of the EXEC card, the logical unit SYSUT5 must be assigned at compile time.

Ⓑ The PARM parameter specifications on the EXEC card indicate that an alphabetically ordered cross-reference dictionary, a flow trace of 10 procedures, and the SYMDMP option are being requested along with other options.

Ⓒ An alphabetically ordered cross-reference dictionary of data-names and procedure-names is produced by the compiler as a result of the SXREF specification in the PARM parameter of the EXEC card.

Ⓓ The file assigned at compile time to SYSUT5 to store SYMDMP information is assigned to SYSUT5 at execution time.

Ⓔ The SYMDMP control cards placed in the input stream at execution time are printed along with any diagnostics.

① The first card is the program-control card where:

(a) TESTRUN is the PROGRAM-ID.

(b) SYSUT5 is the ddname of the SYSUT5 file at execution time.

② The second card is a line-control card which requests a (HEX) formatted dynamic dump of KOUNT, NAME-FIELD, NO-OF-DEPENDENTS, and RECORD-NO prior to the first and every fourth execution of generated card number 70.

③ The third card is also a line-control card which requests a (HEX) formatted dynamic dump of WORK-RECORD and B prior to the execution of generated card number 81.

Ⓕ The type code combinations used to identify data-names in abnormal termination and dynamic dumps are defined. Individual codes are illustrated in Figure 85.

Ⓖ The dynamic dumps requested by the first line-control card.

Ⓗ The dynamic dumps requested by the second line-control card.

Ⓘ Program interrupt information is provided by the system when a program terminates abnormally.

Ⓙ The statement number information indicates the number of the verb and of the statement being executed at the time of the abnormal termination. The name of the program containing the statement is also provided.

Ⓚ A flow trace of the last 10 procedures executed is provided because FLOW=10 was specified in the PARM parameter of the EXEC card.

Ⓛ Selected areas of the Task Global Table are provided as part of the abnormal termination dump.

Ⓜ For each non-VSAM file-name, the generated card number, the file type, the file status (if QSAM), the file organization, the DCB status, and the fields of the DCB and DECB, if applicable, are provided in hexadecimal. For VSAM: the card number, OPEN/CLOSE status, organization, access mode, last I/O operation, and file status.

Ⓝ The fields of records associated with each FD are provided in the format requested on the program-control card.

- Message IKF182I appears after any record Ⓝ that is part of a closed file; the status of a file is described in Ⓜ. If the record is part of a closed file, the contents of the record are not printed; instead, the message appears. Message IKF182I is described more fully in "Appendix K: Diagnostic Messages" in this publication.

Ⓟ The contents of the fields of the Working-Storage Section are provided in the format requested on the program-control card.

Ⓠ The value associated with each of the possible subscripts is provided for each of the data items described with an OCCURS clause.

Ⓡ Asterisks appearing within the EBCDIC representation of the value of a given field indicate that the type and the actual content of the field conflict.

Note: When the SYMDMP option is used, level numbers appear "normalized" in the symbolic dump produced. For example, a group of data items described as:

```
01  RECORDA.
    05 FIELD-A.
      10  FIELD-A1 PIC X.
      10  FIELD-A2 PIC X.
```

will appear as follows in SYMDMP output:

```
01 RECORDA...
02 FIELD-A...
03 FIELD-A1...
03 FIELD-A2...
```

Debugging TESTRUN

1. Reference to the statement number information Ⓙ provided by the SYMDMP option shows that the abnormal termination occurred during the execution of the first verb on card 81.

2. Generated card number 81 contains the statement COMPUTE B = B + 1.

3. Through verification of the contents of B at the time of the abnormal termination (R), it can be seen that the usage of B (numeric packed) conflicts with the value contained in the data area reserved for B (numeric display).

4. The abnormal termination occurred during an attempt to perform an addition on a display item.

More complex errors may require the use of dynamic dumps to isolate the problem area. Line-control cards are included in TESTRUN merely to illustrate how they are used and what output they produce.

Code	Meaning
A	Alphabetic
B	Binary
D	Display
E	Edited
*	Subscripted Item
F	Floating Point
N	Numeric
P	Packed Decimal
S	Signed
OL	Overpunch Sign Leading
OT	Overpunch Sign Trailing
SL	Separate Sign Leading
ST	Separate Sign Trailing

Figure 85. Individual Type Codes Used in SYMDMP Output

```
//TESTRUN  JOB  ('A=SC40'),'BETHKE 1550 J63',MSGLEVEL=(1,1),CLASS=A,    11MVS037
//         MSGCLASS=S
***SETUP        TAPE  SCRTCH  RING=YES
(B)→//    EXEC  VSCBLCLG,
//         PARM.COB='OMAP,PMAP,SXREF,FLOW=10,SYMDMP,QUOTE,NORES',
//         GODMP='SYSOUT=S',
//       SYSUT5='&&SYSUT5,DISP=(,PASS)',
//     PARM.LKED='LIST,LET,XREF'
XXCVS20CLG PROC PROG=IKFCBL00,                                          00001000
XX COBDMP='SYSOUT=A',                                                   00002000
XX GODMP=DUMMY,                                                         00003000
XX PARMCOB='LOA',RGNCOB=128K,CONDCOB='(16,LT)',                         00004000
XX PARMLKD='LIST,XREF,LET',RGNLKED=128K,CONDLKD='(5,LT,COB)',           00005000
XX PARMGO='',RGNGO=192K,CONDGO='((5,LT,COB),(5,LT,LKED))',              00006000
XX GOSET='&GOSET',GO='GO',                                              00007000
XX S1=CBLCOMPL,V1=CBLDEV,U1=SYSDA,                                      00008000
XX S2=CBLCOMPL,V2=CBLDEV,U2=SYSDA,                                      00009000
XX S3=CBLUPM,V3=CBLDEV,U3=SYSDA,                                        00010000
XX S4=CVS20LIB,V4=CBLDEV,U4=SYSDA,                                      00011000
XX S5=CVS20LIB,V5=CBLDEV,U5=SYSDA,                                      00012000
XX S6=CVS20LIB,V6=CBLDEV,U6=SYSDA,                                      00013000
XX S7=CVS20LIB,V7=CBLDEV,U7=SYSDA,                                      00014000
XX S8=CVS20LIB,V8=CBLDEV,U8=SYSDA,                                      00015000
XX S9=CVS20LIB,V9='SER=CBLDEV',U9=SYSDA,                                00016000
XX S10='SYS1.LINKLIB',V10=,U10=                                         00017000
XXCOB  EXEC PGM=&PROG,REGION=&RGNCOB,COND=&CONDCOB,                     00027000
XX PARM='LOA,NOLIB,SIZE=128K,BUF=12K,PMA,DMA,SXR,OPT,RES,&PARMCOB'      00028000
XXSTEPLIB DD DSN=&S1,VOL=SER=&V1,UNIT=&U1,DISP=SHR                      00029000
XX        DD DSN=&S2,VOL=SER=&V2,UNIT=&U2,DISP=SHR                      00030000
XX        DD DSN=&S3,VOL=SER=&V3,UNIT=&U3,DISP=SHR                      00031000
//COB.SYSPRINT  DD  SYSOUT=S
X/SYSPRINT  DD SYSOUT=A                                                 00032000
XXSYSUDUMP DD &COBDMP                                                   00033000
XXSYSUT1 DD UNIT=SYSDA,SPACE=(TRK,(25,3)),DSN=&SYSUT1                   00034000
XXSYSUT2 DD UNIT=SYSDA,SPACE=(TRK,(25,3)),DSN=&SYSUT2                   00035000
XXSYSUT3 DD UNIT=SYSDA,SPACE=(TRK,(25,3)),DSN=&SYSUT3                   00036000
XXSYSUT4 DD UNIT=SYSDA,SPACE=(TRK,(25,3)),DSN=&SYSUT4                   00037000
(A)→XXSYSUT5 DD UNIT=SYSDA,SPACE=(TRK,(25,3)),DSN=&SYSUT5               00038000
XXSYSUT6 DD UNIT=SYSDA,SPACE=(TRK,(25,3)),DSN=&SYSUT6                   00039000
XXSYSLIN DD DSNAME=&LOADSET,DISP=(MOD,PASS),UNIT=SYSDA,                 00040000
XX SPACE=(80,(500,100))                                                 00041000
//COB.SYSIN  DD  *
```

Figure 86. Using the SYMDMP Option to Debug the Program TESTRUN (Part 1 of 11)

```
00001   100010 IDENTIFICATION DIVISION.
00002   100020 PROGRAM-ID.  TESTRUN.
00003   100030   AUTHOR. PROGRAMMER NAME.
00004   100040   INSTALLATION. NEW YORK PROGRAMMING CENTER.
00005   100050   DATE-WRITTEN. JULY 12, 1968.
00006   100060 DATE-COMPILED. JUN 11,1974.
00007   100070   REMARKS. THIS PROGRAM HAS BEEN WRITTEN AS A SAMPLE PROGRAM FOR
00008   100080      COBOL USERS. IT CREATES AN OUTPUT FILE AND READS IT BACK AS
00009   100090      INPUT.
00010
00011   100100 ENVIRONMENT DIVISION.
00012   100110 CONFIGURATION SECTION.
00013   100120   SOURCE-COMPUTER. IBM-360-H50.
00014   100130   OBJECT-COMPUTER. IBM-360-H50.
00015   100140 INPUT-OUTPUT SECTION.
00016   100150 FILE-CONTROL.
00017   100160      SELECT FILE-1 ASSIGN TO UT-2400-S-SAMPLE.
00018   100170      SELECT FILE-2 ASSIGN TO UT-2400-S-SAMPLE.
00019
00020   100180 DATA DIVISION.
00021   100190 FILE SECTION.
00022   100200 FD  FILE-1
00023   100210      LABEL RECORDS ARE OMITTED
00024   100220      BLOCK CONTAINS 100 CHARACTERS
00025   100225      RECORD CONTAINS 20 CHARACTERS
00026   100230      RECORDING MODE IS F
00027   100240      DATA RECORD IS RECORD-1.
00028   100250    01 RECORD-1.
00029   100260      02 FIELD-A PICTURE IS X(20).
00030   100270 FD  FILE-2
00031   100280      LABEL RECORDS ARE OMITTED
00032   100290      BLOCK CONTAINS 5 RECORDS
00033   100300      RECORD CONTAINS 20 CHARACTERS
00034   100310      RECORDING MODE IS F
00035   100320      DATA RECORD IS RECORD-2.
00036   100330    01 RECORD-2.
00037   100340      02 FIELD-A PICTURE IS X(20).
00038
00039   100350 WORKING-STORAGE SECTION.
00040   100360      77 KOUNT PICTURE S99 COMP SYNC.
00041   100370      77 NOMBER  PICTURE S99 COMP SYNC.
00042   100375    01 FILLER.
00043   100380      02 ALPHABET PICTURE X(26) VALUE "ABCDEFGHIJKLMNOPQRSTUVWXYZ".
00044   100395      02 ALPHA REDEFINES ALPHABET PICTURE X OCCURS 26 TIMES.
00045   100405      02 DEPENDENTS PICTURE X(26)  VALUE "012340123401234012340123401234
00046   100410-     "0".
00047   100420      02 DEPEND REDEFINES DEPENDENTS PICTURE X OCCURS 26 TIMES.
00048   100440    01 WORK-RECORD.
00049   100450      02 NAME-FIELD  PICTURE X.
00050   100460      02 FILLER PICTURE X VALUE IS SPACE.
00051   100470      02 RECORD-NO PICTURE 9999.
00052   100480      02 FILLER PICTURE X VALUE IS SPACE.
00053   100490      02 LOCATION PICTURE AAA VALUE IS "NYC".
00054   100500      02 FILLER PICTURE X VALUE IS SPACE.
00055   100510      02 NO-OF-DEPENDENTS PICTURE XX.
00056   100520      02 FILLER PICTURE X(7) VALUE IS SPACES.
00057   100521      01 RECORDA.
```

Figure 86. Using the SYMDMP Option to Debug the Program TESTRUN (Part 2 of 11)

```
00058   100522         02  A PICTURE S9(4) VALUE 1234.
00059   100523         02  B REDEFINES A PICTURE S9(7) COMPUTATIONAL-3.
00060   100530 PROCEDURE DIVISION.
00061   100540 BEGIN.
00062   100550      NOTE THAT THE FOLLOWING OPENS THE OUTPUT FILE TO BE CREATED
00063   100560      AND INITIALIZES COUNTERS.
00064   100570 STEP-1. OPEN OUTPUT FILE-1. MOVE ZERO TO KOUNT NOMBER.
00065   100580      NOTE THAT THE FOLLOWING CREATES INTERNALLY THE RECORDS TO BE
00066   100590      CONTAINED IN THE FILE, WRITES THEM ON TAPE, AND DISPLAYS
00067   100600      THEM ON THE CONSOLE.
00068   100610 STEP-2. ADD 1 TO KOUNT, ADD 1 TO NOMBER, MOVE ALPHA (KOUNT) TO
00069   100620      NAME-FIELD.
00070   100630      MOVE DEPEND (KOUNT) TO NO-OF-DEPENDENTS.
00071   100640      MOVE NOMBER TO RECORD-NO.
00072   100650 STEP-3. DISPLAY WORK-RECORD UPON CONSOLE. WRITE RECORD-1 FROM
00073   100660      WORK-RECORD.
00074   100670 STEP-4. PERFORM STEP-2 THRU STEP-3 UNTIL KOUNT IS EQUAL TO 26.
00075   100680      NOTE THAT THE FOLLOWING CLOSES OUTPUT AND REOPENS IT AS
00076   100690      INPUT.
00077   100700 STEP-5. CLOSE FILE-1. OPEN INPUT FILE-2.
00078   100710      NOTE THAT THE FOLLOWING READS BACK THE FILE AND SINGLES OUT
00079   100720      EMPLOYEES WITH NO DEPENDENTS.
00080   100730 STEP-6. READ FILE-2 RECORD INTO WORK-RECORD AT END GO TO STEP-8.
00081   100731         COMPUTE B = B + 1.
00082   100740 STEP-7. IF NO-OF-DEPENDENTS IS EQUAL TO "0" MOVE "Z" TO
00083   100750      NO-OF-DEPENDENTS. EXHIBIT NAMED WORK-RECORD. GO TO
00084   100760      STEP-6.
00085   100770 STEP-8. CLOSE FILE-2.
00086   100780      STOP RUN.
```

(C) → CROSS-REFERENCE DICTIONARY

DATA NAMES	DEFN	REFERENCE			
A	000058				
ALPHA	000044	000068			
ALPHABET	000043				
B	000059	000081			
DEPEND	000047	000070			
DEPENDENTS	000045				
FIELD-A	000029				
FIELD-A	000037				
FILE-1	000017	000064	000072	000077	
FILE-2	000018	000077	000080	000085	
KOUNT	000040	000064	000068	000070	000074
LOCATION	000053				
NAME-FIELD	000049	000068			
NO-OF-DEPENDENTS	000055	000070	000082		
NOMBER	000041	000064	000068	000071	
RECORD-NO	000051	000071			
RECORD-1	000028	000072			
RECORD-2	000036	000080			
RECORDA	000057				
WORK-RECORD	000048	000072	000080	000083	

Figure 86. Using the SYMDMP Option to Debug the Program TESTRUN (Part 3 of 11)

```
PROCEDURE NAMES                 DEFN     REFERENCE


BEGIN                           000061
STEP-1                          000064
STEP-2                          000068   000074
STEP-3                          000072   000074
STEP-4                          000074
STEP-5                          000077
STEP-6                          000080   000083
STEP-7                          000082
STEP-8                          000085   000080


CARD   ERROR MESSAGE

14     IKF1183I-W     IBM-370 IS ONLY VALID COMPUTER-NAME. IBM-360 SPECIFICATION IGNORED.
58     IKF2190I-W     PICTURE CLAUSE IS SIGNED, VALUE CLAUSE UNSIGNED. ASSUMED POSITIVE.



IEC130I SYSLIB   DD STATEMENT MISSING
IEF142I - STEP WAS EXECUTED - COND CODE 0004
IEF285I   VSCBL1.LMOD                                    KEPT
IEF285I   VOL SER NOS= DB143 .
IEF285I   SYS74162.T203933.RV000.TESTRUN.R0000011        DELETED
IEF285I   VOL SER NOS= 231400.
IEF285I   SYS74162.T203933.RV000.TESTRUN.R0000012        DELETED
IEF285I   VOL SER NOS= 333001.
IEF285I   SYS74162.T203933.RV000.TESTRUN.R0000013        DELETED
IEF285I   VOL SER NOS= 333001.
IEF285I   SYS74162.T203933.RV000.TESTRUN.R0000014        DELETED
IEF285I   VOL SER NOS= 231400.
IEF285I   SYS74162.T203933.RV000.TESTRUN.SYMDBG          PASSED
IEF285I   VOL SER NOS= 333001.
IEF285I   SYS74162.T203933.RV000.TESTRUN.R0000015        DELETED
IEF285I   VOL SER NOS= 231400.
IEF285I   SYS74162.T203933.RV000.TESTRUN.LKEDINP         PASSED
IEF285I   VOL SER NOS= 333001.
IEF285I   SYS74162.T203933.SV000.TESTRUN.R0000016        DELETED
IEF285I   VOL SER NOS= 333001.
IEF285I   SYS74162.T203933.SV000.TESTRUN.R0000017        SYSOUT
IEF285I   VOL SER NOS= 231400.
IEF285I   SYS74162.T203933.RV000.TESTRUN.S0000018        SYSIN
IEF285I   VOL SER NOS= 231400.
IEF285I   SYS74162.T203933.RV000.TESTRUN.S0000018        DELETED
IEF285I   VOL SER NOS= 231400.
IEF373I STEP /COB      / START 74162.2054
IEF374I STEP /COB      / STOP  74162.2058 CPU   0MIN 12.17SEC STOR VIRT 128K
XXLKED   EXEC PGM=IEWL,PARM='XREF,LIST,LET',COND=(5,LT,COB),REGION=128K 00800190
XXSYSLIB   DD DSN=VSCBL1.LIB,DISP=SHR,UNIT=2314,VOL=SER=DB143           00800200
XXSYSLIN   DD DSN=&LKEDINP,DISP=(OLD,DELETE)                            00800210
XX         DD DDNAME=SYSIN                                              00800220
XXSYSUT1   DD SPACE=(1024,(50,20)),UNIT=(2314,SEP=SYSLIN)               00800230
XXSYSLMOD  DD DSN=&LMODLIB(MBRNAME),UNIT=(2314,SEP=(SYSLIN,SYSUT1)),    *00800240
XX            DISP=(MOD,PASS),SPACE=(1024,(50,20,1))                    00800250
//LKED.SYSPRINT  DD  SYSOUT=G
X/SYSPRINT DD SYSOUT=A                                                  00800260
IEF236I ALLOC. FOR TESTRUN  LKED
IEF237I 233   ALLOCATED TO SYSLIB
IEF237I 250   ALLOCATED TO SYSLIN
IEF237I 234   ALLOCATED TO SYSUT1
IEF237I 235   ALLOCATED TO SYSLMOD
IEF237I 230   ALLOCATED TO SYSPRINT
```

Figure 86. Using the SYMDMP Option to Debug the Program TESTRUN (Part 4 of 11)

```
XXSYSLIB DD DSN=&S4,VOL=SER=&V4,UNIT=&U4,DISP=SHR                          00047000
XX   DD DSN=&S5,VOL=SER=&V5,UNIT=&U5,DISP=SHR                             00048000
XX   DD DSN=&S9,VOL=&V9,UNIT=&U9,DISP=SHR                                 00049000
XXSYSUT1 DD UNIT=(SYSDA,SEP=(SYSLIN,SYSLMOD)),SPACE=(1024,(50,20))        00050000
//LKED.SYSPRINT  DD  SYSOUT=S
X/SYSPRINT DD SYSOUT=A                                                    00051000
XXGO EXEC PGM=*.LKED.SYSLMOD,PARM='&PARMGO',REGION=&RGNGO,COND=&CONDGO    00052000
XXSTEPLIB DD DSN=&S6,VOL=SER=&V6,UNIT=&U6,DISP=SHR                        00053000
XX   DD DSN=&S8,VOL=SER=&V8,UNIT=&U8,DISP=SHR                             00054000
XX   DD DSN=&S7,VOL=SER=&V7,UNIT=&U7,DISP=SHR                             00055000
XX   DD DSN=&S10,VOL=&V10,UNIT=&U10,DISP=SHR                              00056000
//GO.SYSOUT  DD  SYSOUT=S
X/SYSOUT DD SYSOUT=A                                                      00057000
XXSYSUDUMP DD &GODMP                                                      00058000
//GO.SYSDBOUT  DD  SYSOUT=S
(D)→//GO.SYSUT5  DD  DSN=&&SYSUT5,DISP=(OLD,PASS)
//GO.SAMPLE  DD  UNIT=2400,LABEL=(,NL),DISP=(NEW,DELETE)
//GO.SYSDBG  DD  *
//
```

Figure 86. Using the SYMDMP Option to Debug the Program TESTRUN (Part 5 of 11)

```
                                        (E)→SYMDMP CONTROL CARDS

(1)→TESTRUN,SYSUT5
(2)→70,ON 1,4,(HEX),KOUNT,NAME-FIELD,NO-OF-DEPENDENTS,RECORD-NO
(3)→81,(HEX),WORK-RECORD,B

                                        NO ERRORS FOUND IN CONTROL CARDS


                                              (F)→TYPE CODES USED IN SYMDMP OUTPUT

                                        CODE      MEANING

                                         A      = ALPHABETIC
                                         AN     = ALPHANUMERIC
                                         ANE    = ALPHANUMERIC EDITED
                                         D      = DISPLAY (STERLING NONREPORT)
                                         DE     = DISPLAY EDITED (STERLING REPORT)
                                         F      = FLOATING POINT (COMP-1/COMP-2)
                                         FD     = FLOATING POINT DISPLAY (EXTERNAL FLOATING POINT)
                                         NB     = NUMERIC BINARY UNSIGNED (COMP)
                                         NB-S   = NUMERIC BINARY SIGNED
                                         ND     = NUMERIC DISPLAY UNSIGNED (EXTERNAL DECIMAL)
                                         ND-OL  = NUMERIC DISPLAY OVERPUNCH SIGN LEADING
                                         ND-OT  = NUMERIC DISPLAY OVERPUNCH SIGN TRAILING
                                         ND-SL  = NUMERIC DISPLAY SEPARATE SIGN LEADING
                                         ND-ST  = NUMERIC DISPLAY SEPARATE SIGN TRAILING
                                         NE     = NUMERIC EDITED
                                         NP     = NUMERIC PACKED DECIMAL UNSIGNED (COMP-3)
                                         NP-S   = NUMERIC PACKED DECIMAL SIGNED
                                         *      = SUBSCRIPTED

(G)→TESTRUN  AT CARD 000070
     LOC     CARD    LV NAME                              TYPE     VALUE

     015758  000040  77 KOUNT                                NB-S     +01
                                                    (HEX)            0001

     015798  000049  02 NAME-FIELD                           AN       A

     0157A3  000055  02 NO-OF-DEPENDENTS                    AN       **
                                                    (HEX)            0000

     0157BA  000051  02 RECORD-NO                           ND       ****
                                                    (HEX)            00000000


     TESTRUN  AT CARD 000070



      LOC     CARD    LV NAME                              TYPE     VALUE

     015758  000040  77 KOUNT                                NB-S     +05
                                                    (HEX)            0005

     015798  000049  02 NAME-FIELD                           AN       E

     0157A3  000055  02 NO-OF-DEPENDENTS                    AN       3

     0157BA  000051  02 RECORD-NO                           ND       0004
```

Figure 86. Using the SYMDMP Option to Debug the Program TESTRUN (Part 6 of 11)

```
TESTRUN  AT CARD 000070
 LOC      CARD    LV NAME                          TYPE     VALUE

075758   000040  77 KOUNT                         NB-S     +09
                                          (HEX)            0009

075798   000049  02 NAME-FIELD                    AN       I

0757A3   000050  02 NO-OF-DEPENDENTS              AN       2

0757AA   000051  02 RECORD-NO                     ND       0008


TESTRUN  AT CARD 000070
 LOC      CARD    LV NAME                          TYPE     VALUE

075758   000040  77 KOUNT                         NB-S     +13
                                          (HEX)            000D

075798   000049  02 NAME-FIELD                    AN       M

0757A3   000050  02 NO-OF-DEPENDENTS              AN       1

0757AA   000051  02 RECORD-NO                     ND       0012


TESTRUN  AT CARD 000070
 LOC      CARD    LV NAME                          TYPE     VALUE

075758   000040  77 KOUNT                         NB-S     +17
                                          (HEX)            0011

075798   000049  02 NAME-FIELD                    AN       Q

0757A3   000050  02 NO-OF-DEPENDENTS              AN       0

0757AA   000051  02 RECORD-NO                     ND       0016


TESTRUN  AT CARD 000070
 LOC      CARD    LV NAME                          TYPE     VALUE

075758   000040  77 KOUNT                         NB-S     +21
                                          (HEX)            0015

075798   000049  02 NAME-FIELD                    AN       U

0757A3   000050  02 NO-OF-DEPENDENTS              AN       4

0757AA   000051  02 RECORD-NO                     ND       0020


TESTRUN  AT CARD 00007C
 LOC      CARD    LV NAME                          TYPE     VALUE

075758   000040  77 KOUNT                         NB-S     +25
                                          (HEX)            0019

075798   000049  02 NAME-FIELD                    AN       Y

0757A3   000050  02 NO-OF-DEPENDENTS              AN       3

0757AA   000051  02 RECORD-NO                     ND       0024


(H)→TESTRUN  AT CARD 000081
 LOC      CARD    LV NAME                          TYPE     VALUE

         000046  01 WORK-RECORD
075758                                    (HEX)            C140F0F0 F0F140D5 E8C340F0   40404040 40404040
075798   000049  02 NAME-FIELD                    AN       A
0757A0   00005C  02 FILLER                       AN
0757AA   000051  02 RECORD-NO                     ND       0001
0757AE   000052  02 FILLER                       AN
0757AF   000053  02 LOCATION                     A        NYC
0757B2   000054  02 FILLER                       AN
0757B3   000055  02 NO-OF-DEPENDENTS              AN       0
0757B5   000056  02 FILLER                       AN

075780   000059  02 B                             NP-S     *1*2*3*
                                          (HEX)            F1F2F3C4
```

Figure 86. Using the SYMDMP Option to Debug the Program TESTRUN (Part 7 of 11)

```
                                COBOL ABEND DIAGNOSTIC AIDS

    PROGRAM        TESTRUN

(I)->LAST PSW BEFORE ABEND = FF8500007E0075F46      SYSTEM COMPLETION CODE = 0C7
(J)->LAST CARD NUMBER/VERB NUMBER EXECUTED -- CARD NUMBER 000081/VERB NUMBER 01.
                                                                FLOW TRACE
(K)->TESTRUN  000068 000072 000068 000072 000068 000072 000068 000072 000077 000080


                                DATA DIVISION DUMP OF TESTRUN

(L)->TASK GLOBAL TABLE        LOC      VALUE

     SAVE AREA                07590U   0030C4C2 00074F80 00074B90 50075F24   6001FC3C 40075F3C 0007AA70 00075900
                              0759FU   0000001A 000758C8 000756B8 00075758   0007AA1C 0007AA70 00076082 000756B8
                              075A1U   00076176 00075C18
     SWITCH                   075A1C   3D02804B
     TALLY                    075A1L   00000000
     SORT SAVE                075A2U   00000000
     ENTRY-SAVE               075A24   00075C9C
     SORT CORE SIZE           075A2C   00000000
     RET CODE                 075A2L   0000
     SORT RET                 075A2E   0000
     WORKING CELLS            075A3U   04400000 50075F24 12D474E8 0007AC10   00075900 00075900 0000001A 000758C8
                              075A5U   000756B8 00075758 0007AA1C 0007BFE8   00076082 000756B8 00075C9C 00075C18
                              075A7U   50077EF0 46075F36 0007AC10 80075900   000757F4 0000001A 08F0F000 00000000
                              075A9U   00000000 588DD1D8 F870D210 C0560700   5820D1A8 07F20000 000009C8 00074ED0
                              075ABU   20000000 00000000 00000000 00000000   00000000 00000000 00000000 00075D9E
                              075ADU   00000000 01000000 00075D8C 00076F6A   60075D48 00075DA0 00000000 0000001A
                              075AFU   00075793 500760E6 00018C80 0007795A   00075960 80075900 00075900 0000001A
                              075B1U   000758C8 00075688 00075758 0007AA1C   00000000 00076082 000756B8 00075C9C
                              075B3U   00075C18 00000000 00000000 00000000   00000000 00000000 00000000 00000000
                              075B5U   00000000 00000000 00000000 00000000
     SORT FILE SIZE           075B6U   00000000
     SORT MODE SIZE           075B64   00000000
     PGT-VN TBL               075B6A   00000000
     IGT-VN TBL               075B6C   00000000
     RESERVED                 075B7U   00000000
     LENGTH OF VN TBL         075B74   0000
     LABEL RET                075B76   00
     RESERVED                 075B77   00
     DBG R14SAVE              075B78   40075F3C
     COBOL INDICATOR          075B7C   6007839C
     A(INITI)                 075B8U   00075688
     DEBUG TABLE PTR          075B84   00000558
     SUBCOM ADDR              075B88   00078D98
     SORT-MESSAGE             075B8C   SYSOUT
     SYSOUT DDNAME            075B94   E3
     RESERVED                 075B9D   80
     COBOL ID                 075B9E   000C
     COMPILED POINTER         075BA0   00075740
     COUNT TABLE ADDRESS      075BA4   00000000
     RESERVED                 075BAU   00000000 00000000
     DBG R11SAVE              075BA8   00075C9C
     COUNT CHAIN ADDRESS      075BAC   00000000
     PRELI CELL PTR           075BBU   00075C98
     UNUSED                   075BB4   00000000 00
     TA LENGTH                075BBD   000000
     RESERVED                 075BBE   00000000 00000000
     PCS LIT PTR              075BC4   00000000
     DEBUGGING                075BC8   00000300
     CD FOR INITIAL INPUT     075BCC   00000000
     OVERFLOW CELLS           (NONE)

     EL CELLS                 075BDU   00075688 0007AA70 00075758
     DECBADR CELLS            (NONE)
     FIB CELLS                075BDC   00000000
     DEBUG TRANSFER           (NONE)
     DEBUG CARD               (NONE)
     DEBUG BLL                (NONE)
     DEBUG VLC                (NONE)
     DEBUG MAX                (NONE)
     RESERVED                 (NONE)
     DEBUG PTR                (NONE)
     TEMP STORAGE             075BEU   00000000 0000001C
     BLL CELLS                075BE8   00000000 00000000
     VLC CELLS                (NONE)
     SBL CELLS                (NONE)
     INDEX CELLS              (NONE)
     OTHER (SEE MEMORY MAP)   075BFU   00075779 00075793 00075DF6 00075DF6   00000000 80075900 00000000 00000000
                              075C1U   0A00098A 15130000
     LPSI=00000000
```

Figure 86. Using the SYMDMP Option to Debug the Program TESTRUN (Part 8 of 11)

```
                                    DATA DIVISION DUMP OF TESTRUN

     LCC      CARD   LV NAME                          TYPE     VALUE

(M)------>000017  FD FILE-1                           QSAM     FILE: CLOSED   ORGANIZATION: PHYSICAL SEQUENTIAL
                                                               LAST SUCCESSFUL I/O STMT: CLOSE     FILE STATUS: 00
     C757F4                                  DCB     00000000 00000000 00000000 00000006   00830000 0007AA01 00004000 000000
     C75814                                          46000001 0007578C E2C1D4D7 D3C54040   02000048 00000001 08000001 00000C
     C75834                                          00000000 00000001 00000001 00000001   00000014 00000001 00000000 000000

          000028  01 RECORD-1

     IKF182I  UNINITIALIZED OR INVALID BASE ADDRESS FOR DATA ITEM ABOVE.

(N)------>000029  02 FIELD-A                          AN

     IKF182I  UNINITIALIZED OR INVALID BASE ADDRESS FOR DATA ITEM ABOVE.

(M)------>000018  FD FILE-2                           QSAM     FILE: OPEN     ORGANIZATION: PHYSICAL SEQUENTIAL
                                                               LAST SUCCESSFUL I/O STMT: READ      FILE STATUS: 00
     C75900                                  DCB     00000000 00000000 00000000 00000005   0083C300 0507AA00 00004000 0007A8
     C75920                                          46077F8C 900758C8 00CC4800 007DC414   12D474E8 00BE1018 07000001 000000
     C75940                                          20202020 0007A920 0007AAD4 0007AA70   00000014 00000001 00000000 00BD6A

(N)
          000036  01 RECORD-2
     C7AA70  000037  02 FIELD-A                          AN       A 0001 NYC 0
(P)->C75758  000040  77 KOUNT                            NB-S     +26
     C7575A  000041  77 NUMBER                           NB-S     +26

          000042  01 FILLER
     C75760  000043  02 ALPHABET                         AN       ABCDEFGHIJKLMNOPQRSTUVWXYZ
          000044  02 ALPHA                              *AN

                          (SUB1)
(Q)->C75760                 1                                    A
     C75761                 2                                    B
     C75762                 3                                    C
     C75763                 4                                    D
     C75764                 5                                    E
     C75765                 6                                    F
     C75766                 7                                    G
     C75767                 8                                    H
     C75768                 9                                    I
     C75769                10                                    J
     C7576A                11                                    K
     C7576B                12                                    L
     C7576C                13                                    M
     C7576D                14                                    N
     C7576E                15                                    O
     C7576F                16                                    P
     C75770                17                                    Q
     C75771                18                                    R
     C75772                19                                    S
     C75773                20                                    T
     C75774                21                                    U
```

Figure 86. Using the SYMDMP Option to Debug the Program TESTRUN (Part 9 of 11)

```
                                DATA DIVISION DUMP OF TESTRUN

 LOC      CARD    LV NAME                          TYPE    VALUE

075775                    22                              V
075776                    23                              W
075777                    24                              X
075778                    25                              Y
075779                    26                              Z
07577A  000045  02 DEPENDENTS                       AN      012340123401234012340123 40
        000047  02 DEPEND                           *AN

                        (SUB1) <-- (Q)
07577A                     1                              0
07577B                     2                              1
07577C                     3                              2
07577D                     4                              3
07577E                     5                              4
07577F                     6                              0
075780                     7                              1
075781                     8                              2
075782                     9                              3
075783                    10                              4
075784                    11                              0
075785                    12                              1
075786                    13                              2
075787                    14                              3
075788                    15                              4
075789                    16                              0
07578A                    17                              1
07578B                    18                              2
07578C                    19                              3
07578D                    20                              4
07578E                    21                              0
07578F                    22                              1
075790                    23                              2
075791                    24                              3
075792                    25                              4
075793                    26                              0

        000046  01 WORK-RECORD <-- (P)
075798  000049  02 NAME-FIELD                         AN      A
075799  000050  02 FILLER                            AN
07579A  000051  02 RECORD-NO                         ND      0001
07579E  000052  02 FILLER                            AN
07579F  000053  02 LOCATION                         A       NYC
0757A2  000054  02 FILLER                            AN
0757A3  000055  02 NO-OF-DEPENDENTS                 AN      0
0757A5  000056  02 FILLER                            AN

        000057  01 RECORD2
0757B0  000058  02 A                                 ND-OT   +1234
075780  000059  02 B                                 NP-S    *1*2*3* <-- (R)
                                                (HEX)       F1F2F3C4

 LOC      CARD    LV NAME                          TYPE    VALUE

                                END OF COBOL DIAGNOSTIC AIDS
```

Figure 86. Using the SYMDMP Option to Debug the Program TESTRUN (Part 10 of 11)

```
IEC130I SYSDTERM DD STATEMENT MISSING
A 0001 NYC 0
B 0002 NYC 1
C 0003 NYC 2
D 0004 NYC 3
E 0005 NYC 4
F 0006 NYC 0
G 0007 NYC 1
H 0008 NYC 2
I 0009 NYC 3
J 0010 NYC 4
K 0011 NYC 0
L 0012 NYC 1
M 0013 NYC 2
N 0014 NYC 3
O 0015 NYC 4
P 0016 NYC 0
Q 0017 NYC 1
R 0018 NYC 2
S 0019 NYC 3
T 0020 NYC 4
U 0021 NYC 0
V 0022 NYC 1
W 0023 NYC 2
X 0024 NYC 3
Y 0025 NYC 4
Z 0026 NYC 0
COMPLETION CODE - SYSTEM=0C7  USER=0000
IEF285I   SYS74162.T203933.RV000.TESTRUN.LMODLIB       PASSED
IEF285I   VOL SER NOS= DC151 .
IEF285I   VSCBL1.LIB                                   KEPT
IEF285I   VOL SER NOS= DB143 .
IEF285I   SYS74162.T203933.RV000.TESTRUN.SYMDBG        DELETED
IEF285I   VOL SER NOS= 333001.
IEF285I   SYS74162.T203933.SV000.TESTRUN.R0000021      SYSOUT
IEF285I   VOL SER NOS= 231400.
IEF285I   SYS74162.T203933.SV000.TESTRUN.R0000022      SYSOUT
IEF285I   VOL SER NOS= 333001.
IEF285I   SYS74162.T203933.SV000.TESTRUN.R0000023      DELETED
IEF285I   VOL SER NOS= 333001.
IEF285I   SYS74162.T203933.SV000.TESTRUN.R0000024      DELETED
IEF285I   VOL SER NOS= 231400.
IEF285I   SYS74162.T203933.RV000.TESTRUN.R0000025      DELETED
IEF285I   VOL SER NOS= L00001.
IEF285I   SYS74162.T203933.RV000.TESTRUN.S0000026      SYSIN
IEF285I   VOL SER NOS= 231400.
IEF285I   SYS74162.T203933.RV000.TESTRUN.S0000026      DELETED
IEF285I   VOL SER NOS= 231400.
IEF373I STEP /GO      / START 74162.2059
IEF374I STEP /GO      / STOP  74162.2105 CPU   0MIN 09.72SEC STOR VIRT  64K
IEF285I   SYS74162.T203933.RV000.TESTRUN.LMODLIB       DELETED
IEF285I   VOL SER NOS= DC151 .
IEF375I  JOB /TESTRUN / START 74162.2054
IEF376I  JOB /TESTRUN / STOP  74162.2105 CPU   0MIN 24.96SEC
```

Figure 86. Using the SYMDMP Option to Debug the Program TESTRUN (Part 11 of 11)

The compiler, linkage editor, COBOL load module, and other system components can produce output in the form of printed listings, punched card decks, diagnostic or informative messages, and data sets directed to tape or mass storage devices. This chapter describes the output listings that can be used to document and debug programs and the format of the output modules. The same COBOL program is used for each example. "Appendix A: Sample Program Output" shows the output formats in the context of a complete listing generated by a sample program.

COMPILER OUTPUT

The output of the compilation job step may include:

- A printed listing of the job control statements
- Device allocation and deallocation messages from the job scheduler
- A printed listing of the statements contained in the source module
- A glossary of compiler-generated information about data
- A printed listing of the object code
- Compiler diagnostic messages
- System messages
- Disposition messages from the job scheduler
- An object module
- A cross-reference listing
- A condensed listing containing source card numbers and the location of the generated instruction for each verb
- Compiler statistics
- Reformatted source code, either with or without a compilation, through the lister feature of the OS/VS COBOL Compiler. For details, see the chapter "Lister Feature."

Diagnostic messages associated with the compilation of the source program are automatically generated as output. The other forms of output may be requested in the PARM parameter in the EXEC statement. The level of diagnostic messages printed depends upon the FLAGW or FLAGE options.

All output to be listed is written on the device specified by the SYSPRINT DD statement. Line spacing of the source listing and the number of lines per page can be controlled by the SPACEn and LINECNT options.

Figure 87 contains a portion of the compiler output listing shown in "Appendix A: Sample Program Output." Each type of output is numbered, and each format within each type is lettered. The text following Figure 87 is an explanation of the illustration.

```
//TESTRUN  JOB  ('A=3C40'),'BETHKE 1550 J63',MSGLEVEL=(1,1),CLASS=A,     11MVS037
//         MSGCLASS=S
***SETUP         TAPE  SCRTCH  RING=YES
//     EXEC  CVS2OCLG,
//          PARM.COB='OMAP,PMAP,SXREF,FLOW=10,SYMDMP,QUOTE,NORES',
//          GODMP='SYSOUT=S',
//       SYSUT5='&&SYSUT5,DISP=(,PASS)',
//     PARM.LKED='LIST,LET,XREF'
XXCVS2OCLG PROC PROG=IKFCBL00,                                          00001000
XX COBDMP='SYSOUT=A',                                                   00002000
XX GODMP=DUMMY,                                                         00003000
XX PARMCOB='LOA',RGNCOB=128K,CONDCOB='(16,LT)',                         00004000
XX PARMLKD='LIST,XREF,LET',RGNLKED=128K,CONDLKD='(5,LT,COB)',           00005000
XX PARMGO='',RGNGO=192K,CONDGO='((5,LT,COB),(5,LT,LKED))',              00006000
XX GOSET='&GOSET',GO='GO',                                              00007000
XX S1=CBLCOMPL,V1=CBLDEV,U1=SYSDA,                                      00008000
XX S2=CBLCOMPL,V2=CBLDEV,U2=SYSDA,                                      00009000
XX S3=CBLUPM,V3=CBLDEV,U3=SYSDA,                                        00010000
XX S4=CVS2OLIB,V4=CBLDEV,U4=SYSDA,                                      00011000
XX S5=CVS2OLIB,V5=CBLDEV,U5=SYSDA,                                      00012000
XX S6=CVS2OLIB,V6=CBLDEV,U6=SYSDA,                                      00013000
XX S7=CVS2OLIB,V7=CBLDEV,U7=SYSDA,                                      00014000
XX S8=CVS2OLIB,V8=CBLDEV,U8=SYSDA,                                      00015000
XX S9=CVS2OLIB,V9='SER=CBLDEV',U9=SYSDA,                                00016000
XX S10='SYS1.LINKLIB',V10=,J10=                                         00017000
XXCOB EXEC PGM=&PROG,REGION=&RGNCOB,COND=&CONDCOB,                      00027000
XX PARM='LOA,NOLIB,SIZE=128K,BUF=12K,PMA,DMA,SXR,OPT,RES,&PARMCOB'      00028000
XXSTEPLIB DD DSN=&S1,VOL=SER=&V1,UNIT=&U1,DISP=SHR                      00029000
XX        DD DSN=&S2,VOL=SER=&V2,UNIT=&U2,DISP=SHR                      00030000
XX        DD DSN=&S3,VOL=SER=&V3,UNIT=&U3,DISP=SHR                      00031000
//COB.SYSPRINT  DD  SYSOUT=S
X/SYSPRINT  DD SYSOUT=A                                                 00032000
XXSYSUDUMP DD &COBDMP                                                   00033000
XXSYSUT1 DD UNIT=SYSDA,SPACE=(TRK,(25,3)),DSN=&SYSUT1                   00034000
XXSYSUT2 DD UNIT=SYSDA,SPACE=(TRK,(25,3)),DSN=&SYSUT2                   00035000
XXSYSUT3 DD UNIT=SYSDA,SPACE=(TRK,(25,3)),DSN=&SYSUT3                   00036000
XXSYSUT4 DD UNIT=SYSDA,SPACE=(TRK,(25,3)),DSN=&SYSUT4                   00037000
XXSYSUT5 DD UNIT=SYSDA,SPACE=(TRK,(25,3)),DSN=&SYSUT5                   00038000
XXSYSUT6 DD UNIT=SYSDA,SPACE=(TRK,(25,3)),DSN=&SYSUT6                   00039000
XXSYSLIN DD DSNAME=&LOADSET,DISP=(MOD,PASS),UNIT=SYSDA,                 00040000
XX SPACE=(80,(500,100))                                                 00041000
//COB.SYSIN  DD  *
```

(1)

```
IEF236I ALLOC. FOR TESTRUN COB
IEF237I 5C0 ALLOCATED TO STEPLIB
IEF237I 5C0 ALLOCATED TO
IEF237I 5C0 ALLOCATED TO
IEF237I JES ALLOCATED TO SYSPRINT
IEF237I JES ALLOCATED TO SYSUDUMP
IEF237I 272 ALLOCATED TO SYSUT1
IEF237I 150 ALLOCATED TO SYSUT2
IEF237I 272 ALLOCATED TO SYSUT3
IEF237I 272 ALLOCATED TO SYSUT4
IEF237I 272 ALLOCATED TO SYSUT5
IEF237I 272 ALLOCATED TO SYSUT6
IEF237I 272 ALLOCATED TO SYSLIN
IEF237I JES ALLOCATED TO SYSIN
```

(2)

Figure 87. Examples of Compiler Output (Part 1 of 4)

```
00001   10001U IDENTIFICATION DIVISION.
00002   10002U PROGRAM-ID.  TESTRUN.
00003   10003U   AUTHOR. PROGRAMMER NAME.
00004   10004U   INSTALLATION.  PALO ALTO DEVELOPMENT CENTER.
00005   10005U   DATE WRITTEN.  AUGUST 8, 1976.
00006   10006U DATE-COMPILED. AUG 24,1976
(3)  :
00082   10074U STEP-7. IF NO-OF-DEPENDENTS IS EQUAL TO "0" MOVE "Z" TO
00083   10075U      NO-OF-DEPENDENTS. EXHIBIT NAMED WORK-RECORD. GO TO
00084   10076U      STEP-6.
00085   10077U STEP-8. CLOSE FILE-2.
00086   10078U      STOP RUN.
```

(A) INTRNL NAME	(B) LVL	(C) SOURCE NAME	(D) BASE	(E) DISPL	(F) INTRNL NAME	(G) DEFINITION	(H) USAGE	R	O	(I) Q	M
DNM=1-148	FD	FILE-1	DCB=01		DNM=1-148		QSAM				F
DNM=1-168	01	RECORD-1	BL=1	000	DNM=1-168	DS 0CL20	GROUP				
DNM=1-189	02	FIELD-A	BL=1	000	DNM=1-189	DS 20C	DISP				
DNM=1-206	FD	FILE-2	DCB=02		DNM=1-206		QSAM				F
DNM=1-226	01	RECORD-2	BL=2	000	DNM=1-226	DS 0CL20	GROUP				
DNM=1-247	02	FIELD-A	BL=2	000	DNM=1-247	DS 20C	DISP				
DNM=1-267	77	KOUNT	BL=3	000	DNM=1-267	DS 1F	COMP				
DNM=1-282	77	NUMBER	BL=3	002	DNM=1-282	DS 1H	COMP				
DNM=1-298	01	FILLER	BL=3	008	DNM=1-298	DS 0CL52	GROUP				
DNM=1-312	02	ALPHABET	BL=3	008	DNM=1-312	DS 26C	DISP				
DNM=1-330	02	ALPHA	BL=3	008	DNM=1-330	DS 1C	DISP	R	O		
(4) DNM=1-348	02	DEPENDENTS	BL=3	022	DNM=1-348	DS 2C	DISP				
DNM=1-368	02	DEPEND	BL=3	022	DNM=1-368	DS 1C	DISP	R	O		
DNM=1-384	01	WORK-RECORD	BL=3	040	DNM=1-384	DS 0CL20	GROUP				
DNM=1-408	02	NAME-FIELD	BL=3	040	DNM=1-408	DS 1C	DISP				
DNM=1-428	02	FILLER	BL=3	041	DNM=1-428	DS 1C	DISP				
DNM=1-442	02	RECORD-NO	BL=3	042	DNM=1-442	DS 4C	DISP-NM				
DNM=1-461	02	FILLER	BL=3	046	DNM=1-461	DS 1C	DISP				
DNM=1-475	02	LOCATION	BL=3	047	DNM=1-475	DS 3C	DISP				
DNM=1-493	02	FILLER	BL=3	04A	DNM=1-493	DS 1C	DISP				
DNM=2-000	02	NO-OF-DEPENDENTS	BL=3	04B	DNM=2-000	DS 2C	DISP				
DNM=2-026	02	FILLER	BL=3	04D	DNM=2-026	DS 7C	DISP				
DNM=2-040	01	RECORD2	BL=3	058	DNM=2-040	DS 0CL4	GROUP				
DNM=2-060	02	A	BL=3	058	DNM=2-060	DS 4C	DISP-NM				
DNM=2-071	02	B	BL=3	058	DNM=2-071	DS 4P	COMP-3	R			

```
                        MEMORY MAP

                (A) TGT                         00318

                SAVE AREA                       00318
                SWITCH                          00360
                TALLY                           00364
                SORT SAVE                       00368
                ENTRY-SAVE                      0036C
(B)             SORT CORE SIZE                  00370
LITERAL POOL (HEX)

005B0 (LIT+0)     00000001  0018A1C10  0000001C  00000008  00000000  48140000
005C8 (LIT+24)    00044000  00000000  C0000000

(5)     DISPLAY LITERALS (BCD)

005D4 (LIT+36)   'WORK-RECORD'

                (C) PGT                         00560

                DEBUG LINKAGE AREA              00560
                OVERFLOW CELLS                  0056C
                VIRTUAL CELLS                   00570
                PROCEDURE NAME CELLS            0059C
                GENERATED NAME CELLS            0059C
                DCB ADDRESS CELLS               005A0
                VNI CELLS                       005A8
                LITERALS                        005B0
                DISPLAY LITERALS                005D4
                PROCEDURE BLOCK CELLS           005E0

REGISTER ASSIGNMENT

(6)  REG 6      BL =3
     REG 7      BL =1
     REG 8      BL =2

(7) WORKING-STORAGE STARTS AT LOCATION 000A0 FOR A LENGTH OF 00060.
```

Figure 87. Examples of Compiler Output (Part 2 of 4)

```
 (A)      (B)         (C)        (D)                (E)         (F)                         (G)
 61      *BEGIN
                     0005E4                         PN=02       EQU   *
                     0005E4                         START       EQU   *
                     0005E4   58 B0 C 080                       L     11,080(0,12)          PBL=1
                     0005E8   58 F0 C 024                       L     15,024(0,12)          V(ILBOFLW1)
                     0005EC   05 1F                             BALR  1,15
                     0005EE   0000003D                          DC    X'0000003D'
 64      *STEP-1
                     0005F2                         PN=03       EQU   *
(8)                  0005F2   58 F0 C 024                       L     15,024(0,12)          V(ILBOFLW1)
                     0005F6   05 1F                             BALR  1,15
                     0005F8   00000040                          DC    X'00000040'
 64        OPEN      0005FC   58 F0 C 028                       L     15,028(0,12)          V(ILBODBG4)
                     000600   05 EF                             BALR  14,15
                     000602   58 10 C 040                       L     1,040(0,12)           DCB=1
                     000606   58 40 1 024                       L     4,024(0,1)
                     00060A   D2 02 4 011 C 02D                 MVC   011(3,4),02D(12)                        V(ILBOEXT0)
                     000610   50 10 D 234                       ST    1,234(0,13)           SA3=1
                     000614   92 0F D 234                       MVI   234(13),X'0F'         SA3=1
                        :

    *STATISTICS*     SOURCE RECORDS =     86    DATA DIVISION STATEMENTS =     25     PROCEDURE DIVISION STATEMENTS =     21
    *OPTIONS IN EFFECT*      SIZE =  131072  BUF =    12288  LINECNT = 57  SPACE1, FLAGW,    SEQ,    SOURCE
    *OPTIONS IN EFFECT*         DMAP,    PMAP, NOCLIST, NOSUPMAP, NOXREF,    SXREF,    LOAD, NODECK, QUOTE, NOTRUNC,    FLOW=    10
(9) *OPTIONS IN EFFECT*      NOTERM, NONUM, NOBATCH, NONAME, COMPILE=01, NOSTATE, NORESIDENT, NODYNAM, NOLIB, NOSYNTAX
    *OPTIONS IN EFFECT*         OPTIMIZE,    SYMDMP, NOTEST,    VERB,    ZWB, SYST, NOENDJOB, NOLVL
    *OPTIONS IN EFFECT*      NOLST , NOFDECK,NOCDECK, LCOL2,  L120,    DUMP , NOADV , NOPRINT,
    *OPTIONS IN EFFECT*      NOCOUNT, NOVBSUM, NOVBREF, LANGLVL(2)
    *OPTIONS IN EFFECT*      DEBUG FILE SIZE = 2      BLOCKS, 1024    BYTES


                                           CROSS-REFERENCE DICTIONARY

     DATA NAMES                        DEFN      REFERENCE

     A                                 000058
     ALPHA                             000044    000068
     ALPHABET                          000043
     B                                 000059    000081
     DEPEND                            000047    000070
     DEPENDENTS                        000045
     FIELD-A                           000029
     FIELD-A                           000037
     FILE-1                            000017    000064   000072   000077
     FILE-2                            000018    000077   000080   000085
     KOUNT                             000040    000064   000068   000070   000074
     LOCATION                          000053
     NAME-FIELD                        000049    000068
     NO-OF-DEPENDENTS                  000055    000070   000082
     NUMBER                            000041    000064   000068   000071
     RECORD-NO                         000051    000071
(10) RECORD-1                          000028    000072
     RECORD-2                          000036    000080
     RECORDA                           000057
     WORK-RECORD                       000048    000072   000080   000083


     PROCEDURE NAMES                   DEFN      REFERENCE

     BEGIN                             000061
     STEP-1                            000064
     STEP-2                            000068    000074
     STEP-3                            000072    000074
     STEP-4                            000074
     STEP-5                            000077
     STEP-6                            000080    000083
     STEP-7                            000082
     STEP-8                            000085    000080
```

Figure 87. Examples of Compiler Output (Part 3 of 4)

```
(11) CARD   ERROR MESSAGE
     5      IKF1067I-W     ' DATE ' SHOULD NOT BEGIN A-MARGIN.
     58     IKF2190I-W     PICTURE CLAUSE IS SIGNED, VALUE CLAUSE UNSIGNED. ASSUMED POSITIVE.


IEF142I TESTRUN COB - STEP WAS EXECUTED - COND CODE 0004
IEF285I   CBLCOMPL                                        KEPT
IEF285I   VOL SER NOS= CBLDEV.
IEF285I   CBLCOMPL                                        KEPT
IEF285I   VOL SER NOS= CBLDEV.
IEF285I   CBLGPM                                          KEPT
IEF285I   VOL SER NOS= CBLDEV.
IEF285I   JES2.JOB00059.S00103                            SYSOUT
IEF285I   JES2.JOB00059.S00104                            SYSOUT
IEF285I   SYS76237.T091056.RA000.TESTRUN.SYSUT1           DELETED
IEF285I   VOL SER NOS= 222222.
IEF285I   SYS76237.T091056.RA000.TESTRUN.SYSUT2           DELETED
(12) IEF285I   VOL SER NOS= 000000.
IEF285I   SYS76237.T091056.RA000.TESTRUN.SYSUT3           DELETED
IEF285I   VOL SER NOS= 222222.
IEF285I   SYS76237.T091056.RA000.TESTRUN.SYSUT4           DELETED
IEF285I   VOL SER NOS= 222222.
IEF285I   SYS76237.T091056.RA000.TESTRUN.SYSUT5           PASSED
IEF285I   VOL SER NOS= 222222.
IEF285I   SYS76237.T091056.RA000.TESTRUN.SYSUT6           DELETED
IEF285I   VOL SER NOS= 222222.
IEF285I   SYS76237.T091056.RA000.TESTRUN.LOADSET          PASSED
IEF285I   VOL SER NOS= 222222.
IEF285I   JES2.JOB00059.SI0101                            SYSIN
IEF373I STEP /COB      / START 76237.0910
IEF374I STEP /COB      / STOP  76237.0911 CPU    0MIN 04.48SEC VIRT   136K SYS   212K
```

Figure 87. Examples of Compiler Output (Part 4 of 4)

1. Listing of job control statements associated with this job step. These statements are listed because MSGLEVEL=(1,1) is specified in the JOB statement. JCL statements with XX instead of // represent statements in a cataloged procedure.

2. Allocation messages from the job scheduler. These messages provide information about the device allocation for the data sets in the job step. They appear after the job control statements in the compile, linkage edit, and execution job steps. For example:

   IEF237I 235 ALLOCATED TO SYSUT1

   indicates that the data set for SYSUT1 has been assigned to the device 235.

3. Source module listing. The statements in the source module are listed exactly as submitted except that a compiler-generated card number is listed in the first column of each line. This number is referred to in diagnostic messages, on the XREF or SXREF listing, and in the object code listing. If NUM is specified, the programmer-encoded source numbers in columns 1 through 6 are used in each of these cases. (See the description of the NUM option under "Options for the Compiler.") The source module is not listed when the NOSOURCE option is specified.

   The following notations may appear on the listing:

   C Denotes that the statement was inserted with a COPY statement. Statements copied will not be listed if SUPPRESS is indicated.

   ** Denotes that the card is out of sequence.

   I Denotes that the card was inserted with an INSERT card.

   If DATE-COMPILED is specified in the Identification Division, any sentences in that paragraph are replaced in the listing by the date of compilation in the following format:

   DATE-COMPILED. month day, year

4. Glossary: The glossary is listed when the DMAP option is specified. The glossary contains information about names in the COBOL source program.

   A and F. The internal name generated by the compiler. This name is used in the compiler object code listing to represent the name used in the source program. It is repeated for readability.

   B. A normalized level number. This level number is determined by the compiler as follows: (1) the first level number of any

hierarchy is always 01, and increments for other levels are always by one; (2) only level numbers 03 through 49 are affected -- level numbers 66, 77, as well as 88 and FD, SD, RD, and CD indicators are not changed.

C. The data name that is used in the source module.

Note that the following Report Writer internally generated data-names can appear under the SOURCE NAME column:

CTL.LVL	Used to coordinate control break activities.
GRP.IND	Used by coding generated for GROUP INDICATE clause.
TER.COD	Used by coding generated for TERMINATE statement.
FRS.GEN	Used by coding generated for GENERATE statement.
-nnnn	Generated report record associated with the file on which the report is to be printed.
RPT.RCD	Build area for print record
CTL.CHR	First or second position of RPT.RCD. Used for carriage control character.
RPT.LIN	Beginning of actual information that will be displayed. Second or third position of RPT.RCD.
CODE-CELL	Used to hold code specified in CODE clause.
E.nnnn	Name generated from COLUMN clause in a level-02 statement.
S.nnnn	Used for elementary level with SUM clause, but not with data-name.
N.nnnn	Used to save the total number of lines used by a report group when relative line numbering is specified.

D and E. For data names, these columns contain information about the address in the form of a base and displacement. For file names, the column contains information about the associated DCB, DECB, and FIB, if any.

G. This column defines storage for each data item. It is represented in assembler-like terminology. Figure 89 refers to information in this column.

H. Usage of the data name. For FD entries, the file processing technique is identified (e.g. QSAM, BDAM, etc.). For group items, GROUP is identified. For elementary items, the information in its USAGE clause is identified, or the USAGE that was specified on its group.

I. A letter under column:

R-Indicates that the data-name redefines another data-name.
O-Indicates that an OCCURS clause has been specified for that data-name.
Q-Indicates that the data-name is the object or contains the object of the DEPENDING ON option of the OCCURS clause.
M-Indicates that the format of the records of the file is:

F = fixed
V = variable
U = undefined
S = spanned

I-Indicates an input CD in a teleprocessing application
O-Indicates an output CD in a teleprocessing application

5. Global Tables and Literal Pool: The global table is listed when the PMAP, CLIST, or DMAP option is specified unless SUPMAP is also specified and an E-level diagnostic message is generated. A global table contains easily addressable information needed by the object program for execution. For example, in the Procedure Division source coding (3), the address of the first instruction under STEP-1, namely:

```
OPEN OUTPUT FILE-1.
```

would be found in the PROCEDURE NAME CELLS entry of the Program Global Table (PGT).

A. Task Global Table (TGT). This table consists of switches, addresses, and work areas whose information changes during execution of the program.

B. Literal Pool. The literal pool lists the collection of the literals in the program, with duplications eliminated. These literals include those specified by the programmer (e.g., MOVE "ABC" TO DATA-NAME) and those generated by the compiler (e.g., to align decimal points in arithmetic computation). The literals are divided into two

groups: those that are referred to by instructions (marked "LITERAL POOL") and those that are referred to by the calling sequences to object time subroutines (marked "DISPLAY LITERALS").

C. Program Global Table (PGT). This table contains the remaining addresses and the literals used by the object program.

For further discussion, see "Appendix J: Fields of the Global Table."

6. Register Assignment: This contains the register assigned to each base locator (BL) in the object program.

If OPT was specified, registers 6 through 9 (and register 10 if not used for the additional OVERFLOW cell of the PGT) are permanently assigned to the most frequently-used BLs. These assignments are printed in the listing. Any additional BLs have registers 14 and 15 assigned to them as and when needed. Such temporary assignments are not printed in the listing.

7. Working-Storage: When PMAP, CLIST, or DMAP is specified, both the location and the length (in hexadecimal) of the Working-Storage Section, if any, are provided.

8. Object Code Listing: The object code listing is produced when the PMAP option is specified unless SUPMAP is also specified and an E-level error is encountered. The actual object code listing contains:

A. The compiler-generated card number or source card number, if NUM is specified. The number refers to the COBOL statement in the source module that contains the verb listed under column B.

B. The relative verb number for each card number.

The statement within which the COBOL verb appears determines the information under columns C, D, F, and G.

If VERB is specified in connection with PMAP or CLIST, procedure-names and verb-names are listed with the associated code. Names preceded by an * in column B are procedure-names.

C. The relative location, in hexadecimal notation, of the object code instruction in the module.

D. The actual object code instruction in hexadecimal notation.

E. The procedure-name number. A number is assigned only to those procedure-names to which reference is made in other Procedure Division statements. This may be a PN (procedure-name) or GN (generated-name) number.

F. The object code instruction in a form that closely resembles assembler language (with displacements in hexadecimal notation).

G. Compiler-generated information about the operands of the generated instruction. This includes names and relative locations of literals. Figures 89 and 90 refer to information in this column.

Note: The programmer can produce a condensed listing by specifying CLIST as an option in place of PMAP. The CLIST option produces only the source card number, the relative verb number, and the location of the first generated instruction, as follows:

```
55  VERB1   0004AC   58   VERB1   0004C0
58  VERB2   0004F2   62   VERB1   00050E
62  VERB2   00051A   62   VERB3   000526
```

9. Statistics: The compiler statistics list the options in effect for this run and the number of Data Division and Procedure Division statements specified. Each level number is counted as one statement in the Data Division. Each verb is counted as one statement in the Procedure Division. COMPILE=nn indicates that this is the nth COBOL source program processed in this invocation of the compiler. If NOBATCH was in effect, this number will be 01. Note that COMPILE=nn is not an option that can be specified or controlled by the user, but is simply information produced by the compiler.

Note: Statistics are not printed if SYNTAX is in effect or if CSYNTAX is in effect and a D- or E-level error occurs.

10. Cross-Reference Dictionary: The XREF dictionary, produced when either the XREF or the SXREF option is specified, consists of two parts:

A. The XREF dictionary for data-names followed by the generated number or source card number of the card on which the statement begins, if NUM is in effect. For condition names, the data-name of the conditional variable appears in the XREF dictionary.

B. The XREF dictionary for procedure-names followed by the generated number or source card number of the card on which the statement begins.

For XREF, all the names begin in the order in which they are defined in the source program. For SXREF, the names appear sorted in alphanumeric order. The number of references appearing for a given name is based on the number of times the name is referred to in the compiler-generated code. (Some data-names in USE FOR DEBUGGING, STRING, UNSTRING, and SEARCH statements are not part of the compiler-generated code, and are therefore not listed in the cross reference output.

11. Diagnostic messages: The diagnostic messages associated with the compilation are always listed. The format of the diagnostic message is:

A. Compiler-generated line number or source card number. This is the number of a line in the source module related to the error.

B. Message identification. The message identification for COBOL compiler diagnostic messages always begins with the symbols IKF.

C. Severity level. There are four severity levels as follows:

W Warning -- This severity level indicates that an error was made in the source program. However, it is not serious enough to hinder the execution of the program. These warning messages are listed only if FLAGW is specified.

C Conditional -- This severity level indicates that an error was made but that the compiler makes an assumption, which in some cases corrects the error. The statement containing the error is retained. Execution can be attempted for its debugging value.

E Error -- This severity level indicates that a serious error has been detected. Usually the compiler makes no corrective assumption. The statement or operand containing the error is dropped. Execution of the program should not be attempted.

D Disaster -- This severity level indicates that a serious error was made. Compilation is not completed. Results are unpredictable.

There is a correlation between severity level and the return codes referred to by the COND parameter. For example, a compilation in which a W-level error is detected generates a return code of 4; a C-level error, a code of 8; an E-level error, of 12; and a D-level error, of 16.

D. Message text. The text identifies the condition that caused the error and indicates the action taken by the compiler.

Since Report Writer and SORT/MERGE generate a number of internal data items and procedural statements, some error messages may reflect internal names. In cases where the error manifests itself mainly in these generated routines, the error messages may indicate the card number of the RD entry for the report under consideration. In addition, there are errors that may indicate the card number of the card upon which the statement containing the error ends rather than the card upon which the error occurred. Messages for errors in the files refer to the card number of the associated SELECT clause. Internal name formats for Report Writer are discussed in the "Glossary."

When a programmer codes a verb, such as SEARCH, where the statement may span several lines and contain several verbs, such as GO TO, MOVE, DISPLAY, etc., the diagnostic will reference the card number containing the most recently encountered verb, when that verb starts a new line.

12. Disposition messages from the job scheduler: These messages contain information about the disposition of the data sets, including volume serial numbers of volumes in which the data sets reside.

DISPLAYING A LIST OF DIAGNOSTIC MESSAGES

The user can generate a complete listing of compiler diagnostic messages along with

```
//ERRMSG       JOB   User Information
//            EXEC  COBUC
//COB.SYSIN    DD    *
       ID DIVISION.
       PROGRAM-ID.  ERRMSG.
       REMARKS.  THIS PROGRAM WILL RESULT IN A LISTING OF ALL
                 COMPILER DIAGNOSTICS AND THEIR EXPLANATIONS.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       PROCEDURE DIVISION.
      *          THE SAME RESULTS CAN BE ACHIEVED BY CHANGING
      *          THE PROGRAM-NAME OF ANY PROGRAM TO 'ERRMSG.'
                STOP RUN.
/*
```

Figure 88. A Program that Produces Compiler Diagnostics and Explanations

Type	Definition[1]	Usage
Group Fixed Length	DS 0CLN	GROUP
Alphabetic	DS NC	DISP
Alphanumeric	DS NC	DISP
Alphanumeric Edited	DS NC	AN-EDIT
Group Variable Length	DS VLI=N	GROUP
Numeric edited	DS NC	NM-EDIT
Sterling Report	DS NC	RPT-ST
External Decimal	DS NC	DISP-NM
External Floating Point	DS NC	DISP-FP
Internal Floating Point	DS 1F[2] or 4C	COMP-1
	DS 1D[2] or 8C	COMP-2
Binary	DS 1H[2],1F[2],2F[2],2C,4C,8C	COMP
Internal Decimal	DS NP	COMP-3
Sterling Non-Report	DS NC	DISP-ST
Index-Name	BLANK	INDEX-NAME
File (FD)	BLANK	FILE PROCESSING TECHNIQUE
Condition (88)	BLANK	BLANK
Report Definition (RD)	BLANK	BLANK
Sort Definition (SD)	BLANK	BLANK

[1]In this column, N = size in bytes, except in group variable length where it is a variable-length cell number.
[2]If the SYNCHRONIZED clause appears, these fields are used.

Figure 89. Glossary Definition and Usage

their explanations by attempting to compile a program with a program-name of ERRMSG specified in the PROGRAM-ID paragraph. An example of how to generate this listing is shown in Figure 88.

Note: Although the user can change the program-name of any source program to ERRMSG, the following compiler options must be in effect: NOCSYNTAX, NODECK, NODMAP, NOFLOW, NOLOAD, NOOPTIMIZE, NOPMAP, NOSTATE, NOSYNTAX, NOSXREF, NOTERM, NOTEST, NOXREF; NOTERM prevents a display of the messages at the terminal.

For a list of object-time messages, see "Appendix K: Diagnostic Messages."

OBJECT MODULE

The object module contains the external symbol dictionary, the text of the program, and the relocation dictionary. It is followed by an END statement that marks the end of the module. For more detailed information about the external symbol dictionary, text, and relocation dictionary, see the publication OS/VS Linkage Editor and Loader.

An object module deck is punched if the DECK option is specified unless SUPMAP is specified and an E-level diagnostic message is generated, and if a SYSPUNCH DD statement is included. An object module is written in an output volume if the LOAD

option is specified unless SUPMAP is specified and an E-level diagnostic message is generated, and if a SYSLIN DD statement is included.

Symbol	Definition
BL	Base Locator
BLL	Base locator for linkage section
CKP	Checkpoint counter
DBGC	Debug card number
DBGI	Debug information pointer
DBGT	Debug transfer
DCB	DCB address
DEC	DECB address
DNM	Source data name
FIB	FIB address
GN	Generated procedure name
INX	Index cell
LIT	Literal
ON	On counter
OVF	Overflow cell
PBL	Procedure block (Optimizer)
PFM	Perform counter
PN	Source procedure name
POV	PGT overflow
PRM	Parameter
PSV	Perform save area
RSV	Report save area
SAV	Save area cell
SA2	Input/Output error save cell
SA3	OPEN parameter
SBL	Secondary base locator
SBS	Subscript address
SSVE	Sort save area
SWT	Switch cell
TLY	Tally cell
TOV	TGT overflow
TS	Temporary storage cell
TS2	Temporary storage (non-arithmetic)
TS3	Temporary storage (synchronization)
TS4	Temporary storage (table handling)
V(BCDNAME)	Virtual name
VIR	Virtual cell
VLC	Variable length cell
VN	Variable procedure name
VNI	Variable name initialization
WC	Working cell
XSA	Exhibit save area
XSW	Exhibit switch

Figure 90. Symbols Used in the Listing and Glossary to Define Compiler-Generated Information

LINKAGE EDITOR OUTPUT

The output of the linkage editor job step may include:

- A printed listing of the job control statements
- A map of the load module after it has been processed by the linkage editor
- A cross-reference list
- Informative messages
- Diagnostic messages
- Disposition messages
- A listing of the linkage-editor control statements
- A load module that must be assigned to a library

Any diagnostic messages or informative messages associated with the linkage editor are automatically generated as output. The other forms of output may be requested by the PARM parameter in the EXEC statement. All output to be listed is written in the data set specified by the SYSPRINT DD statement.

Figure 91 is an example of linkage editor output listing. It shows the job control statements, informative messages, and module map. The different types of output are numbered and each type to be explained is lettered. The text following Figure 91 is an explanation of the illustration.

```
    (1) XXLKED EXEC PGM=IEWL,PARM='&PARMLKD',REGION=&RGNLKED,COND=&CONDLKD     00042000
        XXSYSLIN  DD DSNAME=&LOADSET,DISP=(OLD,DELETE)                          00043000
        XX DD DDNAME=SYSIN                                                      00044000
        XXSYSLMOD DD DSNAME=&GOSET(&GO),DISP=(,PASS),UNIT=SYSDA,                00045000
        XX SPACE=(TRK,(10,2,1))                                                 00046000
        XXSYSLIB DD DSN=&S4,VOL=SER=&V4,UNIT=&U4,DISP=SHR                       00047000
        XX DD DSN=&S5,VOL=SER=&V5,UNIT=&U5,DISP=SHR                             00048000
        XX DD DSN=&S9,VOL=&V9,UNIT=&U9,DISP=SHR                                 00049000
        XXSYSUT1 DD UNIT=(SYSDA,SEP=(SYSLIN,SYSLMOD)),SPACE=(1024,(50,20))     00050000
        //LKED.SYSPRINT  DD  SYSOUT=S
        X/SYSPRINT DD SYSOUT=A                                                  00051000
    (2) IEF237I 272 ALLOCATED TO SYSLIN
        IEF237I DMY ALLOCATED TO
        IEF237I 150 ALLOCATED TO SYSLMOD
        IEF237I 5C0 ALLOCATED TO SYSLIB
        IEF237I 5C0 ALLOCATED TO
        IEF237I 5C0 ALLOCATED TO
        IEF237I 272 ALLOCATED TO SYSUT1
        IEF237I JES ALLOCATED TO SYSPRINT

(3) F64-LEVEL LINKAGE EDITOR OPTIONS SPECIFIED LIST,LET,XREF
             DEFAULT OPTION(S) USED -   SIZE=(196608,65536)

                                              CROSS REFERENCE TABLE

         (A)                                (B)
    CONTROL SECTION                         ENTRY

      NAME     ORIGIN  LENGTH                 NAME    LOCATION     NAME    LOCATION     NAME    LOCATION     NAME    LOCATION
    TESTRUN       00     A76
    ILBODBG *    A78     E36
                                            ILBODBG0    AAA      ILBODBG1    AAE      ILBODBG2    AB2      ILBODBG3    AB6
                                            ILBODBG4    ABA      ILBODBG5    ABE      ILBODBG6    AC2      ILBODBG7    AC5
    ILBODSP *   1880     9F0
                                            ILBODSP0   1882      ILBODSS0   1882
    ILBOEXT *   22A0      50
                                            ILBOEXT0   22A2      ILBOEXT1   22A6
    ILBOFLW *   22F0     536
                                            ILBOFLW0   22F2      ILBOFLW1   22F6      ILBOFLW2   22FA
    ILBOQIO *   2828     56E
(5)                                         ILBOQIO0   282A
    ILBOSRV *   2098     48E
                                            ILBOSRV0   20A2      ILBOSR5    20A2      ILBOSR3    20A2      ILBOSR     20A2
                                            ILBOSRV1   20A6      ILBOSTP1   20A6      ILBOST     20AA      ILBOSTP0   20AA
    ILBOBEG *   3228     128
                                            ILBOBEG0   322A
    ILBOCMM *   3350     388
                                            ILBOCMM0   3352      ILBOCMM1   3356
    ILBOCOMO*   36E0     169
                                            ILBOCOM    36E0
    ILBOMSG *   3850      F2
                                            ILBOMSG0   3852

         (A)              (B)              (C)
    LOCATION  REFERS TO SYMBOL  IN CONTROL SECTION              LOCATION  REFERS TO SYMBOL  IN CONTROL SECTION
        570           ILBOSRV0        ILBOSRV                       574           ILBODBG0        ILBODBG
        578           ILBOFLW0        ILBOFLW                       57C           ILBODBG5        ILBODBG
        580           ILBOSR5         ILBOSRV                       584           ILBOFLW1        ILBOFLW
        588           ILBODBG4        ILBODBG                       58C           ILBOEXT0        ILBOEXT
(6)     590           ILBOQIO0        ILBOQIO                       594           ILBODSP0        ILBODSP
        598           ILBOSRV1        ILBOSRV                       4D0           ILBOCOM0        ILBOCOMO
       1774           ILBOFLW0        ILBOFLW                      1776           ILBOFLW2        ILBOFLW
       177C           ILBOTEF3        $UNRESOLVED(W)               1780           ILBOSTN0        $UNRESOLVED(W)
       1788           ILBOTC00        $UNRESOLVED(W)               30E8           ILBOCOM         ILBOCOMO
       30EC           ILBOCMM0        ILBOCMM                      30F0           ILBOBEG0        ILBOBEG
       30F4           ILBOMSG0        ILBOMSG                      30F8           ILBOSND2        $UNRESOLVED(W)
    ENTRY ADDRESS        00 (C)
(5)
    TOTAL LENGTH       3948 (D)
   ****GO         DOES NOT EXIST BUT HAS BEEN ADDED TO DATA SET
(4) AUTHORIZATION CODE IS         0.


    IEF142I TESTRUN LKED - STEP WAS EXECUTED - COND CODE 0000
    IEF285I   SYS76237.T091056.RA000.TESTRUN.LOADSET        DELETED
    IEF285I   VOL SER NOS= 222222.
    IEF285I   SYS76237.T091056.RA000.TESTRUN.GOSET          PASSED
    IEF285I   VOL SER NOS= 000000.
    IEF285I   CVS.OLIB                                      KEPT
    IEF285I   VOL SER NOS= CBLDEV.
(7) IEF285I   CVS.OLIB                                      KEPT
    IEF285I   VOL SER NOS= CBLDEV.
    IEF285I   CVS.OLIB                                      KEPT
    IEF285I   VOL SER NOS= CBLDEV.
    IEF285I   SYS76237.T091056.RA000.TESTRUN.R0000001       DELETED
    IEF285I   VOL SER NOS= 222222.
    IEF285I   JES2.J0800659.S00105                          SYSOUT
    IEF373I STEP /LKED    / START 76237.0911
    IEF374I STEP /LKED    / STOP  76237.0911 CPU   0MIN 01.50SEC VIRT   132K SYS   204K
```

Figure 91. Linkage Editor Output Showing Module Map and Cross-Reference List

1. <u>The job control statements</u>. These statements are listed because MSGLEVEL=(1,1) is specified on the JOB statement for this job, shown in Figure 87.

2. <u>Allocation messages from the job scheduler</u>. These messages provide information about the device allocation for the data sets in the job step. For example, the message

   IEF2371 230 ALLOCATED TO SYSUT1

   indicates that the data set for SYSUT1 has been assigned to the device 230.

3. <u>Linkage editor informative message</u>. This message lists the PARM options that were specified.

4. <u>Linkage editor informative message</u>. This is a disposition message describing the disposition of the load module.

   A. Name of the load module specified in the DSNAME parameter of the SYSLMOD DD statement

   B. Text of message

5. Module map. The module map is listed when either the XREF or the MAP option is specified in linkage editor processing. The module map shows all control sections in the output module and all entry names in each control section. The control sections are arranged in ascending order according to their assigned origins. All entry names are listed below the control section in which they are defined. Each COBOL program is a control section, and any COBOL library subroutine is a separate control section (except as noted under segmentation).

   A. Control section. Under this heading the name, origin, and length of each control section is listed.
   Name. The name of the control section. This name is the PROGRAM-ID name in the main COBOL program or a called program. Each control section that is obtained from a library by an automatic library call is indicated by an asterisk.
   Origin. The relative origin in hexadecimal notation.
   Length. The number of bytes in each control section in hexadecimal notation.

   B. Entry. The entry names within each control section and their relative location. A called program may have more than one entry point. For a called COBOL program, the entry points are the same as the names specified by the ENTRY statements in the source program.

   C. Entry address. The relative address of the instruction with which processing of the module begins. It will always be INIT1 if the COBOL program is the main program of the load module.

   D. Total length. The total number of bytes, in hexadecimal notation, of the load module. It is the sum of the lengths of all control sections.

6. Cross reference list. The cross reference list, as well as a module map, is listed if the XREF option is specified. The MAP and XREF options should not be specified together. The cross reference list provides the following information:

   A. Location. The relative location in the program where another program is called.

   B. Symbol reference. The name of the entry point of the called program.

   C. In control section. The control section that contains the entry point.

   For example, 440 is the location where a COBOL subroutine is called. ILBOSRV1 is the entry point of the called program. ILBOSRV0 is the control section that contains the entry point ILBOSTP1.

   If XREF is specified, the cross reference list appears before the Entry Address.

7. Disposition messages from the job scheduler. These messages contain information about the disposition of the data sets.

Comments on the Module Map and Cross Reference List

The severity of linkage editor diagnostic messages may affect the production of the module map and the cross reference list.

Since various processing options will affect the structure of the load module, the text of the module map and cross reference list will sometimes provide additional information. For example, the load module may have an overlay structure. In this case, a module map will be listed for each segment in the overlay structure. The cross reference list is the same as that previously discussed, except that segment numbers also are listed to indicate the segment in which each symbol appears.

Listing the Linkage Editor Control Statements: If the LIST option is specified, linkage editor control statements, such as OVERLAY and LIBRARY, are listed.

Linkage Editor Messages

The linkage editor generates two types of messages: module disposition messages and error/warning messages. A description of the messages can be found in the publication OS/VS Linkage Editor and Loader. A complete list of the

error/warning messages is included in the publication OS/VS Message Library: Linkage Editor and Loader Messages.

LOADER OUTPUT

Loader output consists of a collection of diagnostic and error messages, and, if MAP is specified, a storage map of the loaded program. The output data set, SYSLOUT is sequential and blocked as specified by the user in the DCB. For better performance, the user can also specify the number of buffers to be allocated.

Diagnostic messages include a loader heading and a list of options requested by the user. The error messages, identifying the source of error, will be written when the error is detected. After processing is complete, an explanation of the error will be written. A complete list of loader error messages is found in the publication OS/VS Message Library: Linkage Editor and Loader Messages.

The map includes the name and absolute address for each control section and entry point defined in the program. It is written on SYSLOUT concurrently with input processing so it appears in order of input ESD items. The total size and storage extent also are included. Figure 92 is an example of a module map.

COBOL LOAD MODULE EXECUTION OUTPUT

The output generated by program execution (in addition to data written in program output files) can include:

- Data displayed on the console, or on the printer
- Cards
- Messages to the operator
- System informative messages
- System diagnostic message
- A system dump
- Debugging information

Note: If a program ends abnormally and one of the options FLOW, STATE, or SYMDMP is in effect and the SYSDBOUT DD card has been included, debugging information appears in the program listing (see the chapter entitled "Symbolic Debugging Features").

A dump as well as system diagnostic messages are generated automatically if a program contains errors that cause abnormal termination.

Note: If a COBOL program abnormally terminates, then a formatted dump is provided for all COBOL programs compiled with the SYMDMP option which could include the abnormally terminating program and its callers, up to and including the main program. For a discussion of the SYMDMP option as well as of other COBOL symbolic debugging options, see the chapter entitled "Symbolic Debugging Features."

Figure 93 shows an example of output from the execution job step. The following text is an explanation of the illustration.

1. The job control statements. These statements are listed because MSGLEVEL=(1,1) is specified in the JOB statement for this job.
2. The job allocation messages from the job scheduler. These messages indicate the device that is allocated for each data set defined for the job step.
3. Disposition messages from the job scheduler. These messages are contained in the OS/VS message library.
4. Program output on printer. The results of execution of the TRACE and EXHIBIT NAMED statements appear on program listing.
5. Console output. Data is printed on console as a result of execution of DISPLAY UPON CONSOLE.

```
                                        VS LOADER

OPTIONS USED - PRINT,MAP,LET,CALL,RES,NOTERM,SIZE=98304,NAME=**GO

     NAME  TYPE  ADDR        NAME  TYPE  ADDR        NAME  TYPE  ADDR        NAME  TYPE  ADDR        NAME  TYPE  ADDR

  TESTRUN    SD  60010
IEW1141   INCLUDE SYSLIB(ILBODSP)
  ILBOCOM0*  SD  60940     ILBOCOM *  LR  60940     ILBOSRV *  SD  60A00     ILBOSRV0*  LR  60A3A     ILBOSR3 *  LR  60A3A
  ILBOSR5 *  LR  60A3A     ILBOSR  *  LR  60A3A     ILBOSRV1*  LR  60A3E     ILBOSTP1*  LR  60A3E     ILBOST  *  LR  60A42
  ILBOSTP0*  LR  60A42     ILBODSP *  SD  60DC0     ILBODSP0*  LR  60DC2     ILBODSS0*  LR  60DC2     ILBOEXT *  SD  615A0
  ILBOEXT0*  LR  615A2     ILBODBG *  SD  615C0     ILBODBG0*  LR  615C2     ILBODBG1*  LR  615C6     ILBODBG2*  LR  615CA
  ILBODBG3*  LR  615CE     ILBODBG4*  LR  615D2     ILBODBG5*  LR  615D6     ILBOFLW *  SD  621F8     ILBOFLW0*  LR  621FA
  ILBOFLW1*  LR  621FE     ILBOFLW2*  LR  62202     ILBOMSG *  SD  62678     ILBOMSG0*  LR  6267A     ILBOBEG *  SD  62778
  ILBOBEG0*  LR  6277A     ILBOCMM *  SD  627E8     ILBOCMM0*  LR  627EA     ILBOCMM1*  LR  627EE

  TOTAL LENGTH     2B68
  ENTRY ADDRESS   60010

IEW1141    WARNING - CARD RECEIVED NOT AN OBJECT RECORD
```

Figure 92. Module Map Format Example

```
IEF236I ALLOC. FOR TESTRUN  GO
IEF237I 232    ALLOCATED TO PGM=*.DD
IEF237I 235    ALLOCATED TO STEPLIB
IEF237I 250    ALLOCATED TO DD1
IEF237I 250    ALLOCATED TO SYSDBOUT
IEF237I 250    ALLOCATED TO SYSUDUMP
IEF237I 250    ALLOCATED TO SYSOUT
IEF237I 250    ALLOCATED TO SYSPUNCH
IEF237I 582    ALLOCATED TO SAMPLE
XXGO     EXEC PGM=*.LKED.SYSLMOD,COND=((5,LT,COE),(5,LT,LKED))          00800270
XXSTEPLIB  DD DSN=VSCBL1.LIB,DISP=SHR,UNIT=2314,VOL=SER=DB143           00800280
XXDD1      DD DSN=&SYMDBG,DISP=(OLD,DELETE)                             00800290
//GO.SYSDBOUT  DD  SYSOUT=G
X/SYSDBOUT DD SYSOUT=A
//GO.SYSUDUMP  DD  SYSOUT=G                                             00800300
X/SYSUDUMP DD SYSOUT=A
//GO.SYSOUT  DD  SYSOUT=G                                               00800310
X/SYSOUT   DD SYSOUT=A
XXSYSPUNCH DD SYSOUT=E                                                  00800320
//GO.SAMPLE DD UNIT=2400,LABEL=(,NL),DISP=(NEW,DELETE)                  00800330
```

```
WORK-RECORD = A 0001 NYC Z
WORK-RECORD = B 0002 NYC 1
WORK-RECORD = C 0003 NYC 2
WORK-RECORD = D 0004 NYC 3
WORK-RECORD = E 0005 NYC 4
WORK-RECORD = F 0006 NYC Z
WORK-RECORD = G 0007 NYC 1
WORK-RECORD = H 0008 NYC 2
WORK-RECORD = I 0009 NYC 3
WORK-RECORD = J 0010 NYC 4
WORK-RECORD = K 0011 NYC Z
WORK-RECORD = L 0012 NYC 1
WORK-RECORD = M 0013 NYC 2
WORK-RECORD = N 0014 NYC 3
WORK-RECORD = O 0015 NYC 4
WORK-RECORD = P 0016 NYC Z
WORK-RECORD = Q 0017 NYC 1
WORK-RECORD = R 0018 NYC 2
WORK-RECORD = S 0019 NYC 3
WORK-RECORD = T 0020 NYC 4
WORK-RECORD = U 0021 NYC Z
WORK-RECORD = V 0022 NYC 1
WORK-RECORD = W 0023 NYC 2
WORK-RECORD = X 0024 NYC 3
WORK-RECORD = Y 0025 NYC 4
WORK-RECORD = Z 0026 NYC Z
```

```
A 0001 NYC 0
B 0002 NYC 1
C 0003 NYC 2
D 0004 NYC 3
E 0005 NYC 4
F 0006 NYC 0
G 0007 NYC 1
H 0008 NYC 2
I 0009 NYC 3
J 0010 NYC 4
K 0011 NYC 0
L 0012 NYC 1
M 0013 NYC 2
N 0014 NYC 3
O 0015 NYC 4
P 0016 NYC 0
Q 0017 NYC 1
R 0018 NYC 2
S 0019 NYC 3
T 0020 NYC 4
U 0021 NYC 0
V 0022 NYC 1
W 0023 NYC 2
X 0024 NYC 3
Y 0025 NYC 4
Z 0026 NYC 0
```

```
IEF142I - STEP WAS EXECUTED - COND CODE 0000
IEF285I   SYS74163.T204211.RV000.TESTRUN.LMODLIB       PASSED
IEF285I   VOL SER NOS= 231402.
IEF285I   VSCBL1.LIB                                   KEPT
IEF285I   VOL SER NOS= DB143 .
IEF285I   SYS74163.T204211.RV000.TESTRUN.SYMDBG        DELETED
IEF285I   VOL SER NOS= 333001.
IEF285I   SYS74163.T204211.SV000.TESTRUN.R0000011      SYSOUT
IEF285I   VOL SER NOS= 333001.
IEF285I   SYS74163.T204211.SV000.TESTRUN.R0000012      DELETED
IEF285I   VOL SER NOS= 333001.
IEF285I   SYS74163.T204211.SV000.TESTRUN.R0000013      SYSOUT
IEF285I   VOL SER NOS= 333001.
IEF285I   SYS74163.T204211.SV000.TESTRUN.R0000014      DELETED
IEF285I   VOL SER NOS= 333001.
IEF285I   SYS74163.T204211.RV000.TESTRUN.R0000015      DELETED
IEF285I   VOL SER NOS= L00001.
IEF285I   SYS74163.T204211.RV000.TESTRUN.S0000016      SYSIN
IEF285I   VOL SER NOS= 333001.
IEF285I   SYS74163.T204211.RV000.TESTRUN.S0000016      DELETED
IEF285I   VOL SER NOS= 333001.
IEF373I STEP /GO      / START 74163.2056
IEF374I STEP /GO      / STOP  74163.2109 CPU    0MIN 09.47SEC STOR VIRT  64K
IEF285I   SYS74163.T204211.RV000.TESTRUN.LMODLIB       DELETED
IEF285I   VOL SER NOS= 231402.
IEF375I  JOB /TESTRUN / START 74163.2046
IEF376I  JOB /TESTRUN / STOP  74163.2109 CPU   0MIN 24.47SEC
```

Figure 93. Execution Job Step Output

REQUESTS FOR OUTPUT

1. The programmer can request data to be displayed by using the DISPLAY statement and including the following in the job control procedure:

   ```
   //SYSOUT    DD  SYSOUT=A
   ```

2. Message to the operator can also be displayed on the console when requested in the source program (DISPLAY UPON CONSOLE).

3. The programmer can request debugging information in case of an abnormal termination by specifying FLOW, STATE, or SYMDMP and including the following in the job control procedure:

   ```
   //SYSDBOUT  DD  SYSOUT=A
   ```

   In addition, the SYSUT5 DD statement is required for compilation and the SYSDBG DD statement is required at execution time.

   The following DD statement must be used to make the COBOL library module ILBOBEG available at execution time.

   ```
   //STEPLIB DD DSN=subr-libname,DISP=SHR
   ```

   If an error message is printed by the debugging modules, the COBOL library module ILBOBEG is loaded dynamically from the subroutine library. This module is not link edited into the COBOL object program.

   By specifying a //SYSDTERM DD card in addition to the //SYSDBOUT DD card, dynamic dump output will be written onto SYSDTERM while the abend dump output will go to SYSDBOUT.

Note: The TSO programmer should assign SYSDTERM to the terminal since dynamic dump output is interruptable. SYSDBOUT should be assigned to a direct access data set which could be listed at the terminal after the ABEND is complete.

4. The programmer can request a full dump, in case his program is terminated abnormally, by including one of the following in the job control procedure:

   either

   ```
   //SYSUDUMP DD SYSOUT=A
   ```
   or
   ```
   //SYSABEND  DD  SYSOUT=A
   ```

   Both SYSUDUMP and SYSABEND will usually produce the following when an abnormal termination occurs: some system control blocks, work areas (in addition to user modules), and system modules, and--under VS2--GTF trace records if that system service is running. SYSABEND normally also produces the system nucleus.

Dumps and debugging facilities are further explained in "Program Checkout."

OPERATOR MESSAGES

The COBOL load module may issue operator messages. A complete list of these messages and required operator responses can be found in "Appendix K: Diagnostic Messages." MCS considerations are discussed there also.

SYSTEM OUTPUT

Informative and diagnostic messages may appear in the listing during execution of any job step. Further information about system diagnostics is found in the appropriate OS/VS Message Library publication. COBOL messages and associated documentation for this compiler can be displayed by running the ERRMSG program described earlier in "Output."

Each of these messages contains an identification code in the first three columns of the message to indicate the portion of the operating system that generated the message. Figure 94 lists these codes, along with an identification of each.

Code	Identification
ICE	A message from the Sort program.
IEA	A message from the supervisor.
IEC	A message from data management.
IEF	A message from the job scheduler.
IKF	A message from the COBOL compiler.
IER	A message from the Sort program.
IEW	A message from the linkage editor.
IGH	A message from the Sort program.

Figure 94. System Message Identification Codes

A programmer using the COBOL compiler under OS/VS has several methods available to him for testing and debugging his programs or revising them for increased efficiency of operation.

The lister feature provides the COBOL programmer with reformatted source code that contains complete cross-reference information at the source level.

The syntax-checking options can be specified to save programmer and machine time while checking the source statements for syntax errors.

The COBOL debugging language can be used by itself or in conjunction with other COBOL statements. A dump can also be used for program checkout. For a discussion of the COBOL symbolic debugging options, see the chapter entitled "Symbolic Debugging Features."

The execution frequency statistics option, COUNT, facilitates testing, debugging, and optimizing by causing verb execution counts to be generated at the end of the execution of a compiled program. The statistics provide the COBOL programmer with information that aids user program optimization by identifying heavily-used portions of the COBOL source program. They are also useful to the programmer in debugging by providing verification that all parts of a program have been executed.

Note: The program product IBM OS COBOL Interactive Debug (Program Number 5734-CB4) enables the user to debug his COBOL programs from a terminal under TSO, the Time Sharing Option of the Operating System. With its own debugging command language, Interactive Debug gives the user several object-time debugging capabilities including: starting and stopping program execution at selected points, altering the logical flow of program execution, manipulating data, and displaying such things as selected source statements, file status and content, and various kinds of program execution traces. How to accomplish these operations is described fully in IBM OS COBOL Interactive Debug Terminal User's Guide and Reference.

Specifying the TEST compiler option allows a COBOL program to be debugged by the Interactive Debug user. A program that is compiled without the TEST option is unacceptable to the Interactive Debug command processor.

TEST has the following effects on other compiler options: TEST overrides FLOW, STATE, SYMDMP and COUNT. However, BATCH overrides TEST. TEST is also overridden if the source program contains USE FOR DEBUGGING and WITH DEBUGGING MODE statements.

TEST requires the debug file, SYSUT5.

LISTER FEATURE

When the user requests the lister feature, each COBOL statement is begun on a new line and is indented in such a way as to make the logic of the program readily apparent by highlighting level numbers, nested IF statements, etc. For complete information, see the chapter "Lister Feature."

SYNTAX-CHECKING ONLY COMPILATION

The compiler checks the source text for syntax errors and then generates the appropriate error messages, but does not bother to produce object code. With the syntax-checking feature, the programmer can request a compilation either conditionally, with object code produced only if no messages or just W- or C-level messages are generated, or unconditionally, with no object code produced regardless of message level.

For a discussion of the syntax-checking options, SYNTAX and CSYNTAX, see the section "Options for the Compiler" under "Job Control Procedures."

DEBUGGING LANGUAGE

The COBOL debugging language is designed to aid the COBOL programmer in producing an error-free program in the shortest possible time. The sections that follow discuss the use of the various types of debugging language and other methods of program checkout.

DEBUGGING LINES

The user can include debugging lines (any COBOL statement with a D in column 7) in his program to assist in locating logic errors. Through inclusion of the WITH DEBUGGING MODE source clause (essentially a compile-time switch), the statements are made part of the object code and will be executed in line with the rest of the program. Removal of the WITH DEBUGGING MODE clause causes debugging lines to be treated as comments only; they will not be executed. A program containing debugging lines must be syntactically correct in both these modes. (The execution-time option DEBUG/NODEBUG has no control over debugging lines; it only affects USE FOR DEBUGGING declaratives--as explained below.)

DECLARATIVE PROCEDURES--USE FOR DEBUGGING

The USE FOR DEBUGGING feature provides the user with the ability to create his own procedures to examine the internal status of his program during its execution. The USE FOR DEBUGGING statement identifies which program elements it wishes to monitor. COBOL then gives the associated procedure control when these elements are referenced during execution. The procedure also is given access to the DEBUG-ITEM special register, which has been automatically filled with the pertinent current status information.

The USE FOR DEBUGGING procedures can be controlled by two switches: the WITH DEBUGGING MODE source clause for compile-time, and the DEBUG/NODEBUG option for execution-time. WITH DEBUGGING MODE indicates that the procedures are to be compiled as executable code; if WITH DEBUGGING MODE is omitted, the procedures are treated only as comments. Specification of the DEBUG option at execution time indicates that the procedures compiled into the code are in fact bo be executed; if NODEBUG is specified, the procedures are bypassed.

The general rules for USE FOR DEBUGGING declarative procedures and the DEBUG-ITEM special register can be found in IBM VS COBOL for OS/VS. The following considerations also apply:

1. Including USE FOR DEBUGGING declarative procedures and a WITH DEBUGGING MODE clause precludes the use of the SYMDMP and TEST options. If SYMDMP and/or TEST are specified in such a case, they will be rejected.

2. The DEBUG-ITEM special register is variable in length, and depends on the size of the character string it is to contain. This length cannot exceed 32K.

3. A SEARCH or SEARCH ALL statement that refers to identifier-1 as the table to be searched will not result in an invocation of the USE FOR DEBUGGING declarative procedure. If identifier-1 is referred to elsewhere in the statement (for example, as an operand of a WHEN condition), the associated declarative will be invoked.

4. When identifier-1 is the object of an OCCURS DEPENDING ON clause, the contents of the DEBUG-CONTENTS subfield of DEBUG-ITEM will be unpredictable if all three of the following conditions are true:

   a) identifier-1 is changed in the same statement as the data item whose size and/or location is affected by that change, and

   b) the change occurs following the reference to the data item, and

   c) the data item is not subscripted and/or indexed.

5. If identifier-1 appears in a MOVE statement and is subscripted or indexed, and if either of the following is also true:

   a) the subscript or index is changed in the MOVE prior to the reference to identifier-1, or

   b) an OCCURS DEPENDING ON object that affects the size or address of identifier-1 is changed in the MOVE prior to the reference to identifier-1,

   then the DEBUG-ITEM values will reflect identifier-1's address and size just prior to execution of that MOVE.

6. Procedures performed from a USE FOR DEBUGGING declarative will never cause invocation of another USE FOR DEBUGGING declarative.

7. A USE FOR DEBUGGING filename can only be a VSAM or QSAM file.

Figure 95 shows an elementary example of USE FOR DEBUGGING. The program is the same TESTRUN used several times elsewhere in this manual. Here it has been slightly modified through the addition of the

debugging phrase (encircled 1) and a simple declarative (encircled 2). In this example, the programmer wishes to temporarily trap certain input items (ALPHA D's and M's) and boost their index values by one so that they become E's and N's (encircled 3).

Notice that the DEBUG-CONTENTS special register allows the programmer to reference items without using subscripting. Notice also that the subscript value can be modified in the declarative section without affecting that value in the body of the program (i.e., DEBUG-SUB-1 is only a copy). By removing the WITH DEBUGGING MODE clause from the CONFIGURATION SECTION and recompiling, the programmer can disable the debugging declarative--even though the declarative statements are left in the source program.

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  TESTDBUG.
  AUTHOR. PROGRAMMER NAME.
  INSTALLATION.  PALO ALTO DEVELOPMENT CENTER.
  DATE WRITTEN.  AUGUST 8, 1976.
DATE-COMPILED. SEP 10,1976.
  REMARKS. THIS PROGRAM HAS BEEN WRITTEN AS A SAMPLE PROGRAM FOR
     COBOL USERS. IT CREATES AN OUTPUT FILE AND READS IT BACK AS
     INPUT.

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
(1)  SOURCE-COMPUTER.  IBM-370 WITH DEBUGGING MODE.
  OBJECT-COMPUTER.  IBM-370-168.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT FILE-1 ASSIGN TO UT-2400-S-SAMPLE.
    SELECT FILE-2 ASSIGN TO UT-2400-S-SAMPLE.

DATA DIVISION.
FILE SECTION.
FD  FILE-1
    LABEL RECORDS ARE OMITTED
    BLOCK CONTAINS 100 CHARACTERS
    RECORD CONTAINS 20 CHARACTERS
    RECORDING MODE IS F
    DATA RECORD IS RECORD-1.
  01 RECORD-1.
    02 FIELD-A PICTURE IS X(20).
FD  FILE-2
    LABEL RECORDS ARE OMITTED
    BLOCK CONTAINS 5 RECORDS
    RECORD CONTAINS 20 CHARACTERS
    RECORDING MODE IS F
    DATA RECORD IS RECORD-2.
  01 RECORD-2.
    02 FIELD-A PICTURE IS X(20).

WORKING-STORAGE SECTION.
    77 KOUNT PICTURE S99 COMP SYNC.
    77 NOMBER  PICTURE S99 COMP SYNC.
  01 FILLER.
    02 ALPHABET PICTURE X(26) VALUE "ABCDEFGHIJKLMNOPQRSTUVWXYZ".
    02 ALPHA REDEFINES ALPHABET PICTURE X OCCURS 26 TIMES.
    02 DEPENDENTS PICTURE X(26)  VALUE "01234012340123401234012340
    "0".
    02 DEPEND REDEFINES DEPENDENTS PICTURE X OCCURS 26 TIMES.
  01 WORK-RECORD.
    02 NAME-FIELD  PICTURE X.
    02 FILLER PICTURE X VALUE IS SPACE.
    02 RECORD-NO PICTURE 9999.
    02 FILLER PICTURE X VALUE IS SPACE.
    02 LOCATION PICTURE AAA VALUE IS "NYC".
    02 FILLER PICTURE X VALUE IS SPACE.
    02 NO-OF-DEPENDENTS PICTURE XX.
    02 FILLER PICTURE X(7) VALUE IS SPACES.
PROCEDURE DIVISION.
DECLARATIVES.
DEBUG-SECTION SECTION.
(2)  USE FOR DEBUGGING ON ALL REFERENCES OF ALPHA.
    IF DEBUG-CONTENTS = "D" OR "M"
        ADD 1 TO DEBUG-SUB-1
        MOVE ALPHA(DEBUG-SUB-1) TO NAME-FIELD.
END DECLARATIVES.
BEGIN.
    NOTE THAT THE FOLLOWING OPENS THE OUTPUT FILE TO BE CREATED
    AND INITIALIZES COUNTERS.
STEP-1. OPEN OUTPUT FILE-1. MOVE ZERO TO KOUNT NOMBER.
    NOTE THAT THE FOLLOWING CREATES INTERNALLY THE RECORDS TO BE
    CONTAINED IN THE FILE, WRITES THEM ON TAPE, AND DISPLAYS
    THEM ON THE CONSOLE.
STEP-2. ADD 1 TO KOUNT, ADD 1 TO NOMBER, MOVE ALPHA (KOUNT) TO
    NAME-FIELD.
    MOVE DEPEND (KOUNT) TO NO-OF-DEPENDENTS.
    MOVE NOMBER TO RECORD-NO.
STEP-3. DISPLAY WORK-RECORD UPON CONSOLE. WRITE RECORD-1 FROM
    WORK-RECORD.
STEP-4. PERFORM STEP-2 THRU STEP-3 UNTIL KOUNT IS EQUAL TO 26.
    NOTE THAT THE FOLLOWING CLOSES OUTPUT AND REOPENS IT AS
    INPUT.
STEP-5. CLOSE FILE-1. OPEN INPUT FILE-2.
    NOTE THAT THE FOLLOWING READS BACK THE FILE AND SINGLES OUT
    EMPLOYEES WITH NO DEPENDENTS.
STEP-6. READ FILE-2 RECORD INTO WORK-RECORD AT END GO TO STEP-8.
STEP-7. IF NO-OF-DEPENDENTS IS EQUAL TO "0" MOVE "Z" TO
    NO-OF-DEPENDENTS. EXHIBIT NAMED WORK-RECORD. GO TO
    STEP-6.
STEP-8. CLOSE FILE-2.
    STOP RUN.
```

(3)

```
WORK-RECORD = A 0001 NYC Z
WORK-RECORD = B 0002 NYC 1
WORK-RECORD = C 0003 NYC 2
WORK-RECORD = D 0004 NYC 3
WORK-RECORD = E 0005 NYC 4
WORK-RECORD = F 0006 NYC Z
WORK-RECORD = G 0007 NYC 1
WORK-RECORD = H 0008 NYC 2
WORK-RECORD = I 0009 NYC 3
WORK-RECORD = J 0010 NYC 4
WORK-RECORD = K 0011 NYC Z
WORK-RECORD = L 0012 NYC 1
WORK-RECORD = M 0013 NYC 2
WORK-RECORD = N 0014 NYC 3
WORK-RECORD = O 0015 NYC 4
WORK-RECORD = P 0016 NYC Z
WORK-RECORD = Q 0017 NYC 1
WORK-RECORD = R 0018 NYC 2
WORK-RECORD = S 0019 NYC 3
WORK-RECORD = T 0020 NYC 4
WORK-RECORD = U 0021 NYC Z
WORK-RECORD = V 0022 NYC 1
WORK-RECORD = W 0023 NYC 2
WORK-RECORD = X 0024 NYC 3
WORK-RECORD = Y 0025 NYC 4
WORK-RECORD = Z 0026 NYC Z
```

Figure 95. Program with USE FOR DEBUGGING.

TRACE, EXHIBIT, AND ON

Three additional debugging language statements are TRACE, EXHIBIT, and ON. Any one of these statements can be used as often as necessary. They can be interspersed throughout a COBOL source program, or they can be in a packet in the input stream to the compiler.

Program debugging statements may not be desired after testing is completed. A debugging packet can be removed after testing. This allows elimination of the extra object program coding generated for the debugging statements.

The output produced by the TRACE and EXHIBIT statements is listed on the system logical output device (SYSOUT). If these statements are used, the SYSOUT DD statement must be specified in the execution time job step.

The following discussions describe ways to use the debugging language.

Following the Flow of Control

The READY TRACE statement causes the compiler generated card numbers for each paragraph name to be listed on the system output unit when control passes to that point. The output appears as a list of card numbers. If the VERB option is in effect during compilation, paragraph-names rather than card numbers will be displayed.

To reduce execution time, a trace can be stopped with a RESET TRACE statement. The READY TRACE/RESET TRACE combination is helpful in examining a particular area of the program. The READY TRACE statement can be coded so that the trace begins before control passes to that area. The RESET TRACE statement can be coded so that the trace stops when the program has passed the area. The two trace statements can be used together where the flow of control is difficult to determine, e.g., with a series of PERFORM statements or with nested conditionals.

Another way to control the amount of tracing, so that it is done conditionally, is to use the ON statement with the TRACE statement. When the COBOL compiler encounters an ON statement, it sets up a mechanism such as a counter that is incremented during execution whenever control passes through the ON statement. For example, if an error occurs when a specific record is processed, the ON statement can be used to isolate the problem record. The statement should be placed where control passes only once for each record that is read. When the contents of the counter equal the number of the record (as specified in the ON statement), a trace can be taken on that record. The following example shows a way in which the processing of the 200th record could be selected for a TRACE statement.

```
Col.
1       8
--------------------------------------------------------------
        RD-REC.
        .
        .
        .
DEBUG   RD-REC.
        PARA-NM-1.      ON 200 READY TRACE.
                        ON 201 RESET TRACE.
```

If the TRACE statement were used without the ON statement, the processing of every record would be traced.

A common program error could be either (1) failing to break a loop, or (2) unintentionally creating a loop. If many iterations of the loop are required before it can be determined that there is a program error, the ON statement can be used to initiate a trace only after the expected number of iterations has been completed.

Notes:

1. If an error occurs in an ON statement, the diagnostic message may refer to the previous statement number.

2. When READY TRACE encounters a paragraph that is repetitively executed, it does not write out a separate record for each individual execution of the paragraph. Instead, it simplifies its output by keeping an internal count, and writing out a single summary record only when the paragraph iteration finally finishes. This record identifies the number of times the paragraph was executed. READY TRACE output is built to contain multiple paragraph entries on each print line.

3. In the event the paragraph iteration is a nonending loop, READY TRACE output does not include the current print line being built. Therefore, READY TRACE output never identifies the paragraph as having been entered at all. In the event of an ABEND, the last print line of READY TRACE data may not be printed. When either a non-ending loop or abnormal ending of a task occurs, the current print

line is available in the current SYSOUT buffer.

Displaying Data Values during Execution

A programmer can display the value of a data item during program execution by using the EXHIBIT statement. The three forms of this statement display (1) the names and values of the identifiers or nonnumeric

literals listed in the EXHIBIT statement (EXHIBIT NAMED) whenever the statement is encountered during execution, (2) the values of the items listed in this statement only if the value has changed since the last execution (EXHIBIT CHANGED) and (3) the names and values of the items listed in the statement only if the values have changed since the previous execution (EXHIBIT CHANGED NAMED). The first time such a statement is executed, all values are considered changed and are displayed.

Note: The combined total length of all items displayed with EXHIBIT CHANGED and EXHIBIT CHANGED NAMED cannot exceed 32,767 bytes. The length of any one operand must be less than or equal to 256 bytes. The length of a "NAME" must be less than or equal to 120 characters.

Data can be used to check the accuracy of the program. For example, the programmer can display specified fields from records, work the calculations himself, and compare his calculations with the output from his program. The coding for a payroll problem could be:

```
Col.
1       8
-------------------------------------------------------
            .
            .
            .

        GROSS-PAY-CALC.
             COMPUTE GROSS-PAY =
             RATE-PER-HOUR * (HRSWKD
             + 1.5 * OVERTIMEHRS).
        NET-PAY-CALC.
            .
            .
            .

DEBUG   NET-PAY-CALC
        SAMPLE-1. ON 10 AND
             EVERY 10 EXHIBIT NAMED
             RATE-PER-HOUR, HRSWKD,
             OVERTIMEHRS, GROSS-PAY.
```

This coding will cause the values of the four fields to be listed for every tenth data record before net pay calculations are made. The output could appear as:

```
RATE-PER-HOUR = 4.00 HRSWKD = 40.0
     OVERTIMEHRS = 0.0 GROSS-PAY = 160.00
RATE-PER-HOUR = 4.10 HRSWKD = 40.0
     OVERTIMEHRS = 1.5 GROSS-PAY = 173.23
RATE-PER-HOUR = 3.35 HRSWKD = 40.0
     OVERTIMEHRS = 0.0 GROSS-PAY = 134.00
```

Note: Decimal points are included in this example for clarity, but actual printouts depend on the data description in the program.

The preceding is an example of checking at regular intervals (every tenth record). A check of any unusual conditions can be made by using various combinations of COBOL statements in the debug packet. For example:

```
IF OVERTIMEHRS GREATER THAN 2.0
   EXHIBIT NAMED PAYRCDHRS
```

In connection with the previous example, this statement could cause the entire pay record to be displayed whenever an unusual condition (overtime exceeding two hours) is encountered.

The EXHIBIT CHANGED statement also can be used to monitor conditions that do not occur at regular intervals. The values of the items are listed only if the value has changed since the last execution of the statement. For example, suppose the program calculates postage rates to various cities. The flow of the program might be as shown in Figure 96.

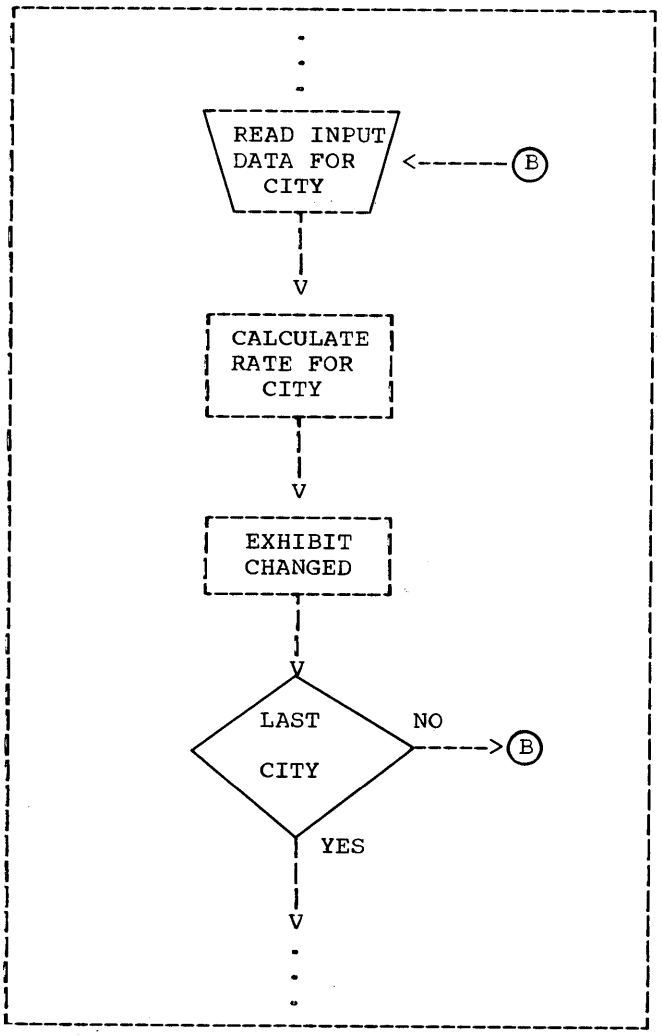


Figure 96. Example of Program Flow

The EXHIBIT CHANGED statement in the program could be:

```
EXHIBIT CHANGED STATE CITY RATE
```

The output from the EXHIBIT CHANGED statement could appear as:

```
01  01  10
    02  15
    03
    04  10
02  01
    02  20
    03  15
    04
03  01  10
     .
     .
     .
```

The first column contains the code for a state, the second column contains the code for a city, and the third column contains the code for the postage rate. The value of an item is listed only if it is changed since the previous execution. For example, since the postage rate to city 02 and 03 in state 01 are the same, the rate is not printed for city 03.

The EXHIBIT CHANGED NAMED statement lists the name of the data item and the value of that item if the value has changed. For example, the program might calculate the cost of various methods of shipping to different cities. After the calculations are made, the following statement could be in the program:

```
EXHIBIT CHANGED NAMED STATE CITY RAIL
   BUS TRUCK AIR
```

The output from this statement could appear as:

```
STATE = 01 CITY = 01 RAIL = 10
   BUS = 14 TRUCK = 12 AIR = 20

CITY = 02

CITY = 03 BUS = 06 AIR = 15

CITY = 04 RAIL = 30 BUS = 25
   TRUCK = 28 AIR = 34

STATE = 02 CITY = 01 TRUCK = 25

CITY = 02 TRUCK = 20 AIR = 30
     .
     .
     .
```

Note that the name of the item and its value are listed only if the value has changed since the previous execution.

Testing a Program Selectively

A debug packet allows the programmer to select a portion of the program for testing. The packet can include test data and can specify operations the programmer wants performed. When the testing is completed, the packet can be removed. The flow of control can be selectively altered by the inclusion of debug packets, as shown in Figure 97.

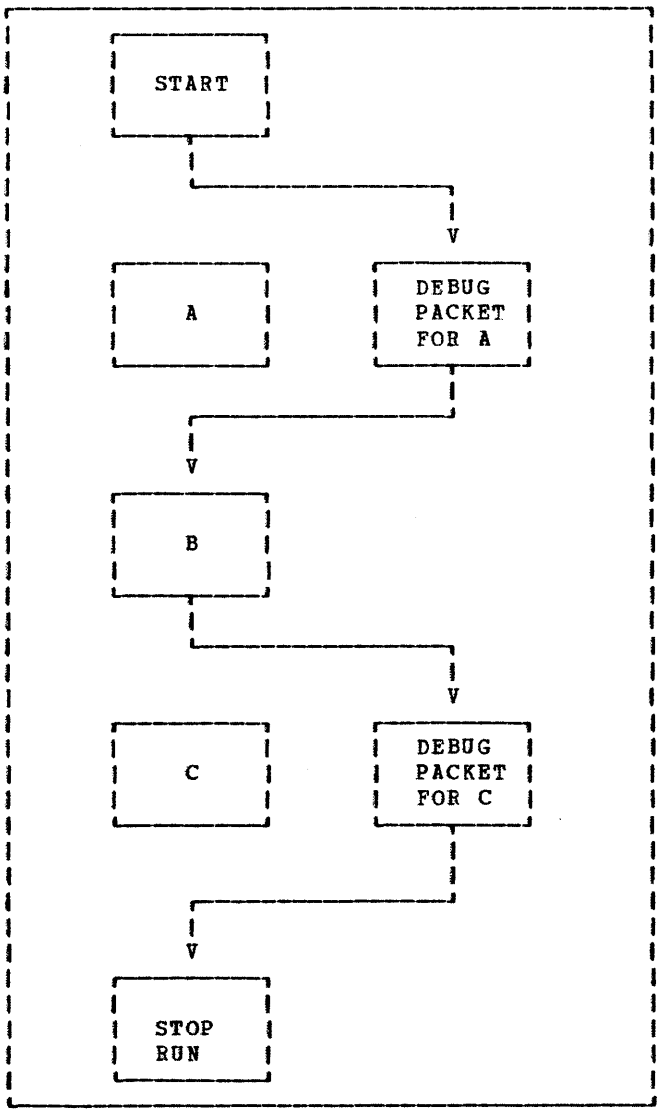


Figure 97. Selective Testing of B

In this program, A creates data, B processes it, and C prints it. The debug packet for A simulates test data. It is first in the program to be executed. In the packet, the last statement is GO TO B,

which permits A to be bypassed. After B is executed with the test data, control passes to the debug packet for C, which contains a GO TO statement that transfers control to the end of the program, bypassing C.

Testing Changes and Additions to Programs

If a program runs correctly but changes or additions can make it more efficient, a debug packet can be used to test changes without modifying the original source program.

If the changes to be incorporated are in the middle of a paragraph, the entire paragraph, with the changes included, must be written in the debug packet. The last statement in the packet should be a GO TO statement that transfers control to the next procedure to be executed.

There are usually several ways to perform an operation. Alternative methods can be tested by putting them in debug packets.

The source program library facility can be used for program checkout by placing a source program in a library (see "Libraries"). Changes or additions to the program can be tested by using the BASIS card and any number of INSERT and DELETE cards. Such changes or additions remain in effect only for the duration of the run.

ABEND DUMPS

If a serious error occurs during execution of a program, the job may be abnormally terminated; any remaining steps are bypassed, and a dump is generated. The programmer can use the dump for program checkout. (However, any pending transfers to an external device may not be completed. For example, if a READY TRACE statement is in effect when the job is abnormally terminated, the last card number may not appear on the external device.) In cases where the abnormal termination does not go to completion, a dump is not produced. This situation may cause duplicate name definition when the next job is run, and is discussed at the end of this section.

If a SYSUDUMP DD statement has been included in the execution-time job step, the system will provide the programmer with a printout, in hexadecimal and EBCDIC format, of main storage. Those areas occupied by the problem program and its data at the time the error occurred, will be included. This printout is called an abnormal termination dump and in VS1 is identified by the heading

*** ABDUMP REQUESTED ***

If a SYSABEND DD statement is specified, the contents of the nucleus is also printed.

If neither a SYSUDUMP nor a SYSABEND DD statement is included in the execution-time job step, or its specification has been destroyed, an indicative dump is produced. This dump does not contain a printout of main storage and is not given under OS/VS2.

All dumps include a completion code designating the condition that caused the termination. The completion code consists of a system code and a user code. Only one of the codes is nonzero. A nonzero system code indicates that the control program detected the error.

The COBOL programmer can request dynamic dumps via a compile-time option. The SYMDMP option, requested in the PARM parameter of the EXEC statement, produces a symbolic formatted dump of the data area of the object program if the program abnormally terminates. At execution time, the user can also request a dynamic dump at any point in the Procedure Division.

Notes:

- If a COBOL program abnormally terminates, then a formatted dump is produced for all COBOL programs compiled with the SYMDMP option which could include the abnormally terminating program and its callers, up to and including the main program.
- The explanation of the system-generated completion codes and a complete description of the dumps are contained in the appropriate Debugging Guide. For a discussion of the COBOL symbolic debugging options, see the chapter entitled "Symbolic Debugging Features."

USER-INITIATED DUMPS

The COBOL programmer can cause a dump at any pre-specified point in his program by coding a call to the library subroutine ILBOABN0, in the following manner:

```
77  COMP-CODE PIC S999 COMP.
      .
      .
      .
MOVE  nnn TO COMP-CODE.
```

```
CALL 'ILBOABNO' USING COMP-CODE.
```

The three-digit number nnn will become the identifier in the Unnn user abend code produced with the resultant dump.

ERRORS THAT CAN CAUSE A DUMP

Following is a discussion of some error conditions that can cause a program to be abnormally terminated and a dump to be listed.

Input/Output Errors

Errors can occur while a COBOL file is being processed. For example, during data transmission, an input/output error may occur that cannot be corrected. In some situations, this will result in the job being terminated (see Figures 52 and 53).

Referring to an input area (non-VSAM/QSAM) before both an OPEN and a READ statement are issued can cause unpredictable results, because base locator (BL) cells and registers are not properly initialized.

Another error that can cause termination is an attempt to read a file whose records are of a different size than those described in the source program. The section "Additional File Processing Information" contains more information about input/output errors.

Errors Caused by Invalid Data

Abnormal termination of a job occurs when a data item with an invalid format is processed in the Procedure Division.

Some of the program errors are:

1. A data item in the Working-Storage Section is not initialized before it is used, causing invalid data to be picked up.

2. An input file or received message contains invalid data or data incorrectly defined by its data description. For example, the contents of the sign position of an internal or external decimal data item in the file may be invalid. The compiler does not generate a test to check the sign position for a valid configuration before the item is used as an operand.

3. If a group item is moved to a group item and the subordinate data descriptions are incompatible, the new data in the receiving field may not match the corresponding data descriptions. (Conversion or editing is not performed in a move involving a group item.)

   Note: A Numeric class test, "IF Numeric", for signed data items, allows C, D, and F as valid signs. For external decimal items this includes X'C1' through X'C9' and X'D1' through X'D9' and X'F1' through X'F9' as valid last bytes. For internal decimal items, this includes X'1C' through X'9C', X'1D' through X'9D', and X'1F' through X'9F' as valid last bytes. Certain invalid numeric data items, such as EBCDIC A through R in the last byte of an external decimal numeric item, and EBCDIC <, *, %, @, (,), _, ', |, ¬, ?, and " in the last byte of an internal decimal item, are considered valid numeric items for an "IF Numeric" test.

4. The SIZE ERROR option is not specified for the COMPUTE statement and the result of the calculation is larger than the specified resultant COMPUTATIONAL data name. Using the result in a subsequent calculation might cause an error.

5. The SIZE ERROR option is not specified for a DIVIDE statement, and an attempt is made to divide by zero.

6. The USAGE specified for a redefining data item is different from the USAGE specified for the redefined item. An error results when the item is referred to by the wrong name for the current content.

7. A record containing a data item described by an OCCURS clause with the DEPENDING ON data-name option, may cause data items in the record to be affected by a change in the value of

data-name during the course of program execution. This may result in incorrectly described data. Additional information about how to correct this situation is included in "Programming Techniques."

8. The data description in the Linkage Section of a called program does not correctly describe the data defined in the calling program.

9. Blanks read into data fields defined as numeric generate an invalid sign.

10. Some common errors that occur when clearing group items in storage are:

    a. Moving ALL ZEROS to a group level item to clear several counters causes an invalid sign to be generated in all of the elementary fields except the lowest order field.

    b. Moving SPACES to a group level item will put invalid data in any numeric field in that group.

    c. Moving 0 to a group level item moves one zero and pads the rest of the fields with blanks.

11. Failure to initialize counters produces incorrect results. No initial values are generated by the compiler unless specifically instructed to do so with a VALUE clause. If such fields are defined as decimal, internal or external, invalid signs may result in addition to unpredictable initial values. If defined as binary, they will cause unpredictable results and, further, if used in subscripting, may exceed the range of the associated OCCURS clause and cause data to be fetched or stored erroneously. An addressing exception may occur if the uninitialized subscript generates a bad address.

12. Not testing to insure that a subscript or index does not exceed the range of the associated OCCURS clause may lead to fetching and storing data from and to some incorrect locations.

13. Failure to initialize an index produces incorrect results. No initial values are generated by the compiler unless a SET statement is executed. When indexing is then specified, the range of the OCCURS clause may be exceeded and cause data to be fetched or stored erroneously. An addressing exception may occur if the initialized index generates an address outside the range of the machine, or a protection exception if data is stored outside the partition of this program.

14. A subscript or index set at zero will address data outside the range of the table.

15. If either HIGH-VALUE or LOW-VALUE is moved to internal or external decimal fields and those fields are used for comparisons, computations, or subscripting, a data exception will occur. HIGH-VALUE and LOW-VALUE are the hexadecimal values X'FF' and X'00', respectively (unless these values have been altered by a user-defined collating sequence).

Other Errors

Additional I/O errors may occur that will result in an abnormal termination (these are listed below). For QSAM files, however, the user can employ the FILE STATUS clause to intercept many of these errors; his program can then identify and deal with them, and thus prevent the abend from occuring.

1. No DD statement is included for a file described in the source program and an attempt is made to access the file. When an OPEN statement for the file is executed, the system console message is written. The programmer can elect to direct the operator to continue processing his program, but any READ, WRITE, REWRITE, or START associated with the unlocated file will fail. (A READ for a missing optional file, however, will follow end-of-file processing rules.) A similar situation exists when a file is closed WITH LOCK and an attempt is made to reopen it.

2. A file is not opened and execution of a READ or WRITE statement for the file is attempted, or a MOVE to a record area in the file is attempted.

3. A GO TO statement, with no procedure name following it, is not properly initialized with an ALTER statement before the first execution of the GO TO statement.

4. Reference is made to an item in a file after end of data. This includes the use of the TERMINATE statement of the

Report Writer feature, if the CONTROL FOOTING, PAGE FOOTING, or REPORT FOOTING contain items that are in the file (e.g., SOURCE data-name, where data-name refers to an item in the file).

5. Block size for an F-format file is not an integral multiple of the record length.

6. In a blocked and/or multiple-buffered file, information in a record is unavailable after a WRITE.

7. A READ is issued for a data set referenced on a DD DUMMY statement. The AT END condition is sensed

immediately and any reference to a record in the data set produces unpredictable results.

8. A STOP RUN statement is executed before all files are closed.

9. A SORT did not execute successfully. The programmer may check SORT-RETURN.

10. An input/output statement is issued for a file after the AT END branch is taken, without closing and reopening the file.

11. A SEND or RECEIVE statement is issued when a message control program is not running.

12. A SEND or RECEIVE statement is issued for a QNAME (i.e., the "QNAME=" parameter of the DD card) that is unknown to the message control program.

In addition to errors that can result in an abnormal termination, errors in the source program can occur that cause parts of the program to be overlaid and the corresponding object code instructions to become invalid. If an attempt is then made to execute one of these instructions, an abnormal termination may result because the operation code of the instruction is invalid, the instruction results in a branch to an area containing invalid instructions, or the instruction results in a branch to an area outside the program, such as an address protected area.

Some COBOL source program errors that can cause this overlaying are:

1. Using a subscript whose value exceeds the maximum specified in the associated OCCURS clause.

2. Using a data-name as a counter whose value exceeds the maximum value valid for that counter.

SYSTEM COMPLETION CODES

The following cases represent some of the errors that can occur in a COBOL program and the interrupt or completion code associated with them. These errors do not necessarily cause an abnormal termination at the time they are recognized and do not always hold true. (See Appendix K for COBOL-initiated U-type completion codes.)

1. 013--Check register 2 of registers at the entry to ABEND. This address points to the DCB in conflict.

2. 043--Error occurred during the attempted opening of a TCAM application program data set, as described below.

   a. A value of 01 in register 0 indicates the attempted opening of a TCAM application program data set without an active message control program (MCP) in the system.

   b. A value of 02 indicates that the QNAME= parameter of a DD statement associated with an input or output DCB for a COBOL program is not the name of a process entry defined in the terminal table.

   c. A value of 03 indicates that the process entry named by the QNAME= parameter of a DD statement associated with a COBOL program is currently being used by another COBOL program.

   d. A value of 04 indicates that insufficient main storage was available in the MCP to build internal control blocks associated with the COBOL program interface. Specify a larger region or partition size in the JOB statement for the MCP.

   e. A value of 05 indicates that insufficient main storage was available in the COBOL work area to build internal control blocks. Specify a larger region or partition size in the JOB statement for the COBOL program.

3. 046--Error occurred during the termination of the TCAM MCP because the COBOL program data set was still open. Specify the STOP RUN statement when COBOL processing is complete. Ensure that all COBOL programs have terminated processing before deactivating the MCP.

4. 0C1--Operation Exception:

   a. When the interrupt is at 000048 or at 004800, look for a missing DD card or an unopened file.

   b. When the interrupt is at 000050, look at register 1 of the registers at entry to ABEND. Add hexadecimal 28 to the address found in register 1. This should point to the DD name of a missing DD statement.

c. When the interrupt is at 00004A, look for a missing card, i.e.,

```
//SYSOUT DD SYSOUT=A
```

any missing JCL card, or the wrong name of a JCL card. Add hexadecimal 28 to the address found in register 1 at entry to ABEND. This should point to the DD name of the DD statement in error.

d. When interrupt is at 00004F, look for inconsistent JCL or check the system-name in the COBOL program.

5. 0C4--Protection Exception:

a. Check for the block size and record size being equal for variable record input or output.

b. Check for missing SELECT statement.

c. If interrupt is at 004814, check for an attempt to READ an unopened input file or a missing DD card.

d. Check for an uninitialized index or subscript.

e. If a QSAM file with FILE STATUS opened OUTPUT, check for a missing DD card.

6. 0C5 and 0C6--Addressing and Specification Exception:

a. Subscript or index value may have exceeded maximum and instruction or table area was overlaid.

b. Check for an improper exit from a procedure being operated on by a PERFORM statement.

c. Check for duplicate close of an input or output file if DS formatting discontinued.

d. A sort is being attempted with an incorrect catalog procedure.

e. Attempting to reference an input/output area before a READ or OPEN statement, respectively.

f. Check for initialized subscript or index value.

7. 0C7--Data Exception:

a. Data field was not initialized.

b. Input record numeric field contains blanks.

c. Subscript or index value exceeded maximum and invalid data was referenced.

d. Data was moved from the DISPLAY field to the COMPUTATIONAL or COMPUTATIONAL-3 field at group level. Therefore, no conversion was provided.

e. The figurative constants ZERO or LOW-VALUE moved to a group level numeric field.

f. Omission of USAGE clause or erroneous USAGE clause.

g. Incorrect Linkage Section data definition, passing parameters in wrong order, omission or inclusion of a parameter, failure to carry over a USAGE clause when necessary, or defining the length of a parameter incorrectly.

8. 001--I/O Error:

a. Register 1 of the SVRB points to the DCB which caused the input/output problem. Look for input record and blocking errors. That is, the input does not agree with the record and blocking descriptions in the DCB, the COBOL file description, or the DD statement LRECL parameter.

b. Attempted to READ after EOF has been sensed.

c. Attempted to write to a QSAM file that has previously encountered end of file (taken a B37 exit), and set the file status to X'34' and/or entered the INVALID KEY routine.

9. 002--Register 2 of registers at the entry to ABEND contains the address of the DCB for the file causing the input/output problem. Check the DCB list for the specific file.

10. 013--Error during execution of an OPEN EXTEND statement. Ensure that the system OPEN EXTEND facility is available. OPEN EXTEND requires at least OS/VS1 Release 6, or OS/VS2 Release 7 with SU8.

11. 213--Error during execution of OPEN statement for data set on mass storage device, as follows:

    a. DISP parameter of DD statement specified OLD for output data set.

b. Input/output error cannot be corrected when reading or writing the DSCB. Recreate the data set or resubmit the job.

12. 214--Error during CLOSE for data set on tape; there is an input/output error that cannot be corrected either in tape positioning or volume disposition. Resubmit the job and inform the field engineer if error persists.

13. 237--Error at EOV:

    a. Incorrect volume serial number specified in SER subparameter of VOLUME parameter of DD statement.

    b. Incorrect volume mounted.

    c. Incorrect labels.

14. 400--If this completion code is generated during a compile step, the member to be compiled has not been extracted from the source library for compilation.

15. 413--Error during execution of an OPEN statement for a data set on tape:

    a. Volume serial number was not specified for input data set.

    b. Volume could not be mounted on the allocated device.

    c. There is an input/output error in reading the volume label that cannot be corrected.

16. 804--The error occurred during a GETMAIN. If this error occurs when a non-COBOL program (such as IMS or an installation-defined assembler program) links to a COBOL load module many times in a job step, the programmer should determine if the NOENDJOB option was used; if so, specifying the ENDJOB option may correct the problem.

17. 806--The error occurred during execution of a LINK, XCTL, ATTACH, or LOAD macro instruction. An error was detected by the control program routine for the BLDL macro instruction. The contents of register 15 indicate the nature of the error:

    04 The requested program was not found in the indicated private, job, or link library.

    08 An uncorrectable input/output error occurred when the control program attempted to search the directory of the library indicated as containing the requested program.

18. 80A--Insufficient contiguous main storage for linkage to some phase of the compiler. The programmer should see if secondary data-set allocation has caused an extra DEB to be built at lower main storage addresses within the region. If so, this problem can be corrected by assigning sufficient primary extents for the data set in question. See "Data Set Requirements" for further information. If this error occurs when a non-COBOL program (such as IMS or an installation-defined assembler program) links to a COBOL load module many times in a job step, the programmer should determine if the NOENDJOB option was used; if so, specifying the ENDJOB option may correct the problem.

19. 813--Error during execution of an OPEN statement in verification of labels:

    a. Volume serial number specified in VOLUME parameter of DD statement is incorrect.

    b. Data set name specified in DSNAME parameter is incorrect.

    c. Wrong volume is mounted.

20. 906--The system use count limit was exceeded during the execution of a LINK, XCTL, LOAD, or ATTACH macro. If this error occurs when a non-COBOL program (such as IMS or an installation-defined assembler program) links to a COBOL load module many times in a job step, the programmer should determine if the NOENDJOB option was used; if so, specifying the ENDJOB option may correct the problem.

Finding Location of Program Interruption in COBOL Source Program Using the Condensed Listing

To determine the location of the interruption, the programmer should proceed as follows:

1. From first page of dump:

   a. Get completion code and program interruption storage location.

   b. Determine the starting address of the program (PRB address+20).

2. From linkage editor listing:

   a. Determine storage address for each module. Add starting address of the program to origin of each module.

   b. Determine module in which interrupt storage location falls.

   c. Determine relative address. Subtract module storage address from interrupt location.

3. From Procedure Division map:

   a. Find the highest previous relative address in the condensed listing. That statement is in error.

   b. Get line number and verb of COBOL source statement.

4. From source listing find the line number and verb of source statement causing program interruption.

USING THE ABNORMAL TERMINATION DUMP

The programmer can also determine the cause of an abnormal termination with the following material:

1. The COBOL program object code listing.

2. A knowledge of the layout of the COBOL object module.

3. The full abnormal termination dump in conjunction with the linkage editor map or cross reference list.

A description of the linkage editor output and of the COBOL object code listing is found in "Output." Figure 91 shows the layout of the COBOL program object module.

Note: The information in this section about the use of the abnormal termination dump applies only when running under OS/VS1. For information about the abnormal termination dumps under OS/VS2, see the publication, OS/VS2 Debugging Guide.

The abnormal termination dump provides the address at which the load module has been loaded (load address) and the address of the instruction that caused the interrupt. The programmer computes the load module area by adding the load address to the load module length, as shown in the linkage editor output. It is now possible to determine whether the instruction falls within the load module. If it does not, the interrupt could have resulted from an improper branch to a point outside the load module or an error occurring in another part of the system.

If the instruction does fall within the load module, the programmer now determines in which part: the main program, a COBOL library subroutine, or a called program. The ranges of the various parts are determined by adding their relative origins, as shown in the linkage editor output, to the load address.

If the instruction occurred in an object module generated for a COBOL program, (i.e., the main program), the programmer can determine whether or not the instruction was one of the generated object code instructions. He can determine the address of the first instruction in the Procedure Division (as found in the object code listing) by adding its relative location to the location of the object module (load address plus relative origin). If it was one of the object code instructions, a similar technique can be used to locate the exact instruction. If it was not one of these instructions, the error has occurred in another part of the object module. Control possibly went there because of an improper branch.

If the instruction that initiated the dump occurred in a COBOL library subroutine, or if the original program called another program and the instruction occurred in the called program, the instruction can be located by a similar technique. The linkage editor cross reference list indicates the locations where the call to the program or subroutine in question was made.

The following general rules can be used to determine the cause of the dump and the error.

1. Determine the COBOL statement that generated the code leading to the program check.

   a. The top of the system dump will tell the address of the PC (Program Check) instruction and the type of PC. Locate the instruction in the core dump.

   b. Determine the relocation factor of the program from the linkage editor map. Subtract the relocation factor from the address of the invalid instruction.

   c. The address that results may be located in the procedure division map generated by the MAP option. (The coding shown at this location of the map should correspond to the instruction located in step one.)

   d. Preceding the address and code found in step three, find the sequence number of the corresponding COBOL statement in the listing and the number of the element in the sentence that generated the code.

2. Be sure the COBOL statement is coded properly.

3. If the statement is coded properly, go back to the main storage dump and determine the type of PC.

   a. If it is a data exception, the programmer will probably find that the instruction is a decimal instruction, and that one of the fields either will not have a valid sign or will contain digits other than 0 to 9. To determine this, it will be necessary to find the fields in main storage. Inspect bits 4 through 7 of the low-order byte for a valid sign (A through F). If one is not present, this is the cause of the PC.

      If one or both of the fields being operated on are defined as external decimal, the programmer will find one or more pack instructions immediately ahead of the PC instruction. From these determine the address of the external decimal field that generated the invalid sign. Several common causes of data exceptions are given in "Errors Caused by Invalid Data."

   b. If it is a protection exception, one possible cause is that a base register used in the instruction has not been initialized. Base registers in COBOL are initialized at different times. For QSAM and VSAM input files the register is initialized at OPEN; for other input files, the register is not initialized until the first successful read. For output files, the registers are initialized during the processing of the OPEN statement. When faced with a protection exception, the programmer should go to the COBOL source program to ascertain that no data has been moved prior to the time when base registers are initialized.

   c. If an addressing or specification exception occurs, the programmer may find upon inspection (but not always) that registers have been unexpectedly modified and the problem becomes one of finding out how. Two possible approaches are:

      (1) Check the addresses in registers 14 and 15 against the address of the PC instruction. If the address of the PC instruction is equal to or slightly larger than the address in register 15, the address probably is in a subroutine, and the address in register 14 should be the return address. A BAL or BALR instruction probably will precede the return address. The programmer should look for this particularly when the problem is not with a COBOL statement. If the PC instruction has an address equal to or a bit larger than the address in register 14, then the programmer probably has just returned from a subroutine, and register 15 should still be pointing to the entry address of the subroutine. The programmer should check the coding to see if this could reasonably be so, and check the entry points listed on the linkage editor map. If this approach bears further action, a listing of the subroutine would be needed

or the instructions from the dump must be interpreted.

(2) If the foregoing step does not locate the error, the programmer should check back through the dump to see what exists between the PC instruction and the last unconditional branch in order to determine the possible course of events.

The sample COBOL program ABEND and its output, shown in Figure 98, illustrates in detail the way in which an object code listing, a cross-reference table, and an abnormal termination dump can be used together to debug a program. The circled numerals in the figures are cited in the associated text. Note that all values are expressed in hexadecimal format unless otherwise indicated.

In that example, the completion code in the dump, ①, indicates the condition causing the abnormal termination. If the system part of the code is nonzero, the explanation can be found in the appropriate Debugging Guide. In the program ABEND, the completion code is 0C7; invalid data is the reason for termination.

Debugging the Program

Suggested below are general procedures for locating and correcting the source statement responsible for abnormal termination.

1. The INTERRUPT hhhhhh entry, ②, gives the hexadecimal address of the instruction following the instruction that initiated the interrupt and caused the dump. This address can be used to determine the relative location of the instruction in the load module (see item 4 below).

2. To determine the main storage area occupied by the load module, add the total length of the module, in hexadecimal format, to its load address. The load address can be obtained from the EPA entry, ③, of the CDE specification. The last six digits of this entry are the address of the entry point (INIT1) in the COBOL program.

   The total length of the load module is indicated in the TOTAL LENGTH entry, ④, in the linkage editor output. The highest location in the load module is:

   ③ + ④

   Thus, the range is from ③ to ③ + ④. Since the address ② falls within this range, the instruction initiating the dump must be within the load module.

3. To determine the relative location within the load module of the instruction indicated in the INTERRUPT entry, subtract the load address from the address of the instruction.

4. To determine whether or not the instruction occurred in the object module generated for the program, compare its relative location ② - ③ with the total length, ⑤, of the object module. If the relative location were greater than the size of the object module, then the error would not be part of this program. A relative location between the size of the object module, ⑤, and the total length ④ would indicate that the abnormal termination had occurred in one of the COBOL library subroutines. Such an error could be located by comparing the relative location with the relative origin of the subroutines. In this example, ② - ③ is less than the object module size ⑤, so the instruction occurred in the main program.

5. To determine whether or not the abnormal termination occurred in one of the object code instructions generated as a result of a statement in the Procedure Division of the source program, compare its relative location with the relative location of the first generated instruction in the Procedure Division, ⑥. In this example, the relative location of the instruction is greater than that of the first generated instruction and so it can be found by locating the corresponding relative location. The immediately preceding object code instruction then is the instruction that initiated the dump, ⑦. In this example, it is an instruction generated as a result of a COMPUTE statement. Checking back to the source program listing, the corresponding statement 18, ⑧, is located and 'B' is seen to be the data-name that caused the trouble. Data item B is defined in the Data Division, ⑨, as a COMPUTATIONAL-3 or internal decimal item, but the value at B is there as a result of a VALUE clause for A, the item that B redefines. This value is in external decimal format since there is no USAGE clause specified. The configuration of A is invalid for B and results in an interrupt.

Determining the Location of an ABEND When Running Dynamically: When running dynamically, the programmer should do the following to determine whether the abend occurred in the main program.

1. Figure 99 is a Load List of the same program shown in Figure 98, but compiled with the DYNAM option. The compiler produces a Load List that contains the COBOL subroutine library names and the addresses used in the program. These are anything beginning with ILBO. (A) The programmer is particularly interested in any ILBO subroutine that does not end in a zero, such as ILBORNT, ILBOREC, ILBODSP, etc.

2. In this case, the abend has occurred at (D). To determine whether this is within the main program, go to the Load List, and look for the subroutine with its address closest to that of the abend, which is at (B).

3. Look below to the second part of the Load List. This contains the length of the subroutines that begin at the address specified above. In this case at (B), under the LN column, the

```
00001   00001  IDENTIFICATION DIVISION.
00002   00002  PROGRAM-ID. ABEND.
00003   00003  REMARKS.
00004   00004          THIS IS A PROGRAM TO ILLUSTRATE THE ABNORMAL
00005   00005             TERMINATION OF A PROGRAM.
00006   00006  ENVIRONMENT DIVISION.
00007   00007  CONFIGURATION SECTION.
00008   00008  SOURCE-COMPUTER. IBM-370-168.
00009   00009  OBJECT-COMPUTER. IBM-370-168.
00010   00010
00011   00011  DATA DIVISION.
00012   00012
00013   00013  WORKING-STORAGE SECTION.
00014   00014  01  RECORDA.
00015   00015      02  A PICTURE S9(4) VALUE 1234.
00016   00016      02  B REDEFINES A PICTURE S9(7) COMPUTATIONAL-3. (9)
00017   00017  PROCEDURE DIVISION.
00018   00018      COMPUTE B = B + 1. (8)
00019   00019      STOP RUN.

      INTRNL NAME   LVL SOURCE NAME                BASE   DISPL   INTRNL NAME   DEFINITION   USAGE      R  O  Q  M
      DNM=1-032     01  RECORDA                    BL=1     000   DNM=1-032     DS 0CL4      GROUP
      DNM=1-052     02  A                          BL=1     000   DNM=1-052     DS 4C        DISP-NM
      DNM=1-063     02  B                          BL=1     000   DNM=1-063     DS 4P        COMP-3     R


                        MEMORY MAP

                     TGT                    000A8
                    .
                  SAVE AREA                 000A8
                  SWITCH                    000F0
                  TALLY                     000F4
                  SORT SAVE                 000F8
                  ENTRY-SAVE                000FC
                  SORT CORE SIZE            00100
                  RET CODE                  00104
                  SORT RET                  00106
                  WORKING CELLS             00108
                  SORT FILE SIZE            00238
                  SORT MODE SIZE            0023C
                  PGT-VN TBL                00240
                  TGT-VN TBL                00244
                  RESERVED                  00248
                    :
                    :
                  UNCTL CELLS               002C0
                  PFMCTL CELLS              002C0
                  PFMSAV CELLS              002C0
                  VN CELLS                  002C0
                  SAVE AREA =2              002C0
                  SAVE AREA =3              002C0
                  XSASW CELLS               002C0
                  XSA CELLS                 002C0
                  PARAM CELLS               002C0
                  RPTSAV AREA               002C0
                  CHECKPT CTR               002C0

LITERAL POOL (HEX)

002C0 (LIT+0)     1C


                     PGT                    002C0

                  OVERFLOW CELLS            002C0
                  VIRTUAL CELLS             002C0
                  PROCEDURE NAME CELLS      002CC
                  GENERATED NAME CELLS      002CC
                  DCB ADDRESS CELLS         002D0
                  VNI CELLS                 002D0
                  LITERALS                  002D0
                  DISPLAY LITERALS          002D1

  REGISTER ASSIGNMENT

   REG 6   BL =1

WORKING-STORAGE STARTS AT LOCATION 000A0 FOR A LENGTH OF 00008.
```

Figure 98. COBOL Program That Will Abnormally Terminate (Part 1 of 3)

```
18       COMPUTE    0002D2 (6)                    START    EQU   *
                    0002D2  F8 70 D 208 C 010              ZAP   208(8,13),010(1,12)   TS=01          LIT+0
                    0002D8  FA 43 D 208 6 000        (7)   AP    208(5,13),000(4,6)    TS=04          DNM=1-63
                    0002DE  F8 33 6 000 D 20C              ZAP   000(4,6),20C(4,13)    DNM=1-53       TS=04+1
19       STOP       0002E4                        GN=01    EQU   *
                    0002E4  58 F0 C 008                    L     15,008(0,12)          V(ILBOSRV1)
                    0002E8  07 FF                          BCR   15,15
                    0002EA  50 D0 5 008           INIT2    ST    13,008(0,5)
                            .
                            .
                            .
                    00000C  C1C2C5D5C4404040               DC    X'C1C2C5D5C4404040'
                    000014  E5E2D9F1                       DC    X'E5E2D9F1'
                    000018  07 00                          BCR   0,0
                    00001A  98 9F F 024                    LM    9,15,024(15)
                    00001E  07 FF                          BCR   15,15
                    000020  96 02 1 034                    OI    034(1),X'02'
                    000024  07 FE                          BCR   15,14
                    000026  41 F0 0 001                    LA    15,001(0,0)
                    00002A  07 FE                          BCR   15,14
                    00002C  00000032A                      ADCON L4(INIT3)
                    000030  00000000                       ADCON L4(INIT1)
                    000034  00000000                       ADCON L4(INIT1)
                    000038  000002C0                       ADCON L4(PGT)
                    00003C  000000A8                       ADCON L4(TGT)
                    000040  000002D2                       ADCON L4(START)
                    000044  000002EA                       ADCON L4(INIT2)
                    000048                                 DS    15F
                    000084  00000000                       DC    X'00000000'
                    000088  F1F24BF4F24BF5F2               DC    X'F1F24BF4F24BF5F2'
                    000090  C1E4C740F1F66B40               DC    X'C1E4C740F1F66B40'
                    000098  F1F9F7F6                       DC    X'F1F9F7F6'

*STATISTICS*     SOURCE RECORDS =    19     DATA DIVISION STATEMENTS =     3     PROCEDURE DIVISION STATEMENTS =     2
*OPTIONS IN EFFECT*     SIZE =  151072  BUF =   12288  LINECNT = 57  SPACE1, FLAGW,   SEQ,   SOURCE
*OPTIONS IN EFFECT*       DMAP,   PMAP, NOCLIST, NOSUPMAP, NOXREF,   SXREF,   LOAD, NODECK, APOST, NOTRUNC, NOFLOW
*OPTIONS IN EFFECT*     NOTERM, NONUM, NOBATCH, NONAME, COMPILE=01, NOSTATE, NORESIDENT, NODYNAM, NOLIB, NOSYNTAX
*OPTIONS IN EFFECT*     NOOPTIMIZE, NOSYMDMP, NOTEST,   VERB,   ZWB, SYST, NOENDJOB, NOLVL
*OPTIONS IN EFFECT*     NOLST , NOFDECK,NOCDECK, LCOL2,  L120,   DUMP , NOADV , NOPRINT,
*OPTIONS IN EFFECT*     NOCOUNT, NOVBSUM, NOVBREF, LANGLVL(2)


F64-LEVEL LINKAGE EDITOR OPTIONS SPECIFIED LIST,LET,XREF
          DEFAULT OPTION(S) USED -  SIZE=(196608,65536)


                                        CROSS REFERENCE TABLE

  CONTROL SECTION                                ENTRY

    NAME     ORIGIN   LENGTH (5)                  NAME    LOCATION     NAME    LOCATION     NAME    LOCATION     NAME    LOCATION
  ABEND          00      390
  ILBOCOM0*     390      169                      ILBOCOM      390
  ILBOSRV *     500      48E                      ILBOSRV0     50A    ILBOSR5      50A    ILBOSR3      50A    ILBOSR       50A
                                                  ILBOSRV1     50E    ILBOSTP1     50E    ILBOST       512    ILBOSTP0     512
  ILBOBEG *     990      128                      ILBOBEG0     992
  ILBOCMM *     AB8      38B                      ILBOCMM0     ABA    ILBOCMM1     ABE
  ILBOMSG *     E48       F2                      ILBOMSG0     E4A


  LOCATION  REFERS TO SYMBOL  IN CONTROL SECTION     LOCATION  REFERS TO SYMBOL  IN CONTROL SECTION
      2C0          ILBOSRV0        ILBOSRV              2C4          ILBOSR5         ILBOSRV
      2C8          ILBOSRV1        ILBOSRV              260          ILBOCOM0        ILBOCOM0
      850          ILBOCOM         ILBOCOM0             854          ILBOCMM0        ILBOCMM
      858          ILBOBEG0        ILBOBEG              85C          ILBOMSG0        ILBOMSG
      860          ILBOSND2       $UNRESOLVED(W)
 ENTRY ADDRESS       00

 TOTAL LENGTH      F40 (4)
****GO        DOES NOT EXIST BUT HAS BEEN ADDED TO DATA SET
AUTHORIZATION CODE IS        0.
```

Figure 98. COBOL Program That Will Abnormally Terminate (Part 2 of 3)

```
JOB ABEND            STEP GO              TIME 124329   DATE 76229    ID = 000                              PAGE 0001

COMPLETION CODE      SYSTEM = 0C7 (1)

PSW AT ENTRY TO ABEND  078D2000 0007539E        ILC 6   INTC 0007

ASCB 00FE8A28
             ASCB   C1E2C3C2   FWDP   00FE7E68   BWDP   00FE6B08   CMSF   00000000   SVRB   00000000   SYNC   00000000
             IOSP   00000000   SPL    00FE8AF8   CPUS   00000001   IDSQ   001E0004   IDDP   E2423078   STOR   0F18DC00
             LDA    007FF548   RSM    00FE88E8   CSCB   00FE8B10   TSB    00000000   EJST   00000000   XJST   1D737000
             CWST   89737A1F   XWST   02235000   JSTL   0000011E   ECB    807FD790   UBET   8973742F   QSCV   001FF630
             DUMP   007FF008   FW1    FFFF0000   TMCH   00000000   ASXB   007FF300   SWCT   02570000   SSRB   00000000
             VSC    00000039   TCBS   00000001   LOCK   00000000   LSQH   00000000   QECB   00000000   MECB   40000000
             DUCB   00FE89C8   DUXB   00FF2A48   FMCT   00240000   XMPQ   00000000   IQEA   00000000   RTMA   00000000
             MCC    00000000   JBNI   00FE8370   JBNS   00FE8B20   LGCB   00000000   VGTT   00000000   PCTT   00000000
             SMCT   00000000   SWTL   0000141D   SRBT   00000000   ATME   03336000

TCB  7D3128  KBP   007FD9D0   PIE    00000000   DEB   007D8798   TID 007C0020   CMP   9D0C7000   TRN   00000000
             MSS   007FC5A8   PK-FLG 80010000   FLG   0000FFFF   LLS 007D3840   JLB   007C8A50   JPQ   007D34E0
             PSA   01074FB0   TCB    00000000   TME   00000000   JST 007D3128   NTC   00000000   OTC   007FCA00
             LTC   00000000   IQE    00000000   ECB   007C8174   TSF 20000000   D-PQE 007FF548   AQE   007C8C18
             STAB  007FD04C   TCT    807FC960   USER  00000000   SDF 00000000   MDID  00000000   JSCB  007FBF84
             RESV  00000000   IOBRC  00000000   XCPD  00000000   EXT 00000000   BITS  00000000   DAR   00000000
             EXT2  007D3250   AECB   00000000   RESV  00000000   BAK 007FCA00   RTMWA 007DEC98   IOTM  00000000
             TMSAV 00000000   ABCR   00000000   RESV  00000000   FOE 00000000   SWA   007DCD50   RESV  00000000
             BID   E3C3C240   RTM1   00000000   ESTA  00000000   UKY 007C8600   CPVI  0040FFFF   BYTI  48040000
             RPT   00000000   OBTB   00000000   SWAS  007D9F30   SCB 00000000   GTF   00000000   SVAB  00000000
             EVENT 00000000   RESV   00000000   RESV  00000000   RES 00000000   RESV  00000000   RESV  00000000


ACTIVE RBS

PRB  7C9F98  RESV  00000000  RESV 00000000  RTPSW1 078D2000   0007539E  RTPSW2 00060007   0007C0C0
             FLG1  00000000  WC-L-IC 00060007
             RESV  00000000  APSW     00000000  SZ-STAB 00110082       FL-CDE 007D34E0  PSW 078D2000   0007539E
             W/TTR 00000000  WT-LNK  007D3128
             RG 0-7    007D3B50    00074FF8    00000040    007F26F4    007F26D0    007FCA00    007C0018    FD000000
             RG 8-15   007C8150    807FC960    00000000    007FCFE0    70DCA602    00074FB0    40DCABFC    007C8180

SVRB 7FD900  RESV  00000000  RESV 00000000  RTPSW1 00000000   00000000  RTPSW2 00000000   00000000
             FLG1  20000000  WC-L-IC 00020033
             RESV  00000000  APSW     00000000  SZ-STAB 0019D022       FL-CDE 00000000  PSW 070C1000   00048590
             W/TTR 00000000  WT-LNK  007C9F98
             RG 0-7    00000000    50075432    00000040    007F25F4    00075384    50075416    00075160    0007536F
             RG 8-15   00075370    000753EA    000750C0    000750C0    00075380    00075168    00075392    000753EC
             EXTSA     00000000    50075432    00018C30    007D3128    007FD900    00FE8A28    0003AA64    00000000
                       007DEC98    FF000368    40000101    900C7000
             SCB       00000000    00000000    00000000    00000000    00000000    RESV    00000000

LOAD LIST

        NE 00000000   RSP-CDE 00FE7660   CNT 00010001


CDE                                                                                          (3)
     7D34E0       NCDE 0000000D   RBP 007C9F98 NM GO         EPA 000750C0   XL/MJ 007C8000   USE 00010000   ATTR 0B20000
     FE7660       NCDE 00FEE47D   RBP 00000000 NM IGG0190K   EPA 00F2A000   XL/MJ 00FE7680   USE 00000000   ATTR 8000000

SAVE AREA TRACE

GO       WAS ENTERED VIA LINK

SA   074FB0  WD1 00000000   HSA 00000000   LSA 00075168   RET 00018C80   EPA 000750C0   R0  00703B50
             R1  00074FF8   R2  00000040   R3  007F26F4   R4  007F26D0   R5  007FCA00   R6  007C0018
             R7  FD000000   R8  007C8150   R9  807FC960   R10 00000000   R11 007FCFE0   R12 70DCA602

SA   075168  WD1 00000000   HSA 00074FB0   LSA 00000000   RET 00000000   EPA 00000000   R0  00000000
             R1  00000000   R2  00000000   R3  00000000   R4  00000000   R5  00000000   R6  00000000
             R7  00000000   R8  00000000   R9  00000000   R10 00000000   R11 00000000   R12 00000000

INTERRUPT AT 07539E (2)

PROCEEDING BACK VIA REG 13

SA   075168  WD1 00000000   HSA 00074FB0   LSA 00000000   RET 00000000   EPA 00000000   R0  00000000
             R1  00000000   R2  00000000   R3  00000000   R4  00000000   R5  00000000   R6  00000000
             R7  00000000   R8  00000000   R9  00000000   R10 00000000   R11 00000000   R12 00000000

GO       WAS ENTERED VIA LINK

REGS AT ENTRY TO ABEND

     FLTR 0-6      0000000000000000      0000000000000000      0000000000000000      0000000000000000

     REGS 0-7      00000000   50075432   00000040   007F26F4      00075384   50075416   00075160   0007536F
     REGS 8-15     00075370   000753EA   000750C0   000750C0      00075380   00075168   00075392   000753EC

LOAD MODULE    GO

075CC0   90ECD00C 185D05F0 4580F010 C1C2C5D5   C4404040 E5E2D9F1 0700989F F02407FF   *.......0..0.ABEND   VSR1....0...*
075CE0   96021034 07FE41F0 000107FE 000753EA   000750C0 000750C0 00075380 00075168   *........0.....................*
075100   00075392 000753AA 00000000 00000000   00000000 00000000 00000000 00000000   *................................*
075120   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000   *................................*
075140   00000000 00000000 F1F242F4 F248F5F2   C1E4C740 F1F66840 F1F9F7F6 00000000   *........12.42.52AUG 16, 1976....*
```

Figure 98. COBOL Program That Will Abnormally Terminate (Part 3 of 3)

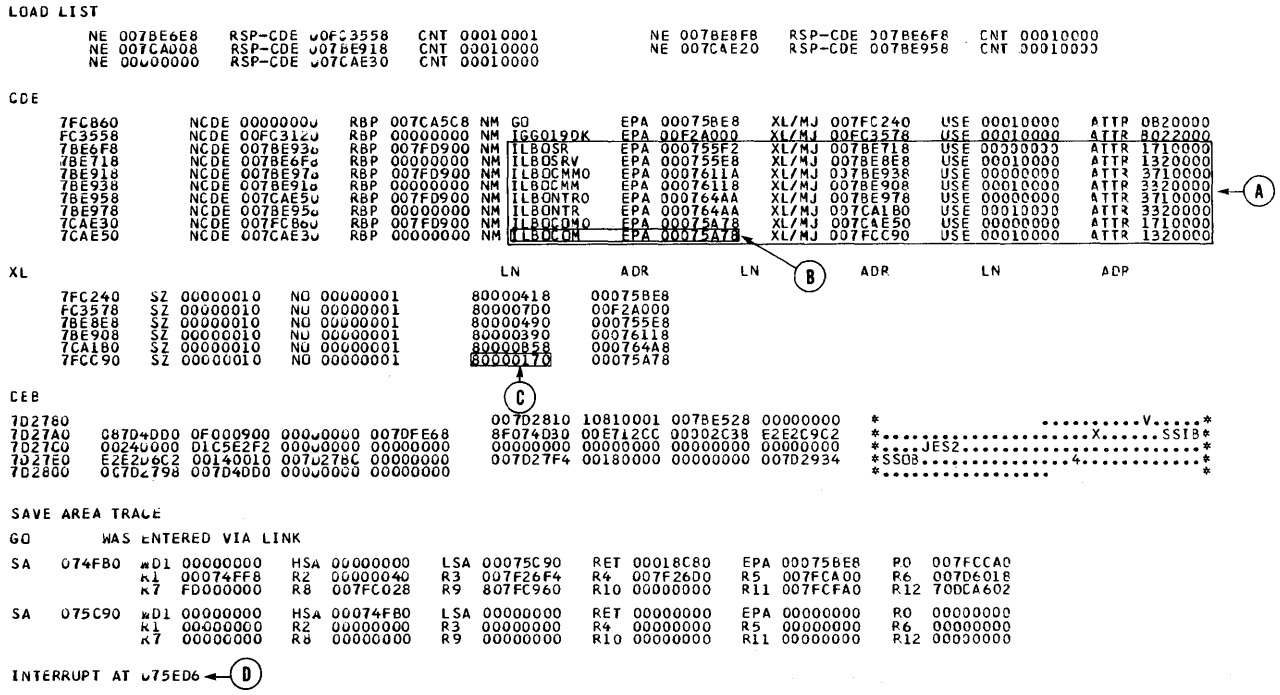


Figure 99. Load List of Program That Will Abnormally Terminate

length of the subroutine is at Ⓒ. Adding the length of the subroutine Ⓒ to the starting address Ⓑ, results in a number falling within the confines of the main program.

4. After this is determined, the programmer continues his debugging in the specified manner.

Finding Data Records in an Abnormal Termination Dump

The glossary, listed when the DMAP option is specified, contains information about all data-names described in the COBOL source program. The location assigned to a given data-name may be found by using the BL number and displacement specified for that entry in the glossary, and then locating the appropriate BL cell in the TGT. The hexadecimal sum of the glossary displacement and the contents of the cell should give the relative address of the area desired. This can be converted to an absolute address as described in the text associated with Figure 98.

Since the sample problem program shown in Figure 100 was interrupted because of a data exception, the programmer should locate the contents of field B at the time of the interrupt. The numerals encircled in the two techniques given below refer to information similarly labeled in the sample program.

Using the General Registers: The general registers usually contain information that can be helpful to the programmer who is trying to locate specific data.

1. Locate data-name B, ① , in the glossary. It appears under the column headed SOURCE-NAME. Source-name B has been assigned to base locator 3 (i.e., BL=3) with a displacement of 058. The sum of the value of base locator 3 and the hexadecimal displacement value 58 is the address of data-name B.

2. The Register Assignment table, ② , lists the registers assigned to each base locator. Register 6 has been assigned to BL=3.

3. The contents of the 16 general registers at the time of the interrupt are displayed at the beginning of the dump, ③

4. The location of data-name B, ④ , can now be determined by adding the contents of register 6 and the hexadecimal displacement value 58. The result is the address of the leftmost byte of the 4-byte field B. Field B contains F1F2F3C4. This is external decimal representation and does not correspond to the USAGE COMPUTATIONAL-3 defined in the source listing.

Using the TGT Memory Map: If the general registers appear not to contain meaningful information, it may be that errors in the problem program have destroyed their contents. In such a case, the alternate method of locating data-names given below should be helpful.

1. The location assigned to a given data-name may also be found by using the BL CELLS relocation value given in the TGT Memory Map, ⑤ . To find the location of the BL cells, add ⑤ (from the TGT table) to the entry point address of the object module, ⑥.

2. The first four bytes are the first BL cell, the second four bytes are the second BL cell, etc. Note that the third BL cell, ⑦ , contains the same value as that contained in register 6.

   Note: Use of the FLOW and STATE options eliminates the need for the calculations described above. All that is needed for program debugging is the output from FLOW and STATE printed at the end of the listing, ⑧ , and described below.

   A. Specification of either FLOW or STATE causes the PROGRAM-ID, the completion code, and the PSW for the last problem program executed before the abnormal termination to be printed out.

   B. If STATE is in effect, the printed output includes the compiler-generated card number for the last verb executed.

   C. If FLOW is in effect, the words FLOW TRACE are printed out, together with the PROGRAM-ID and the card numbers of the procedure-names executed for all COBOL programs with the FLOW option in effect.

   For further discussion of the FLOW and STATE compiler options, including their relationship to the NUM option and to the SYMDMP option, see the chapter entitled "Symbolic Debugging Features."

```
00001   10001U IDENTIFICATION DIVISION.
00002   10002U PROGRAM-ID.  TESTRUN.
00003   10003U   AUTHOR. PROGRAMMER NAME.
00004   10004U   INSTALLATION.  PALO ALTO DEVELOPMENT CENTER.
00005   10005U   DATE WRITTEN.  AUGUST 8, 1976.
00006   10006U DATE-COMPILED. AUG 23,1976.
00007   10007U   REMARKS. THIS PROGRAM HAS BEEN WRITTEN AS A SAMPLE PROGRAM FOR
00008   10008U     COBOL USERS. IT CREATES AN OUTPUT FILE AND READS IT BACK AS
00009   10009U     INPUT.
00010
00011   10010U ENVIRONMENT DIVISION.
00012   10011U CONFIGURATION SECTION.
00013   10012U   SOURCE-COMPUTER.  IBM-370-168.
00014   10013U   OBJECT-COMPUTER.  IBM-370-168.
00015   10014U INPUT-OUTPUT SECTION.
00016   10015U FILE-CONTROL.
00017   10016U     SELECT FILE-1 ASSIGN TO UT-2400-S-SAMPLE.
00018   10017U     SELECT FILE-2 ASSIGN TO UT-2400-S-SAMPLE.
00019
00020   10018U DATA DIVISION.
00021   10019U FILE SECTION.
00022   10020U FD  FILE-1
00023   10021U     LABEL RECORDS ARE OMITTED
00024   10022U     BLOCK CONTAINS 100 CHARACTERS
00025   10023U     RECORD CONTAINS 20 CHARACTERS
00026   10024U     RECORDING MODE IS F
00027   10025U     DATA RECORD IS RECORD-1.
00028   10026U   01 RECORD-1.
00029   10027U     02 FIELD-A PICTURE IS X(20).
00030   10028U FD  FILE-2
00031   10029U     LABEL RECORDS ARE OMITTED
00032   10030U     BLOCK CONTAINS 5 RECORDS
00033   10031U     RECORD CONTAINS 20 CHARACTERS
00034   10032U     RECORDING MODE IS F
00035   10033U     DATA RECORD IS RECORD-2.
00036   10034U   01 RECORD-2.
00037   10035U     02 FIELD-A PICTURE IS X(20).
00038
00039   10036U WORKING-STORAGE SECTION.
00040   10037U     77 KOUNT PICTURE S99 COMP SYNC.
00041   10038U     77 NOMBER  PICTURE S99 COMP SYNC.
00042   10039U   01 FILLER.
00043   10040U     02 ALPHABET PICTURE X(26) VALUE "ABCDEFGHIJKLMNOPQRSTUVWXYZ".
00044   10041U     02 ALPHA REDEFINES ALPHABET PICTURE X OCCURS 26 TIMES.
00045   10042U     02 DEPENDENTS PICTURE X(26)  VALUE "01234012340123401234012340
00046   10043U-    "0".
00047   10044U     02 DEPEND REDEFINES DEPENDENTS PICTURE X OCCURS 26 TIMES.
00048   10045U   01 WORK-RECORD.
00049   10046U     02 NAME-FIELD  PICTURE X.
00050   10047U     02 FILLER PICTURE X VALUE IS SPACE.
00051   10048U     02 RECORD-NO PICTURE 9999.
00052   10049U     02 FILLER PICTURE X VALUE IS SPACE.
00053   10050U     02 LOCATION PICTURE AAA VALUE IS "NYC".
00054   10051U     02 FILLER PICTURE X VALUE IS SPACE.
00055   10052U     02 NO-OF-DEPENDENTS PICTURE XX.
00056   10053U     02 FILLER PICTURE X(7) VALUE IS SPACES.
00057   10054U     01 RECORDA.
00058   10055U        02  A PICTURE S9(4) VALUE 1234.
00059   10056U        02  B REDEFINES A PICTURE S9(7) COMPUTATIONAL-3.
00060   10057U PROCEDURE DIVISION.
00061   10058U BEGIN.
00062   10059U     NOTE THAT THE FOLLOWING OPENS THE OUTPUT FILE TO BE CREATED
00063   10060U     AND INITIALIZES COUNTERS.
00064   10061U STEP-1. OPEN OUTPUT FILE-1. MOVE ZERO TO KOUNT NOMBER.
00065   10062U     NOTE THAT THE FOLLOWING CREATES INTERNALLY THE RECORDS TO BE
00066   10063U     CONTAINED IN THE FILE, WRITES THEM ON TAPE, AND DISPLAYS
00067   10064U     THEM ON THE CONSOLE.
00068   10065U STEP-2. ADD 1 TO KOUNT, ADD 1 TO NOMBER, MOVE ALPHA (KOUNT) TO
00069   10066U     NAME-FIELD.
00070   10067U     MOVE DEPEND (KOUNT) TO NO-OF-DEPENDENTS.
00071   10068U     MOVE NOMBER TO RECORD-NO.
00072   10069U STEP-3. DISPLAY WORK-RECORD UPON CONSOLE. WRITE RECORD-1 FROM
00073   10070U     WORK-RECORD.
00074   10071U STEP-4. PERFORM STEP-2 THRU STEP-3 UNTIL KOUNT IS EQUAL TO 26.
00075   10072U     NOTE THAT THE FOLLOWING CLOSES OUTPUT AND REOPENS IT AS
00076   10073U     INPUT.
00077   10074U STEP-5. CLOSE FILE-1. OPEN INPUT FILE-2.
00078   10075U     NOTE THAT THE FOLLOWING READS BACK THE FILE AND SINGLES OUT
00079   10076U     EMPLOYEES WITH NO DEPENDENTS.
00080   10077U STEP-6. READ FILE-2 RECORD INTO WORK-RECORD AT END GO TO STEP-8.
00081   10078U        COMPUTE B = B + 1.
00082   10079U STEP-7. IF NO-OF-DEPENDENTS IS EQUAL TO "0" MOVE "Z" TO
00083   10080U     NO-OF-DEPENDENTS. EXHIBIT NAMED WORK-RECORD. GO TO
00084   10081U     STEP-6.
00085   10082U STEP-8. CLOSE FILE-2.
00086   10083U     STOP RUN.
```

Figure 100. Program with Data Interrupt (Part 1 of 5)

INTRNL NAME	LVL	SOURCE NAME	BASE	DISPL	INTRNL NAME	DEFINITION	USAGE	R	O	Q	M
DNM=1-148	FD	FILE-1	DCB=01		DNM=1-148		QSAM				F
DNM=1-168	01	RECORD-1	BL=1	000	DNM=1-168	DS 0CL20	GROUP				
DNM=1-189	02	FIELD-A	BL=1	000	DNM=1-189	DS 20C	DISP				
DNM=1-206	FD	FILE-2	DCB=02		DNM=1-206		QSAM				F
DNM=1-226	01	RECORD-2	BL=2	000	DNM=1-226	DS 0CL20	GROUP				
DNM=1-247	02	FIELD-A	BL=2	000	DNM=1-247	DS 20C	DISP				
DNM=1-267	77	KOUNT	BL=3	000	DNM=1-267	DS 1H	COMP				
DNM=1-282	77	NOMBER	BL=3	002	DNM=1-282	DS 1H	COMP				
DNM=1-298	01	FILLER	BL=3	008	DNM=1-298	DS 0CL52	GROUP				
DNM=1-312	02	ALPHABET	BL=3	008	DNM=1-312	DS 26C	DISP				
DNM=1-330	02	ALPHA	BL=3	008	DNM=1-330	DS 1C	DISP	R	O		
DNM=1-348	02	DEPENDENTS	BL=3	022	DNM=1-348	DS 25C	DISP				
DNM=1-368	02	DEPEND	BL=3	022	DNM=1-368	DS 1C	DISP	R	O		
DNM=1-384	01	WORK-RECORD	BL=3	040	DNM=1-384	DS 0CL20	GROUP				
DNM=1-408	02	NAME-FIELD	BL=3	040	DNM=1-408	DS 1C	DISP				
DNM=1-428	02	FILLER	BL=3	041	DNM=1-428	DS 1C	DISP				
DNM=1-442	02	RECORD-NO	BL=3	042	DNM=1-442	DS 4C	DISP-NM				
DNM=1-461	02	FILLER	BL=3	046	DNM=1-461	DS 1C	DISP				
DNM=1-475	02	LOCATION	BL=3	047	DNM=1-475	DS 3C	DISP				
DNM=1-493	02	FILLER	BL=3	04A	DNM=1-493	DS 1C	DISP				
DNM=2-000	02	NO-OF-DEPENDENTS	BL=3	04B	DNM=2-000	DS 2C	DISP				
DNM=2-026	02	FILLER	BL=3	04D	DNM=2-026	DS 7C	DISP				
DNM=2-040	01	RECORD4	BL=3	058	DNM=2-040	DS 0CL4	GROUP				
DNM=2-060	02	A	BL=3	058	DNM=2-060	DS 4C	DISP-NM				
DNM=2-071	02	B ←(1)	BL=3	058	DNM=2-071	DS 4P	COMP-3	R			

MEMORY MAP

TGT	00318
SAVE AREA	00318
SWITCH	00360
TALLY	00364
SORT SAVE	00368
ENTRY-SAVE	0036C
SORT CORE SIZE	00370
RET CODE	00374
SORT RET	00376
WORKING CELLS	00378
SORT FILE SIZE	004A8
SORT MODE SIZE	004AC
PGT-VN TBL	004B0
TGT-VN TBL	004B4
RESERVED	004B8
LENGTH OF VN TBL	004BC
LABEL RET	004BE
RESERVED	004BF
DBG R14SAVE	004C0
COBOL INDICATOR	004C4
A(INITI)	004C8
DEBUG TABLE PTR	004CC
SUBCOM PTR	004D0
SORT-MESSAGE	004D4
SYSOUT DDNAME	004D8
RESERVED	004DD
COBOL ID	004DE
COMPILED POINTER	004E0
COUNT TABLE ADDRESS	004E4
RESERVED	004E8
DBG R11SAVE	004F0
COUNT CHAIN ADDRESS	004F4
PRBL1 CELL PTR	004F8
RESERVED	004FC
TA LENGTH	00501
RESERVED	00504
PCS LIT PTR	0050C
DEBUGGING	00510
CD FOR INITIAL INPUT	00514
OVERFLOW CELLS	00518
BL CELLS	00518 ←(5)
DECBADR CELLS	00524
FIB CELLS	00524
TEMP STORAGE	00528
TEMP STORAGE-2	00530
TEMP STORAGE-3	00530
TEMP STORAGE-4	00530
BLL CELLS	00530
VLC CELLS	00538
SBL CELLS	00538
INDEX CELLS	00538
SUBADR CELLS	00538
ONCTL CELLS	00540
PFMCTL CELLS	00540
PFMSAV CELLS	00540
VN CELLS	00544
SAVE AREA =2	0054C
SAVE AREA =3	0054C
XSASW CELLS	00554
XSA CELLS	00554
PARAM CELLS	00554
RPTSAV AREA	00558
CHECKPT CTR	00558
DEBUG TABLE	00558

Figure 100. Program with Data Interrupt (Part 2 of 5)

```
LITERAL POOL (HEX)

00580 (LIT+0)     00000001  00141C10  0000001C  00000008  00000000  48140000
005C8 (LIT+24)    00044000  00000000  C0000000

     DISPLAY LITERALS (BCD)

005D4 (LTL+36)   'WORK-RECORD'


                    PGT                         00560

               DEBUG LINKAGE AREA               00560
               OVERFLOW CELLS                   0056C
               VIRTUAL CELLS                    00570
               PROCEDURE NAME CELLS             0059C
               GENERATED NAME CELLS             0059C
               DCB ADDRESS CELLS                005A0
               VNI CELLS                        005A8
               LITERALS                         00580
               DISPLAY LITERALS                 005D4
               PROCEDURE BLOCK CELLS            005E0


 REGISTER ASSIGNMENT

  REG 6      BL =3 <-(2)
  REG 7      BL =1
  REG 8      BL =2

WORKING-STORAGE STARTS AT LOCATION 000A0 FOR A LENGTH OF 00060.

61     *BEGIN
                  0005E4                        PN=02    EQU   *
                  0005E4                        START    EQU   *
                  0005E4  58 B0 C 080                    L     11,080(0,12)          PBL=1
                  0005E8  58 F0 C 024                    L     15,024(0,12)          V(ILBOFLW1)
                  0005EC  05 1F                          BALR  1,15
                  0005EE  0000003D                       DC    X'0000003D'
64     *STEP-1
                  0005F2                        PN=03    EQU   *
                  0005F2  58 F0 C 024                    L     15,024(0,12)          V(ILBOFLW1)
                  0005F6  05 1F                          BALR  1,15
                  0005F8  00000040                       DC    X'00000040'
64       OPEN     0005FC  58 F0 C 028                    L     15,028(0,12)          V(ILBODBG4)
                  000600  05 EF                          BALR  14,15
                  000602  58 10 C 040                    L     1,040(0,12)           DCB=1
                  000606  58 40 1 024                    L     4,024(0,1)
                  00060A  D2 02 4 011 C 02D              MVC   011(3,4),02D(12)                      V(ILBOEXT0)
                  000610  50 10 D 234                    ST    1,234(0,13)           SAB=1
                  000614  92 0F D 234                    MVI   234(13),X'0F'         SAB=1
                  000618  96 80 D 234                    OI    234(13),X'80'         SAB=1
                  00061C  41 10 D 234                    LA    1,234(0,13)           SAB=1
                  000620  D2 03 D 060 C 057              MVC   060(4,13),057(12)     WC=01         LIT+7
                  000626  58 F0 C 030                    L     15,030(0,12)          V(ILBOQI00)
                  00062A  05 EF                          BALR  14,15
                  00062C  58 10 C 040                    L     1,040(0,12)           DCB=1
                  000630  D2 03 D 060 C 058              MVC   060(4,13),058(12)     WC=01         LIT+11
                  000636  58 F0 C 030                    L     15,030(0,12)          V(ILBOQI00)
                  00063A  05 EF                          BALR  14,15
                  00063C  58 70 D 200                    L     7,200(0,13)           BL =1
64       MOVE     000640  D2 01 6 000 C 050              MVC   000(2,6),050(12)      DNM=1-267     LIT+0
                  000646  D2 01 6 002 C 050              MVC   002(2,6),050(12)      DNM=1-282     LIT+0
66     *STEP-2
                  00064C                        PN=04    EQU   *
                  00064C  58 F0 C 024                    L     15,024(0,12)          V(ILBOFLW1)
                  000650  05 1F                          BALR  1,15
                  000652  00000044                       DC    X'00000044'
66       ADD      000656  48 30 C 052                    LH    3,052(0,12)           LIT+2
                  00065A  4A 30 6 000                    AH    3,000(0,6)            DNM=1-267
                  00065E  40 30 6 000                    STH   3,000(0,6)            DNM=1-267
68       ADD      000662  48 30 C 052                    LH    3,052(0,12)           LIT+2
                  000666  4A 30 6 002                    AH    3,002(0,6)            DNM=1-282
                  00066A  40 30 6 002                    STH   3,002(0,6)            DNM=1-282
68       MOVE     00066E  41 40 6 008                    LA    4,008(0,6)            DNM=1-330
                  000672  48 30 6 000                    LH    3,000(0,6)            DNM=1-257
                  000676  5C 20 C 050                    M     2,050(0,12)           LIT+0
                  00067A  1A 43                          AR    4,3
                  00067C  58 40 C 050                    S     4,050(0,12)           LIT+0
                  000680  50 40 D 220                    ST    4,220(0,13)           SBS=1
                  000684  58 E0 D 220                    L     14,220(0,13)          SBS=1
                  000688  D2 00 6 040 E 000              MVC   040(1,6),000(14)      DNM=1-408     DNM=1-330
70       MOVE     00068E  41 40 6 022                    LA    4,022(0,6)            DNM=1-368
                  000692  48 30 6 000                    LH    3,000(0,6)            DNM=1-267
                  000696  5C 20 C 050                    M     2,050(0,12)           LIT+0
                  00069A  1A 43                          AR    4,3
                  00069C  58 40 C 050                    S     4,050(0,12)           LIT+0
                  0006A0  50 40 D 224                    ST    4,224(0,13)           SBS=2
                  0006A4  58 F0 D 224                    L     15,224(0,13)          SBS=2
                    .
                    .
                    .
```

Figure 100. Program with Data Interrupt (Part 3 of 5)

```
JOB TESTRUN          STEP GO              TIME 202835   DATE 76236    ID = 000                              PAGE 0001

COMPLETION CODE      SYSTEM = 0C7

PSW AT ENTRY TO ABEND  078D2000 00075F46        ILC 6   INTC 0007

ASCB 00FCDCA0
             ASCB   C1E2C3C2   FWDP   00FE83F8   BWDP   00FBCA28   CMSF   00000000   SVRB   00000000   SYNC   00000000
             IOSP   00000000   SPL    00FCDD70   CPUS   00000001   IDSQ   00200006   IODP   114E0079   STOR   0F35EC00
             LDA    007FF548   RSM    00FCDB60   CSCB   00FCDD88   TSB    00000000   EJST   00000001   XJST   FF0F2000
             EWST   897CAF22   XWST   4016B000   JSTL   0000011E   ECB    807FD790   UBET   897C998A   QSCV   001FF630
             DUMP   007FF008   FW1    FFFF0000   TMCH   00000000   ASXB   007FF300   SWCT   7A0C0000   SSRB   00000000
             VSC    00000058   TCBS   00000001   LOCK   00000000   LSQH   00000000   QECB   00000000   MECB   40000000
             LUCB   00FCDC40   OUXB   00FF2CC8   FMCT   00290000   XMPQ   00000000   IQEA   00000000   RTWA   00000000
             MLC    00000000   JBNI   00FE8B98   JBNS   00FCDD98   LGCB   00000000   VGTT   00000000   PCTT   00000000
             SMCT   00000000   SWTL   00001410   SRBT   00000000   ATME   33E76000

TCB  7CDB60  KBP    007FD900   PIE    00000000   DEB  007D3B80   TIO 007D5020   CMP   900C7000   TRN  00000000
             MSS    007D1690   PK-FLG 80010000   FLG  0000FFFF   LLS 007DEAE0   JLB   007D1F00   JPQ  007AC090
             FSA    01074FB0   TCB    00000000   TME  00000000   JST 007CDB60   NTC   00000000   GTC  007FCA00
             LTC    00000000   LQE    00000000   ECB  007DEFB4   TSF 20000000   D-PQE 007FF548   AQE  007C0CF8
             STAB   007FD04C   TCT    807FC010   USER 00000000   SDF 00000000   MOID  00000000   JSCB 007D0F64
             KESV   00000000   LOBRC  00000000   XCPD 00000000   EXT 00000000   BITS  00000000   DAR  00000000
             EXT2   007CDC88   AECB   00000000   RESV 00000000   BAK 007FCA00   RTMWA 007CD700   IOTM 00000000
             TMSAV  00000000   ABCR   00000000   RESV 00000000   FOE 00000000   SWA   007D1820   RESV 00000000
             GID    E3C3C240   KTM1   00000000   ESTA 80000000   UKY 007D17A0   CPVI  0040FFFF   BYT1 48040000
             KPT    00000000   UBT8   00000000   SWAS 007D0F30   SCB 00000000   GTF   00000000   SVAB 00000000
             EVENT  00000000   KESV   00000000   RESV 00000000   RES 00000000   RESV  00000000   RESV 00000000


ACTIVE RBS

PRB  7CF040  KESV   00000000  RESV 00000000  RTPSW1 078D2000   00075F46  RTPSW2 00060007   000760FF
             FLG1   02000000  WL-L-IC 00060007
             KESV   00000000  APSW   00000000  SZ-STAB 00110082       FL-CDE 007D1ED8  PSW 078D2000   00075F46
             W/TTR  00000000  WT-LNK  007CDB60
             KG 0-7    007CF010    00074FF8    00000040    007F26F4    007F26D0    007FCA00    007D5018    FD000000
             KG 8-15   007DEF90    807FC010    00000000    007FCC08    70DC4602    00074FB0    4DDCA8FC    007DEFC0

SVRB 7FD9D0  KESV   00000000  KESV 00000000  RTPSW1 00000000   00000000  RTPSW2 00000000   00000000
             FLG1   20000000  WC-L-IC 00020033
             KESV   00000000  APSW   00000000  SZ-STAB 0019D022      FL-CDE 00000000  PSW 070C1000   00D48590
             W/TTR  00000000  WT-LNK  007CF040
             KG 0-7    40075F3C    0007AA70    40075F3C    0000001A    000758C8    00075688    00075758    0007AA1C
             KG 8-15   0007AA70    00076082    000756B8    00075C9C    00075C18    000759D0    50075F24    6001FC3C
             EXTSA     40075F3C    0007AA70    00018C30    007CDB60    007FD9D0    00FCDCA0    0003AA64    00000000
                       007CD7D0    FF000368    40000101    900C7000
             SCB       00000000    00000000    00000000    00000000    00000000   RESV    00000000

SVRB 7FD900  KESV   00000000  KESV 00000000  RTPSW1 00000000   00000000  RTPSW2 00000000   00000000
             FLG1   02000000  WC-L-IC 0002000C
             KESV   00000000  APSW   00000000  SZ-STAB 0019D022      FL-CDE 00000000  PSW 07DC2000   00D0B20E
             W/TTR  00000000  WT-LNK  007FD9D0
             KG 0-7    000000E0    007CDBD0    00FE4040    0080E600    007CDB60    007CD8D0    50D4801C    007CD7D0
             KG 8-15   007DDE68    00000000    007DDF50    007DDEC0    007CD7F0    007CD944    50D4828C    00000000
             EXTSA     00000000    007CD418    00000384    007FD964    20FD0000    00000000    00000000    00000000
                       00000000    00000000    00000000    00000000
             SCB       00000000    00000000    00000000    00000000    00000000   RESV    00000000


LOAD LIST

        NE 007DE0B0   RSP-CDE 00FE8DD0   CNT 00010001          NE 007AC0B0   RSP-CDE 00FE8EB0   CNT 00010001
        NE 007AC050   RSP-CDE 00FECFD0   CNT 00010001          NE 007DE000   RSP-CDE 007AC070   CNT 00010000
        NE 007DE0E0   RSP-CDE 00FBD2A0   CNT 00010001          NE 007DE120   RSP-CDE 00FCD700   CNT 00010001
        NE 007DEB48   RSP-CDE 007DE140   CNT 00010000          NE 007D1918   RSP-CDE 007DEB58   CNT 00010000
        NE 00000000   RSP-CDE 007D1928   CNT 00010000

                                                    (6)
CDE                                                  |
                                                     v
     7D1ED8       NCDE 00000000  RBP 007CF040 NM GD          EPA 00075688   XL/MJ 007CF000   USE 00010000   ATTR 8020000
     FE8DD0       NCDE 00FE9F38  RBP 00000000 NM IGG019DK    EPA 00F2A000   XL/MJ 00FE8DF0   USE 00010000   ATTR 8022000
     FE8EB0       NCDE 00FED820  RBP 00000000 NM IGG019AA    EPA 00F1EF60   XL/MJ 00FE8ED0   USE 00020000   ATTR 8022000
     FECFD0       NCDE 00FEE780  RBP 00000000 NM IGG019AQ    EPA 00BE1018   XL/MJ 00FECFF0   USE 00030000   ATTR 8922000
     7AC070       NCDE 007DE820  RBP 007FD9D0 NM ILB0D220    EPA 0007C6FA   XL/MJ 007AC090   USE 00000000   ATTR 3710000
     7AC090       NCDE 007AC070  RBP 00000000 NM ILB0D22     EPA 0007C6F8   XL/MJ 007AC060   USE 00010000   ATTR 3320000
     FBD2A0       NCDE 00FCD700  RBP 00000000 NM IGG019CW    EPA 00BD6A40   XL/MJ 00FBD2C0   USE 00020000   ATTR 8023000
     FCD700       NCDE 00FCE180  RBP 00000000 NM IGG019CU    EPA 00D61898   XL/MJ 00FCD720   USE 00020000   ATTR 8122000
     7DE140       NCDE 007DEAC0  RBP 007FD9D0 NM ILB0D210    EPA 0007967A   XL/MJ 007DE828   USE 00000000   ATTR 3710000
     7DE828       NCDE 007DE140  RBP 00000000 NM ILB0D21     EPA 00079678   XL/MJ 007DE130   USE 00010000   ATTR 3320000
     7DEB58       NCDE 007CB010  RBP 007FD9D0 NM ILB0D200    EPA 00079C42   XL/MJ 007DEAC0   USE 00000000   ATTR 3710000
     7DEAC0       NCDE 007DEB58  RBP 00000000 NM ILB0D20     EPA 00079C40   XL/MJ 007CB678   USE 00010000   ATTR 3320000
     7D1928       NCDE 007D1ED8  RBP 007FD9D0 NM ILB0D010    EPA 000753B4   XL/MJ 007CB010   USE 00000000   ATTR 3710000
     7CB010       NCDE 007D1928  RBP 00000000 NM ILB0D01     EPA 000753A8   XL/MJ 007CB000   USE 00010000   ATTR 3320000
```

Figure 100. Program with Data Interrupt (Part 4 of 5)

```
INTERRUPT AT 075F46

PROCEEDING BACK VIA REG 13

SA   0759D0  WD1 0030C4C2   HSA 00074FB0   LSA 00074B90   RET 50075F24   EPA 6001FC3C   R0  40075F3C
             R1  0007AA70   R2  00075900   R3  0000001A   R4  000758C8   R5  00075688   R6  00075758
             R7  0007AA1C   R3  0007AA70   R9  00076082   R10 00756B8   R11 00076176   R12 00075C18

GO       WAS ENTERED VIA LINK

SA   074FB0  WD1 00000000   HSA 00000000   LSA 000759D0   RET 00018C80   EPA 03075688   R0  007CF010
             R1  00074FF8   R2  00000040   R3  007F26F4   R4  007F26D0   R5  007FCA00   R6  007D5018
             R7  FD000000   R8  007DEF90   R9  807FC010   R10 00000000   R11 007FCC08   R12 70DCA602
VTAM NOT ACTIVE FOR THIS CALL

REGS AT ENTRY TO ABEND                                                                          (3)

     FLTR 0-6      0000000000000000      0000000000000000              0000000000003000      0000000000000000

     REGS 0-7      40075F3C   0007AA70   40075F3C   0000001A           000758C8   00756B8   00075758   0007AA1C
     REGS 8-15     0007AA70   00076082   00756B8    00075C9C           00075C18   003759D0  50075F24   6001FC3C

LOAD MODULE    GO

0756A0                                                                 90ECD00C 185D05F0   *                    .......0*
0756C0   4580F010 E3C5E2E3 D9E4D540 E5E2D9F1    0700989F F02407FF 96021034 07FE41F0   *..0.TESTRUN VSP1....0.........0*
0756E0   0001U7FE 00076082 000756B8 00075688    00075C18 000759D0 00075C9C 00076042   *..............................*
0757CC   CC00U000 00000000 000U0000 00000000    00000000 00000000 00000000 00000000   *..............................*
      LINE 075720 SAME AS ABOVE
075740   F2F04BF1 F848F5F4 C1E4C740 F2F36B40    F1F9F7F6 00000000 001A001A 00000000   *20.18.54AUG 23. 1976..........*
075760   C1C2C3C4 C5C6C7C8 C9D1D2D3 D4D5D6D7    D8D9E2E3 E4E5E6E7 E8E9F0F1 F2F3F4F0   *ABCDEFGHIJKLMNOPQRSTUVWXYZ012340*
075780   F1F2F3F4 F0F1F2F3 F4F0F1F2 F3F4F0F1    F2F3F4F0 00000000 C140F0F0 F0F140D5   *1234012340123401234....A 0001 N*
0757A0   E8C340F0 40404040 4040U4040 00000000 (4) F1F2F3C4 00000000 00000000 00000000   *YC 0         ....123D........*
0757C0   00000000 00000000 000U0000 0507795A    000756DE 00000001 00000001 00000000   *..............................*
0757E0   0000U000 00000000 000U0000 00000000    80000000 00000000 00000000 00000000   *..............................*
075800   00000006 00830000 0007AA01 00004000    00000001 46000001 000757BC E2C1D4D7   *.............. .............SAMP*
075820   D3C54040 02000048 000U0001 08000001    00000064 00000000 00000001 00000001   *LE  ..........................*
075840   0000U001 00000014 000U0001 00000000    00000001 05EF3703 00000000 00000000   *..............................*
075860   0000U000 00000000 000U0000 00900000    00000000 02000001 00140014 00000000   *..............................*


075880   0000U000 00000000 000U0000 00000000    00000000 00000000 00000000 00000000   *..............................*
      LINE 0758A0 SAME AS ABOVE
0758C0   0000U000 00000000 000U0000 00000000    00000000 00000000 0507795A 000756DE   *..............................*
0758E0   0000U001 00000001 000U0000 00000000    00000000 00000000 00000000 80000000   *..............................*
075900   C000U000 00000000 000U0000 00000005    0083C300 0507AA00 00004000 0007A8B8   *.....................C..... .....*
075920   46077F8C 900758C8 00CL4800 00703414    12F1EF60 00BE10L8 07000001 000D0064   *.......H.........l.............*
075940   2020L020 0007A920 0007AAD4 0007AA70    00000014 00000001 00000000 00BD6A40   *............M.................. *
075960   C5EF0700 00000000 000U0000 00000000    00000000 00000000 00900000 00000000   *..............................*
075980   0204U001 00140804 00U00000 00000000    00078F40 00000000 00000000 00000000   *.....................  ..........*
0759A0   0000U000 00000000 000U0000 00000000    00000000 00000000 00000000 00000000   *..............................*
0759C0   0000U000 00000000 000U0000 00000000    0030C4C2 00074FB0 00074B90 50075F24   *..................DB..........*
0759E0   6001FC3C 40075F3C 0007AA70 00075900    0000001A 000758C8 00075688 00075758   *.... .....................H.......*
075A00   0007AA1C 0007AA70 00076082 000756B8    00076176 00075C18 3D02804B 00000000   *..............................*
075A20   0000U000 00075C9C 000U0000 00000000    04400000 50075F24 12F1EF60 0007AC10   *.................. .......l......*
075A40   0007U900 00075900 000U001A 000758C8    000756B8 00075758 0007AA1C 0007BFE8   *.............H.............Y*
075A60   00076082 000756B8 00075C9C 00075C18    50077EF0 46075F36 0007AC10 80075900   *..............................*
075A80   0007U7F4 0000001A 08FUF000 00000000    00000000 58B0D1D8 F870D210 C0560700   *...4.....00...........JQ8.K.....*
075AA0   5820U1A8 07F20000 000U09C8 00074ED0    20000000 00000000 00000000 00000000   *.J..2.....H...................*
075AC0   0000U000 00000C00 000U0000 00075D9E    00000000 01000000 00075D8C 00076F6A   *..............................*
075AE0   60075D48 00075DA0 000U0000 0000001A    00075793 500760E6 00018C80 0007795A   *.....................M.........*
075B00   00079960 80C75900 00075900 0000001A    000758C8 00075688 00075758 0007AA1C   *.....................H.........*
075B20   0000U000 00076082 000756B8 00075C9C    00075C18 00000000 00000000 00000000   *..............................*
075B40   0000U000 00000000 000U0000 00000000    00000000 00000000 00000000 00000000   *..............................*
075B60   0000U000 00000000 000U0000 00000000    00000000 00000000 40075F3C 6007839C   *.....................  .......*
075B80   00079688 00000558 00078D98 E2E8E2D6    E4E34040 E380000C 00075740 00000000   *.............SYSOUT  T.......*
075BA0   0000U000 00000000 00075C9C 00000000 (7) 00075C98 00000000 00000000 00000000   *..............................*
075BC0   0000U000 00000000 000U0000 00000000    00075688 0007AA70 00075758 00000000   *..............................*
075BE0   0000U000 0000001C 000U0000 00000000    00075779 00075793 00075DF6 00075DF6   *.....................6...6*
075C00   0000U000 80075900 000U0000 00000000    0A00098A 16340000 50B0D1D8 58B0C01C   *.....................JQ....*
075C20   07FBU000 00076C18 0007845A 00076162    000779AA 00076176 0007845A 000779AE   *..............................*
075C40   00076172 0007795A 00077EE2 00076F6A    0007845E 00075F36 000757F4 00075900   *.............S.............4....*
075C60   00075DF6 00000000 000U0001 001A1C10    0000001C 00000008 00000000 48140000   *...6..........................*
075C80   00044000 00000000 C00U0000 E6D6D9D2    6D09C5C3 D6D9C400 00075C9C 58B0C080   *.. .........WORK.RECORD.......*

(A) PROGRAM        TESTRUN                    COBOL ABEND DIAGNOSTIC AIDS

(A) LAST PSW BEFORE ABEND = FF8500U7E0075F46     SYSTEM COMPLETION CODE = 0C7                (8)

(B) LAST CARD NUMBER/VERB NUMBER EXECUTED -- CARD NUMBER 000081/VERB NUMBER 01.

                                                 FLOW TRACE
(C) TESTRUN  000068 000072 000068 000072 000068 000072 000068 000072 000077 000080
```

Figure 100. Program with Data Interrupt (Part 5 of 5)

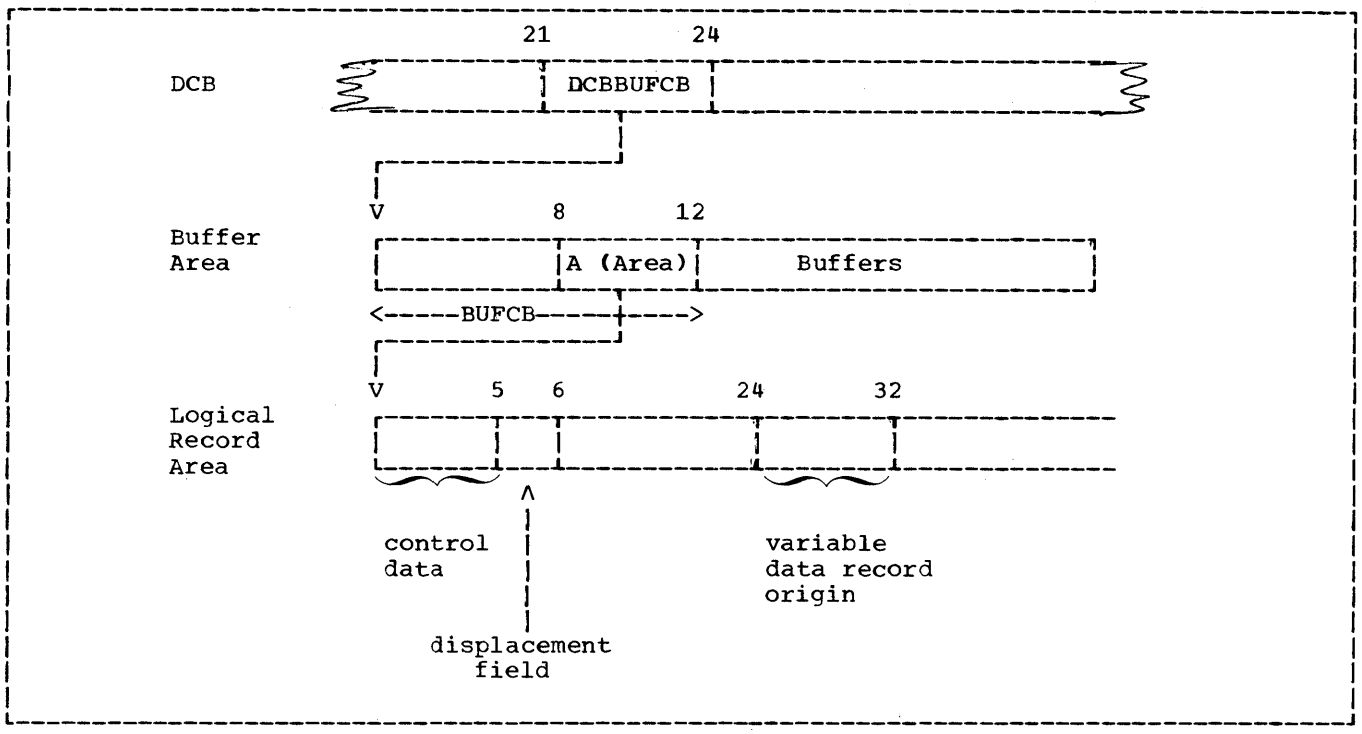


Figure 101. Locating the QSAM Logical Record Area

Locating Data Areas for Spanned Records

QSAM: QSAM (sequential) spanned records allocate a Logical Record Area in which complete logical records may be assembled (see "Record Formats"). Figure 101 illustrates the relationship between the DCB, the Buffer Areas, and the Logical Record Area.

1. The DCB contains the DCBBUFCB field at a displacement of 21 bytes from the origin of the DCB. The contents of DCBBUFCB points to the origin of the Buffer Control Block (BUFCB) in the Buffer Area.

2. The BUFCB field contains an Area-Address (A(Area)) at a displacement of 8 bytes from the origin of the Buffer Area. The Area-Address points to the origin of the Logical Record Area.

3. The Logical Record Area contains a displacement field at a displacement of 5 bytes from its origin. This field contains a value from 0 to 8 indicating the number of bytes the record has been displaced. The contents of this 1-7 byte field must be added to the value 24 (the first byte in the variable data record origin area) in order to locate the beginning of the logical data record within the Logical Record Area. Note that the first 4 bytes of the Logical Record Area are control data indicating the length of the Logical Record Area (including the 4 bytes of control data).

Note: The Logical Record Area is not allocated for QSAM records formatted in V, U, or F mode.

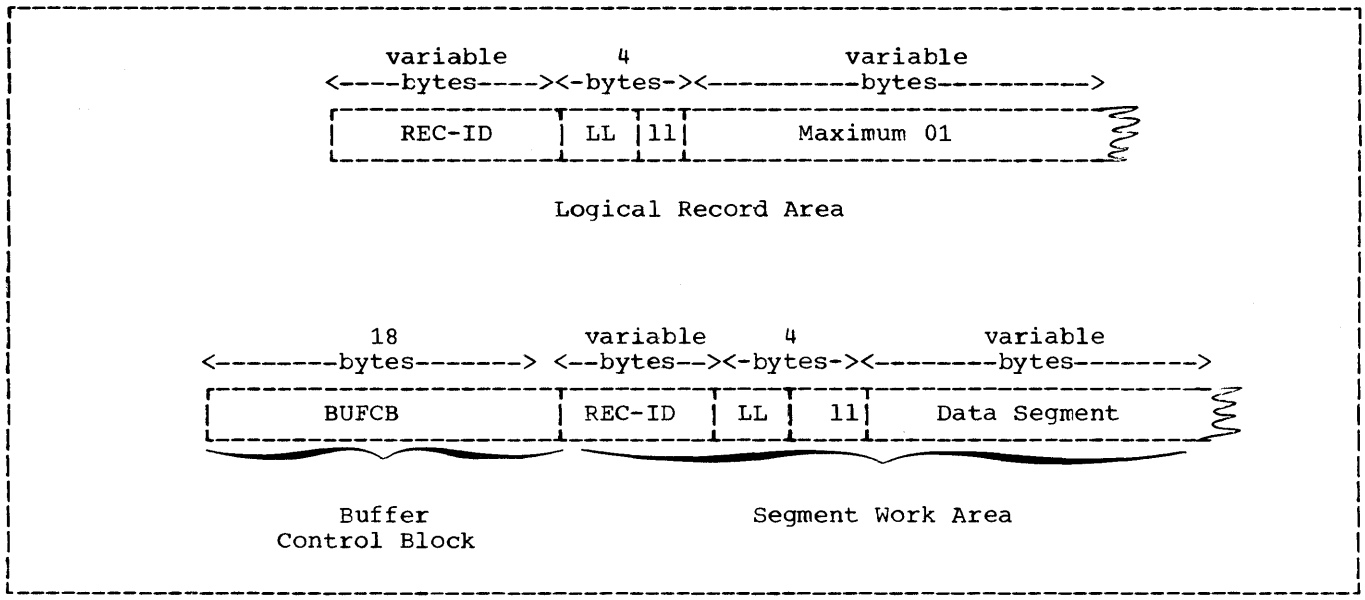


Figure 102. Logical Record Area and Segment Work Area for BDAM and BSAM Spanned Records

BSAM and BDAM: BSAM and BDAM (direct) spanned records allocate a Segment Work Area. This work area is used for temporary storage of record segments before a complete logical record is assembled in the Logical Record Area. Figure 102 illustrates the Logical Record Area and the Segment Work Area.

Note: The segment work area is not allocated for BSAM and BDAM records formatted in V, U, or F mode.

The following discussion illustrates the relationship between the DCB, the Logical Record Area, and the Segment Work Area as shown in Figure 102.

BDAM

1. The DCB address plus 100 bytes points to the beginning of the BUFCB (Buffer Control Block).
2. The contents of the BL assigned to the level-01 entry in an FD points to the Logical Record Area labeled "Maximum 01" in Figure 102 (see Figure 98 for an example of the BL pointer.)

BSAM output

1. The DCB address plus 76 bytes points to the beginning of the BUFCB (Buffer Control Block).
2. The DECB address plus 12 bytes points to the beginning of the Logical Record Area.

BSAM input

1. The DECB address plus 12 bytes points to the beginning of the Segment Work Area.
2. The DCB address plus 100 bytes points to the beginning of the Logical Record Area.

Locating TCAM Data Areas

In a teleprocessing application, control blocks, called queue blocks, are created for a given partition/region. For input operations, the number of queue blocks created agrees with the number of queues accessed. For output operations, however, only one queue block is created for each partition/region. The numerals within the boxes in Figures 103 and 104 refer to the numbered paragraphs below.

1. The TGT address plus 440 bytes points to the SUBCOM field (see Figure 184 in Appendix J: "Fields of the Global Table"). The fullword at X'50' bytes into SUBCOM points to the first RECEIVE queue block. The fullword at

X'54' off SUBCOM points to the SEND queue block. In both cases, the first field contains the data control block (DCB).

2. At X'58' bytes into either a RECEIVE or a SEND queue block, the first byte of the 4-byte field indicates whether the address that follows represents a TCAM buffer or a BSAM buffer. If the two high-order bits are on, the address contained in the next three bytes is for a TCAM buffer.

   Note: For TCAM there is only one buffer; for BSAM there is one buffer for each queue.

Relative Location	Field
0	RQBDCB 1
58	RQBUFAD 2
5C	RQBUFSIZ 3
5E	RQBSUMGV 4
60	RQBDECB 5
74	RQBNEXT 6
78	RQBAMTL 7
7A	RQBDTS 9
90	RQBDDNAM 11
98	RQBRECDL 12
99	RQBHELDC 13
9A	RQBISITP 14
9B	Reserved
9C	RQBNBL 15
A0	RQBQNAME 16

Figure 103. Fields of the RECEIVE Queue Block

Relative Location	Field
0	IHADCB 1
58	SQBUFAD 2
5C	SQBUFSIZ 3
5E	SQMORER 8
60	SQBENDIC 10
61	SQBRECDL 12
62	Reserved
64	SQBDNLH 17
68	SQBMPLFH 18
6C	SQBPBFH 19
70	SQBFSTOA 20
74	SQBFSTOL 21

Figure 104. Fields of the SEND Queue Block

3. In either a RECEIVE or a SEND queue block, this field specifies the size of the buffer, whose format is pictured in Figure 105. (For a list of codes used in the TCAM control byte, see Figure 106.)

4. The RQBSUMGV field of the RECEIVE queue block indicates the number of bytes of data given to the user for this request.

5. The RQBDECB field of the RECEIVE queue block contains the data event control block (DECB).

6. In the RECEIVE queue block, the RQBNEXT field provides the address of the next queue block. If this field is zero, there are no additional queue blocks.

7. The RQBAMTL field of the RECEIVE queue block indicates the amount of data being held from the last request.

8. For BSAM only, the SQMORER field of the SEND queue block indicates the number of unused bytes left in the buffer.

9. The 22-byte RQBDTS field of the RECEIVE queue block contains the date and time of the last message received from this queue, as well as the source of the message.

10. The SQBENDIC field of the SEND queue block contains the end indicator (in

zoned decimal) specified in the COBOL source statement.

11. The RQBDDNAM field contains the ddname for the queue block specified in the COBOL teleprocessing program.

12. The RQBRECDL field in the RECEIVE queue block and the SQBRECDL field in the SEND queue block contain the record delimiter specified in the MCP.

13. The RQBHELDC field in the RECEIVE queue contains a character that, in some instances, is the next data character.

14. The RQBISITP field in the RECEIVE queue block is a switch byte.

15. The RQBNBL field contains the enable/disable link.

16. The RQBQNAME field contains the TCAM queue name.

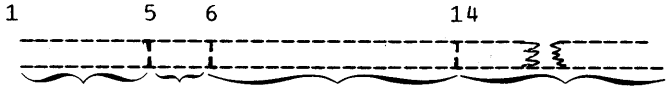
17. The SQBDNLH field contains the head of the DNL (destination name list).

18. The SQBMPLFH field contains the free list head of the MPL (message phrase list).

19. The SQBPBFH field contains the free list head of the PB (phase buffer).

20. The SQBFSTOA field contains the free storage pointer.

21. The SQBFSTOL field contains the free storage length.



Note: The prefix, the TCAM control byte, and the source ID must be user specified for a SAM file. However, if the user invokes the SEND statement to create a SAM file for subsequent input, then the COBOL compiler adds bytes 1 through 13 (see Figure 148 in the chapter entitled "Using the Teleprocessing Feature").

Figure 105. Structure of a TCAM Record

INCOMPLETE ABNORMAL TERMINATION

If a job is abnormally terminated and the abnormal termination process goes to completion, the following procedures are carried out:

- A dump (ABDUMP) is produced by the system.
- The data sets in the job steps are disposed of as specified in the DISP parameter (i.e., kept, deleted, etc.). This is indicated in the job scheduler disposition messages produced for the job step.
- Temporary data sets, including those passed from previous job steps, are deleted.

When the abnormal termination process does not go to completion (i.e., no end of dump message is present), none of these procedures will be carried out. Those data sets in the job step that were in existence previous to the point at which the error condition occurred will remain in effect. For data sets on direct access volumes, the names will remain tabulated in the Volume Table of Contents (VTOC) of the volume (see "Additional File Processing Information" for details on the VTOC). The result of an incomplete abnormal termination is that space needed by a subsequent job will be unavailable, or, if the same job is then rerun, duplicate name definition will result for those data sets that are newly created in the job step. This is true for temporary data sets for which the system has assigned the name, as well as data sets for which the programmer has assigned the name.

Code	Meaning
X'F1'	The first block of a multiblock message
X'F5'	The first block of a multiblock message, with end of segment indicated
X'40'	An intermediate data block
X'F4'	An intermediate data block, with end of segment indicated
X'F2'	The last block of a multiblock message
X'F6'	The last block of a multiblock message, with end of segment indicated
X'F3'	A single block message
X'F7'	A single block message, with end of segment indicated

Figure 106. Codes Used in the TCAM Control Byte

SCRATCHING NON-VSAM DATA SETS

To avoid duplicate name definition and to ensure that space will be available for newly created data sets, the programmer can scratch his direct-access volume data sets by using the utility program IEHPROGM. To scratch such a data set means to remove its data set label (which includes its name) from the VTOC and to make the space assigned to it available for reallocation. Scratching does not uncatalog any cataloged data sets. This is done by the UNCATLG option of the IEHPROGM.

If a DSNAME parameter has been specified in the DD statement for the data set, the IEHPROGM utility program requires the name of the data set. For data sets named by the programmer, the specified name is the dsname. For data sets for which the DSNAME=&&name convention has been used, an internal name

SYSyyddd.Ttttttt.axnnn.jobname.name

is assigned by the system, where jobname is the name of the job and name is from the &&name. If no DSNAME parameter is specified, an internal name is assigned by the system. For data sets with no DSNAME parameter there exists an option by which the programmer can specify that all such data sets on the volume be scratched, without having to specify their names.

If the programmer wishes to obtain a listing of the names of all the data sets on a volume, including system-assigned internal names, he can use the utility program IEHLIST. This program provides a listing of the VTOC of the volume.

Information on how to use these utility programs is contained in the publication OS/VS Utilities.

OBTAINING EXECUTION STATISTICS

Execution statistics are invoked via the COUNT option at compile time and the presence of SYSDBOUT and SYSCOUNT DD statements at execution time. No source language coding changes are required. COUNT facilitates testing, debugging, and optimizing by providing the programmer with verb counts at the following times:

- STOP RUN
- GOBACK in the main program
- Abnormal termination of a job.
- At CANCEL time, verb counts for the canceled program and any of its subprograms.

When COUNT is specified, the following items should be taken into account:

1. When COUNT is specified, the compiler divides the program into blocks of verbs. When the statistics are printed, the last block of verbs executed in each program unit is indicated. If the program abnormally terminates, the statement causing the abnormal termination can be determined (by using the symbolic debugging features, for example). The programmer should then subtract one from the verb count for each verb flagged which follows the abending verb.

2. If COUNT is requested, the user may need to increase the REGION parameter on his load module EXEC card. The dynamic space required for COUNT is approximately 512 bytes plus 80 bytes per program unit being monitored, and four bytes per count block (see the compiler output statistics). The requirements for each program unit are rounded to the next 128-byte boundary.

3. The OTHERWISE verb is treated as if the user coded the ELSE verb.

4. Both the SYSDBOUT and SYSCOUNT DD statements must be specified at execution time.

Debugging and Testing

The execution statistics clearly identify the following areas of the program:

- Untested and weakly tested areas of the program
- The last blocks entered and executed
- Possible sources of unnecessary code
- The most heavily used parts of the program; that is, those parts most susceptible to changes.

OPTIMIZATION METHODS

Based on execution frequency, the following types of optimization can be implemented by the user:

- Resequencing the program
- Insight into SYMDMP
- Common expression elimination
- Backward movement
- Unrolling
- Jamming
- Unswitching
- Incorporating procedures inline
- Tabling
- Efficiency guidelines

Note, however, that each optimization technique can result in more inefficient code if the statistics used in optimizing the program are not representative of the normal program flow. In addition, it is recommended that any optimization methods implemented be documented in the program.

Resequencing the Program

The COBOL Procedure Division should be organized as follows:

1. All frequently-used paragraphs or sections should be located near the routines that use them.
2. All infrequently used paragraphs or sections should be grouped together and apart from frequently-used routines.
3. The most frequently-referenced data items should be placed in the beginning of the Working-Storage Section.

Insight into SYMDMP Output

The area where dynamic symbolic dumps are to be used can be pointed to by the execution statistics. Knowledge of what area of code is executed and how often it is executed should give the user information on what sections should be further investigated.

Common Expression Elimination

This technique is designed to eliminate unnecessary arithmetic calculations. An arithmetic expression calculation is considered unnecessary if it represents a value calculated elsewhere that will always be used without modification. One such example would be an arithmetic expression whose operands are not redefined or reevaluated, but the expression is recalculated.

Backward Movement

This technique facilitates moving calculations and other operations from an area of code frequently executed to an area less frequently executed. For example, an expression calculated within a PERFORM procedure (using a Format 2, 3, or 4 PERFORM statement) which always yields the same value for that PERFORM statement could be calculated inline or in another procedure which would be executed just prior to the regularly executed PERFORM procedure. Another example might be an expression which is calculated in many procedures which are often executed by way of a PERFORM in succession. This expression could be removed from all the procedures and calculated just once prior to the procedures.

Unrolling

Procedures which are frequently executed may be expanded so that the statements within the procedure are repeated, with slight modification, to reduce the procedure overhead. For example,

```
PERFORM    YEARLY-GROSS-CALC VARYING
           WEEK-NO
           FROM 1 BY 1 UNTIL WEEK-NO
           GREATER THAN 52.

YEARLY-GROSS-CALC.
           ADD GROSS-SALARY (WEEK-NO)
           TO YEARLY-GROSS
```

could be replaced by

```
PERFORM    YEARLY-GROSS-CALC VARYING
           WEEK-NO
           FROM 1 BY 4 UNTIL WEEK-NO
           GREATER THAN 52.

YEARLY-GROSS-CALC.
ADD        GROSS-SALARY (WEEK-NO),
           GROSS-SALARY (WEEK-NO+1),
           GROSS-SALARY (WEEK-NO+2),
           GROSS SALARY (WEEK-NO+3) TO
           YEARLY-GROSS.
```

In addition, indexing might be useful in this example.

Jamming

In some instances, two procedures can be merged into one procedure, thereby saving some procedure overhead. An example of this might be replacing

```
MOVE 0 TO WEEK-NUM.
PERFORM YEARLY-GROSS-CAL 52 TIMES.
MOVE 0 TO WEEK-NUM.
PERFORM YEARLY-NET-CAL 52 TIMES.
      .
      .
      .
YEARLY-GROSS-CAL.
   ADD 1 TO WEEK-NUM.
   ADD GROSS-SALARY (WEEK-NUM)
   TO YEARLY-GROSS.
YEARLY-NET-CAL.
   ADD 1 TO WEEK-NUM.
   ADD NET-SALARY (WEEK-NUM) TO
   YEARLY-NET.
```

by

```
MOVE 0 TO WEEK-NUM.
PERFORM YEARLY-CAL 52 TIMES.
      .
      .
      .
YEARLY-CAL.
   ADD 1 TO WEEK-NUM.
   ADD GROSS-SALARY (WEEK-NUM)
   TO YEARLY-GROSS.
   ADD NET-SALARY (WEEK-NUM)
   TO YEARLY-NET.
```

Unswitching

Procedures may contain tests that result in the same action for any set of executions of that procedure. In such a case, the test can be removed from the procedure and the procedure duplicated. For example, if "SWITCH" is not changed within the loop, replace

```
KOUNT=0
PERFORM    JOBS-TOTAL-CAL JOB-NUM TIMES.
      .
      .
      .
JOB-TOTAL-CAL.
   ADD 1 TO KOUNT
   ADD JOB-COST (KOUNT) TO TOTAL-JOB-COST.
   IF SWITCH = 0 ADD JOB-EXPENSE (KOUNT)
      TO TOTAL-EXPENSES ELSE
   ADD JOB-EXPENSE (KOUNT) OVERHEAD TO
      TOTAL-EXPENSES.
   ADD JOB-INCOME (KOUNT) TO TOTAL INCOME.
   IF SWITCH = 0 ADD JOB-PROFIT (KOUNT) TO
      TOTAL-PROFITS ELSE
   COMPUTE TOTAL-PROFITS = TOTAL-PROFITS +
      JOB-INCOME (KOUNT) -
   JOB-COST (KOUNT) - JOB-EXPENSE (KOUNT)
      - OVERHEAD.
```

by

```
KOUNT = 0
   IF SWITCH = 0
   PERFORM JOB-TOTAL-CAL-0
   JOB-NUM TIMES ELSE
   PERFORM JOB-TOTAL-CAL-1
   JOB-NUM TIMES.
      .
      .
      .

JOB-TOTAL-CAL-0.
   ADD 1 TO KOUNT.
   ADD JOB-COST (KOUNT) TO TOTAL-JOB-COST.
   ADD JOB-EXPENSE (KOUNT) TO
      TOTAL/EXPENSES.
   ADD JOB-INCOME (KOUNT) TO TOTAL-INCOME.
   ADD JOB-PROFIT (KOUNT) TO
      TOTAL-PROFITS.
JOB-TOTAL-CAL-1.
   ADD 1 TO KOUNT
   ADD JOB-COST (KOUNT) TO TOTAL-JOB-COST
   ADD JOB-EXPENSE (KOUNT), OVERHEAD TO
      TOTAL-EXPENSE
   ADD JOB-INCOME (KOUNT) TO TOTAL-INCOME
   COMPUTE TOTAL-PROFITS = TOTAL-PROFITS +
      JOB-INCOME (KOUNT)
   - JOB-COST (KOUNT) - JOB-EXPENSE
      (KOUNT) - OVERHEAD.
```

Incorporating Procedures Inline

Based on module size, number of repetitions, modification activities, future expansion considerations, and frequency statistics, small procedures can be moved inline to minimize overhead requirements.

Tabling

This technique is designed to replace many IF statements by one table-look-up statement, or by one computed GO TO statement. For example, if the same data item is tested in many successive IF statements to set the value of another data item to some constant, and the range of tested values of the original data item is limited, then a predetermined table of values could be used to assign the value ofthe second data item. Similarly, many consecutive statements of the form "IF data-item-1 = some-constant GO TO some-procedure" could be replaced by one computed GO TO statement.

Efficiency Guidelines

Based on execution frequency statistics, the following types of coding inefficiencies may be removed.

1. Unaligned decimal places in arithmetic or numeric comparison operands.
2. Different size operands in moves, comparisons, or arithmetic operations.
3. Mixed usage in arithmetic or numeric comparison operands.
4. Display usage in arithmetic operands or one numeric operand and one display operand in a comparison.
5. SYNC missing for COMP or COMP-1, -2, or -4 items.
6. Inefficient COMP type picture; that is, no sign or more than 9 digits in a COMP item and no sign, even number of digits, or more than 16 digits in COMP-3 items.
7. Certain calls to object-time subroutines.
8. Indexing instead of subscripting and vice versa.
9. Noncomputational subscripts.

PROGRAMMING TECHNIQUES

Some techniques for increasing the efficiency of a COBOL program are described in this chapter. It is divided into seven parts. The first four parts deal in general with coding a COBOL program. The fifth is concerned with the Report Writer feature, the sixth with table handling, and the seventh with queue structure description.

GENERAL CONSIDERATIONS

Spacing the Source Program Listing

There are four statements that can be coded in any or all of the four divisions of a source program: SKIP1, SKIP2, SKIP3, and EJECT. These statements provide the user with the ability to control the spacing of a source listing and thereby improve its readability.

Coding Considerations

These suggestions will aid efficiency:

- Use the RES option and place frequently used COBOL subroutines into the Paged Link Pack Area.
- Avoid repetitive sequences of

  CALL

  CANCEL

  for the same subprograms.
- If a short subprogram is referenced only once or twice (and is not an unusual situation routine), then its code should be incorporated in the calling program, if convenient.
- Subprograms should be loaded near the programs which use them. This can be done via linkage editor control cards.
- Segmentation in many cases is no longer desirable.
- Data-names of constant value should be grouped together. Data-names whose values vary during execution should also be grouped together and should be separate from those of constant value.
- All frequently used subroutines should be loaded near each other. This can be done via linkage editor control cards.
- FDs for files that will be opened at the same time should be grouped together.
- The most frequently referenced data items should be placed in the beginning of the Working Storage Section.
- The COBOL Procedure Division should be organized generally as follows:

  a. All frequently used paragraphs or sections should be located near the routines that use them.

  b. All infrequently used paragraphs or sections should be grouped together and apart from frequently used routines.

ENVIRONMENT DIVISION

APPLY WRITE-ONLY Clause

To make optimum use of buffer space allocated when creating a physical sequential file with blocked V-mode records, the programmer may use the APPLY WRITE-ONLY clause for the file. Use of this option causes a buffer to be truncated only when the next record does not fit in the buffer. (If the APPLY WRITE-ONLY clause is not specified, the buffer is truncated when the maximum size record will not fit in the space remaining in the buffer.) When using APPLY WRITE-ONLY, all the WRITE statements must have FROM options. None of the subfields of the associated records may be referred to by procedure statements and they may not be the object of the DEPENDING ON option in an OCCURS clause.

QSAM Spanned Records

Except for APPLY WRITE-ONLY, ADVANCING, POSITIONING, and APPLY RECORD-OVERFLOW, all the options for variable length record files apply to spanned records.

APPLY RECORD-OVERFLOW Clause

For non-VSAM files, the APPLY RECORD-OVERFLOW clause makes more efficient use of direct access storage space by using the Track Overflow feature. If APPLY RECORD-OVERFLOW is specified, a record that does not fit on a track will be partially written on that track and the remainder will be written on the next available track.

The use of the APPLY RECORD-OVERFLOW option requires that Track Overflow be specified at system generation time.

APPLY CORE-INDEX Clause

To minimize processing time with indexed sequential files accessed randomly, the programmer should use the APPLY CORE-INDEX clause. Use of this option causes the highest level index to be brought into main storage for input/output operations. This speeds processing by eliminating the extra time needed to search the index on the volume.

BDAM-W File Organization

The use of BDAM-W for file organization results in less system generated coding than for BDAM-D. When BDAM-D is used and a WRITE statement is issued, extra code must be generated to compare the contents of the ACTUAL KEY of the WRITE statement with the key of the preceding READ statement to determine whether the system should add or update a record. If the keys are the same the record is updated. If the keys are different the record is added.

BDAM-W eliminates this comparison step. The system adds a record when a WRITE statement is issued and updates a record when a REWRITE statement is issued. Wh2

DATA DIVISION

OVERALL CONSIDERATIONS

Maximum Data Division Size

The absolute maximum of the Data Division (excluding Linkage Section) is 1,044,480 bytes. (This includes File, Working-Storage, Communication, and Report Sections.) Other maximum lengths (in bytes) are:

- data description entry -- 32,767
- variable length table -- 32,767
- fixed-length group item (including fixed-length table) in Working-Storage or Linkage Section -- 131,071

Prefixes

Assign a prefix to each level-01 item in a program, and use this prefix on every subordinate item (except FILLER) to associate a file with its records and work-areas. For example, MASTER is the prefix used here:

```
FILE SECTION.
FD  MASTER-INPUT-FILE
    .
    .
    .
01  MASTER-INPUT-RECORD.
    .
    .
    .
WORKING-STORAGE SECTION.
01  MASTER-WORK-AREA.
    05  MASTER-PAYROLL PICTURE 9(3).
    05  MASTER-SSNO PICTURE 9(9).
```

If files or work-areas have the same fields, use the prefix to distinguish between them. For example, if three files all have a date field, instead of DATE, DAT, and DA-TE, use MASTER-DATE, DETAIL-DATE, and REPORT-DATE. Using a unique prefix for each level-01 and all subordinate fields makes it easier for a person unfamiliar with the program to find fields in the program listing, and to know which fields are logically part of the same record or area.

When using the MOVE statement with the CORRESPONDING option and referring to individual fields, redefine or rename

"corresponding" names with the prefixed unique names. This technique eliminates excessive qualifying. For example:

```
01   MST-WORK-AREA.
     05  SAME-NAMES.      (***)
         10  LAST-NAME  PIC...
         10  FIRST-NAME  PIC...
         10  PAYROLL  PIC...
         .
         .
         .
     05  DIFF-NAMES REDEFINES SAME-NAMES.
         10  MST-LAST-NAME  PIC...
         10  MST-FIRST-NAME  PIC...
         10  MST-PAYROLL  PIC...
01   RPT-WORK-AREA.
     05  SAME-NAMES.      (***)
         10  PAYROLL  PIC...
         10  FILLER  PIC...
         10  FIRST-NAME  PIC...
         10  FILLER  PIC...
         10  LAST-NAME  PIC...
         .
         .
         .
PROCEDURE DIVISION.
         .
         .
         .
     IF MST-PAYROLL IS EQUAL TO HDQ-PAYROLL
     AND MST-LAST-NAME
     IS NOT EQUAL TO PRRV-LAST-NAME
     MOVE CORRESPONDING
     MST-WORK-AREA
     TO RPT-WORK-AREA.
```

Note: Fields marked with a triple asterisk (***) in the foregoing listing must have exactly the same names for their subordinate fields in order to be considered corresponding. The same names must not be the redefining ones, or they will not be considered to correspond.

Level Numbers

The programmer should use widely incremented level numbers, i.e., 01, 05, 10, 15, etc., instead of 01, 02, 03, 04, etc., in order to allow room for future insertions of group levels. For readability, indent level numbers. Use level-88 numbers for codes. Then, if the codes must be changed, the Procedure Division coding for tests need not be changed.

FILE SECTION

RECORD CONTAINS Clause

The programmer should use the RECORD CONTAINS integer CHARACTERS clause in order to save himself as well as any future programmer the task of counting the data record description positions. Also, the compiler can then diagnose errors if the data record description conflicts with the RECORD CONTAINS clause.

COMMUNICATION SECTION

The Communication Section of a COBOL program must be specified if the program is to take advantage of the Teleprocessing Feature (TP). Through the inclusion of Communication Description (CD) entries, the programmer establishes communication between the COBOL object program and the Message Control Program (MCP).

CD Entries

When specified, the Communication Section must contain at least one CD entry. For example, a single CD entry would be sufficient for applications with either an input or an output message but not both. A COBOL TP program that is both to receive and to send messages must contain at least two[1] CD entries, as below.

```
CD cd-name FOR INPUT.
CD cd-name FOR OUTPUT.
```

The CD entry may instead be pre-written and included in the user-created library. The programmer may then include the entry in a COBOL program by means of a COPY statement.

```
CD cd-name COPY library-name.
```

The input CD contains such information as input queue and sub-queue names, message date and time, the source, the message text length, the end key, the message status key, and the message count. The output CD contains the text length, the destination count, a status key and error key (both possibly repeating), and the name of the output queue. For information about the CD formats possible, see the publication IBM VS COBOL for OS/VS.

---

[1] Multiple input and output CD entries may be specified.

Note: The required inclusion of the parameter DATE=YES in all input TPROCESS entries whose destination is a COBOL program results in the placing of the date and time of message entry in the input CD (see the section "Additional Interface Considerations" in the chapter entitled "Using the Teleprocessing Feature").

WORKING-STORAGE SECTION

Separate Modules

In a large program, the programmer should plan ahead for breaking the programs into separately compiled modules, as follows:

1. When employing separate modules, an attempt should be made to combine entries of each Working-Storage Section into a single level-01 record (or one level-01 record for each 32K bytes). Logical record areas can be indicated by use of level-02, level-03 etc., entries. A CALL statement with the USING option is more efficient when a single item is passed than when many level-01 and/or level-77 items are passed. When this method is employed, mistakes are more easily avoided.

2. Areas that do not have VALUE clauses should be separated from areas that do need VALUE clauses. VALUE clauses (except for level-88 items) are invalid in the Linkage Section.

3. When the Working-Storage Section is one level-01 item with no VALUE clauses, the COPY statement can easily be used to include the item as the description of a Linkage Section in a separately compiled module.

4. See "Use of Segmentation Feature" for more information on how to modularize the Procedure Division of a COBOL program.

Locating the Working-Storage Section in Dumps

When any one or more of the options PMAP, CLIST, and DMAP are specified, both the location and the length (in hexadecimal) of the Working-Storage Section, if any, are provided (see the section "Options for the Compiler" in the chapter entitled "Job Control Procedures").

Alternatively, the programmer may locate this section in object-time dumps by including the following two statements in the program, in the order given:

```
77  FILLER PICTURE X(44), VALUE "PROGRAM
    XXXXXXXX WORKING-STORAGE BEGINS HERE".

01  FILLER PICTURE X(42), VALUE "PROGRAM
    XXXXXXXX WORKING-STORAGE ENDS HERE".
```

These two nonnumeric literals will appear in all dumps of the program, delineating the Working-Storage Section. The program-name specified in the PROGRAM-ID clause should replace the XXXXXXXX in the literal.

DATA DESCRIPTION

The Procedure Division operations that most often require adjustment of data items include the MOVE statement, the IF statement when used in a relation test, and arithmetic operations. Efficient use of data description clauses, such as REDEFINES, RENAMES PICTURE, and USAGE, avoids the generation of extra code.

REDEFINES Clause

REUSING DATA AREAS: The main storage area can be used more efficiently by writing different data descriptions for the same data area. For example, the coding that follows shows how the same area can be used as a work area for the records of several input files that are not processed concurrently:

```
WORKING-STORAGE SECTION.
01  WORK-AREA-FILE1.
     (largest record description for FILE1)
.
.
.

01  WORK-AREA-FILE2 REDEFINES
    WORK-AREA-FILE1.
     (largest record description for FILE2)
.
.
.
```

ALTERNATE GROUPINGS AND DESCRIPTIONS: Program data can often be described more efficiently by providing alternate groupings or data descriptions for the same data. For example, a program refers to both a field and its subfields, each of which is more efficiently described with a

different usage. This can be done with the REDEFINES clause as follows:

```
01  PAYROLL-RECORD.
    05  EMPLOYEE-RECORD PICTURE X(28).
    05  EMPLOYEE-FIELD REDEFINES
        EMPLOYEE-RECORD.
        10  NAME          PIC X(23).
        10  NUMBERX       PIC S9(5) COMP SYNC.
    05  DATE-RECORD       PIC X(10).
```

As an example of different data descriptions specified for the same data, the following illustrates how a table (TABLEA) can be initialized:

```
    05  VALUE-A.
        10  A1 PICTURE S9(9) COMPUTATIONAL
            VALUE IS ZEROES.
        10  A2 PICTURE S9(9) COMPUTATIONAL
            VALUE IS 1.
        .
        .
        .
        10  A100 PICTURE S9(9)
            COMPUTATIONAL VALUE IS 99.

    05  TABLEA REDEFINES VALUE-A
        PICTURE S9(9) COMPUTATIONAL
         OCCURS 100 TIMES.
```

Note: Caution should be exercised when a redefining or redefined item is used as a subscript. If the value of a redefining item is changed in the Procedure Division and the redefined item is used as a subscript, or vice versa, then no new calculation for the subscript is performed.

RENAMES Clause

By permitting a programmer to rename various fields, the RENAMES clause enables alternate, possibly overlapping, groupings of elementary data. The following example shows how three fields of a record can be renamed:

```
01  OUT-REC.
    05  FIELD-X.
        10  SUMMARY-GROUPX.
            15  FILE-1 PICTURE X.
            15  FILE-2 PICTURE X.
            15  FILE-3 PICTURE X.
    05  FIELD-Y.
        10  SUMMARY-GROUPY.
            15  FILE-1 PICTURE X
            15  FILE-2 PICTURE X.
            15  FILE-3 PICTURE X.
    05  FIELD-Z.
        10  SUMMARY-GROUPZ.
            15  FILE-1 PICTURE X.
            15  FILE-2 PICTURE X.
            15  FILE-3 PICTURE X.
66  SUM-X RENAMES FIELD-X.
66  SUM-XY RENAMES FIELD-X THRU FIELD-Y.
66  SUM-XYZ RENAMES FIELD-X THRU FIELD-Z.
```

If each level-15 item contained either an A or an I, a programmer could find out how many files contained an A by doing a complete tally of files in OUT-REC:

```
EXAMINE SUM-XYZ TALLYING ALL "A"
```

Or a programmer might just want to tally the files in FIELD-X:

```
EXAMINE SUM-X TALLYING ALL "A"
```

In short, renaming fields can lead to better utilization of coding, storage, and time by facilitating the streamlining of Procedure Division operations.

PICTURE Clause

DECIMAL-POINT ALIGNMENT: Procedure Division operations are most efficient when the decimal positions of the data items involved are aligned. If they are not, the compiler generates instructions to align the decimal positions before any operations involving the data items can be executed. This is referred to as scaling.

Assume, for example, that a program contains the following instructions:

```
WORKING-STORAGE SECTION.
77  A PICTURE S999V99.
77  B PICTURE S99V9.
.
.
.

PROCEDURE DIVISION.
.
.
    ADD A TO B.
```

Time and internal storage space are saved by defining B as:

```
77 B PICTURE S99V99.
```

If it is inefficient to define B differently, a one-time conversion can be done, as explained in "Data Format Conversion."

FIELDS OF UNEQUAL LENGTH: When a data item is moved to another data item of a different length, the following should be considered:

- If the items are external decimal items, the compiler generates instructions to insert zeros in the high-order positions of the receiving field when it is the larger.

- If the items are nonnumeric, the compiler generates instructions to insert spaces in the low-order positions of the receiving field (or the high-order positions if the JUSTIFIED RIGHT clause is specified. This generation of extra instructions can be avoided if the sending field is described with a length equal to or greater than the receiving field.

Use of Sign: The absence or presence of a plus or minus sign in the description of an arithmetic field often can affect the efficiency of a program. The following paragraphs discuss some of the considerations.

Decimal Items: The sign position in an internal or external decimal item can contain:

1. A plus or minus sign. If S is specified in the PICTURE clause, a plus or minus sign is inserted when either of the following conditions prevails:

   a. The item is in the Working-Storage Section and a VALUE clause has been specified.

   b. A value for the item is assigned as a result of an arithmetic operation during execution of the program.

   If an external decimal item is punched, printed, or displayed, an overpunch will appear in the low-order digit. In EBCDIC, the configuration for low-order zeros normally is a nonprintable character. Low-order digits of positive values will be represented by one of the letters A through I (digits 1 through 9); low-order digits of negative values will be represented by one of the letters J through R (digits 1 through 9).

2. A hexadecimal F. If S is not specified in the PICTURE clause, an F is inserted in the sign position when either of following conditions exists:

   a. The item is in the Working-Storage Section and a VALUE clause has been specified.

   b. A value for the item is developed during the execution of the program.

   An F is treated as positive, but is not an overpunch.

3. An invalid configuration. If an internal or external decimal item contains an invalid configuration in the sign position, and if the item is involved in a Procedure Division operation, the program will be abnormally terminated.

Items for which no S has been specified (unsigned items) are treated as absolute values. Whenever a value (signed or unsigned) is stored in, or moved in an elementary move to an unsigned item, a hexadecimal F is stored in the sign position of the unsigned item. For example, if an arithmetic operation involves signed operands and an unsigned result field, compiler-generated code will insert an F in the sign position of the result field when the result is stored.

For internal and external decimal items used as input, it is the user's responsibility to ensure that the input data is valid. The compiler does not generate a test to ensure that the configuration in the sign position is valid.

When a group item is involved in a move, the data is moved without regard to the level structure of the group items involved. The possibility exists that the configuration in the sign position of a subordinate numeric item may be destroyed. Therefore, caution should be exercised in moves involving group items with subordinate numeric fields or with other group operations such as READ or ACCEPT.

SIGN Clause

This clause, which specifies both the position and the mode of the operational sign for a numeric data description entry, is required only when an explicit description of the sign's properties is necessary. The SIGN clause may be specified for either a numeric data description entry whose PICTURE contains the character S or a group item that contains at least one such numeric data description entry.

The numeric data description entries to which the SIGN clause applies must be described, implicitly or explicitly, as USAGE IS DISPLAY. Only one SIGN clause may be associated with any given numeric data description entry.

The format of the SIGN clause is as follows:

SIGN IS {LEADING / TRAILING} [SEPARATE CHARACTER]

Use of the SEPARATE CHARACTER Option: The programmer can elect to consider the character S in the PICTURE character string as a separate character or not, as he chooses. If the SEPARATE CHARACTER option is specified:

- The position of the character S is not taken to be a digit position.
- The character S is counted in determining the size of the data item.
- The characters '+' and '-' are used for the positive and the negative operational signs, respectively.
- If neither the character '+' nor the character '-' is present in the data at object time, an error takes place and the program ABENDS.

Whether or not the SEPARATE CHARACTER option is in effect, the operational sign is assumed to be associated with either the LEADING or the TRAILING digit position, as specified, of the elementary numeric data item.

USAGE Clause

This clause should be written at the highest level possible.

DATA FORMAT CONVERSION: Operations involving mixed, elementary numeric data formats require conversion to a common format. This usually means that additional storage is used and execution time is increased. The code generated must often move data to an internal work area, perform any necessary conversion, and then execute the indicated operation. Often, too, the result may have to be converted in the same way (see Figure 107).

If it is impractical to use the same data formats throughout a program, and if two data items of different formats are frequently used together, a one-time conversion can be effected. For example, if A is defined as a COMPUTATIONAL item and B as a COMPUTATIONAL-3 item, A can be moved to a work area that has been defined as COMPUTATIONAL-3. This move causes the data in A to be converted to COMPUTATIONAL-3. Whenever A and B are used in a Procedure Division operation, reference can be made to the work area rather than to A. Using this technique, the conversion is performed only once, instead of each time an operation is performed.

The following eight cases show how data conversions are handled on mixed elementary items for names, data comparisons, and arithmetic operations. Moves to and from group items, without the CORRESPONDING option, as well as comparisons involving group items, are done without conversion.

Numeric DISPLAY to COMPUTATIONAL-3:

To Move Data: Converts DISPLAY data to COMPUTATIONAL-3 data.

To Compare Data: Converts DISPLAY data to COMPUTATIONAL-3 data.

To Perform Arithmetic Operations: Converts DISPLAY data to COMPUTATIONAL-3 data.

Numeric DISPLAY to COMPUTATIONAL:

To Move Data: Converts DISPLAY data to COMPUTATIONAL-3 data and then to COMPUTATIONAL data.

To Compare Data: Converts DISPLAY to COMPUTATIONAL-3 and then to COMPUTATIONAL or converts both DISPLAY and COMPUTATIONAL data to COMPUTATIONAL-3 data.

To Perform Arithmetic Operations: Converts DISPLAY data to COMPUTATIONAL-3 or COMPUTATIONAL data.

COMPUTATIONAL-3 to COMPUTATIONAL:

To Move Data: Moves COMPUTATIONAL-3 data to a work field and then converts COMPUTATIONAL-3 data to COMPUTATIONAL data.

To Compare Data: Converts COMPUTATIONAL data to COMPUTATIONAL-3 or vice versa, depending on the size of the field.

To Perform Arithmetic Operations: Converts COMPUTATIONAL data to COMPUTATIONAL-3 or vice versa, depending on the size of the field.

COMPUTATIONAL to COMPUTATIONAL-3:

To Move Data: Converts COMPUTATIONAL data to COMPUTATIONAL-3 data in a work field, then moves the work field.

To Compare Data: Converts COMPUTATIONAL to COMPUTATIONAL-3 data or vice versa, depending on the size of the field.

Usage	Bytes Required	Typical Use	Converted for Arithmetic Operations	Special Characteristics
DISPLAY (external decimal)	1 per digit (except for V)	Input from cards, output to cards, listings	Yes	May be used for numeric fields up to 18 digits long. Fields over 15 digits require extra instructions if used in computations.
DISPLAY (external floating point)	1 per character (except for V)	Input from cards, output to cards, listings	Yes	Converted to COMPUTATIONAL-2 format via COBOL library subroutine.
COMP-3 (internal decimal)	1 byte per 2 digits plus 1 byte for the low-order digit and sign	Input to a report item Arithmetic fields Work areas	Sometimes when a small COMP-3 item is used with a small COMP item.	Requires less space than DISPLAY. Convenient form for decimal alignment. Can be used in arithmetic computations without conversion. Fields over 15 digits require a subroutine when used in computations.
COMP (binary)	2 if 1≤N≤4 4 if 5≤N≤9 8 if 10≤N≤18 where N is the number of 9s in the PICTURE clause	Subscripting Arithmetic fields	Sometimes for both mixed and unmixed usages	Rounding and testing for the ON SIZE ERROR condition are cumbersome if calculated result is greater than 9(9). Fields of over 9 digits require more handling.
COMP-1 (internal floating point)	4 (short-precision)	Fractional exponentiation	No	Tends to produce less accuracy if more than 17 significant digits are required and if the exponent is big. Requires floating-point feature.
COMP-2 (internal floating point)	8 (long-precision)	Fractional exponentiation when more precision is required	No	Same as COMPUTATIONAL-1

Figure 107. Data Format Conversion

To Perform Arithmetic Operations: Converts COMPUTATIONAL to COMPUTATIONAL-3 data or vice versa, depending on the size of the field.

COMPUTATIONAL to Numeric DISPLAY:

To Move Data: Converts COMPUTATIONAL data to COMPUTATIONAL-3 data and then to DISPLAY data.

To Compare Data: Converts DISPLAY to COMPUTATIONAL or both COMPUTATIONAL and DISPLAY data to COMPUTATIONAL-3 data, depending on the size of the field.

To Perform Arithmetic Operations: Depending on the size of the field, converts DISPLAY data to COMPUTATIONAL data, or both DISPLAY and COMPUTATIONAL data to COMPUTATIONAL-3 data in which case the result is generated in a COMPUTATIONAL-3 work area and then converted and moved to the DISPLAY result field.

COMPUTATIONAL-3 to Numeric DISPLAY:

To Move Data: Converts COMPUTATIONAL-3 data to DISPLAY data.

To Compare Data: Converts DISPLAY data to COMPUTATIONAL-3 data. The result is generated in a COMPUTATIONAL-3 work area and is then converted and moved to the DISPLAY result field.

Numeric DISPLAY to Numeric DISPLAY:

To Perform Arithmetic Operations: Converts all DISPLAY data to COMPUTATIONAL-3 data. The result is generated in a COMPUTATIONAL-3 work area and is then converted to DISPLAY and moved to the DISPLAY result field.

External Floating-Point to Any Other: When an external floating-point item is to be used in an airthmetic operation or in data manipulation, precision errors may occur due to required conversions.

Internal Floating-Point to Any Other: When an item described as COMPUTATIONAL-1 or COMPUTATIONAL-2 (internal floating-point) is used in an operation with another data format, the item in the other data format is always converted to internal floating-point. If necessary, the internal floating-point result is then converted to the format of the other data item.

Special Considerations for DISPLAY and COMPUTATIONAL Fields

NUMERIC DISPLAY FIELDS: Zeros are not inserted into numeric DISPLAY fields by the instruction set. When numeric DISPLAY data is moved, the compiler generates instructions that insert any necessary zeros into the DISPLAY fields. When numeric DISPLAY data is compared, and one field is smaller than the other, the compiler generates instructions to move the smaller item to a work area where zeros are inserted.

COMPUTATIONAL-1 AND COMPUTATIONAL-2 FIELDS: If an arithmetic operation involves a mixture of short-precision and long-precision fields, the compiler generates instructions to expand the short-precision field to a long-precision field before the operation is executed.

COMPUTATIONAL-3 FIELDS: The compiler does not have to generate instructions to insert high-order zeros for ADD and SUBTRACT statements that involve COMPUTATIONAL-3 data. The zeros are inserted by the instruction set.

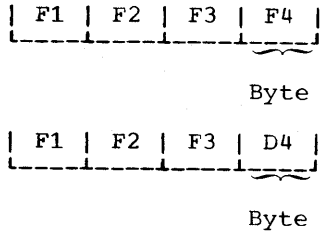
Data Formats in the Computer

The various COBOL data formats and how they appear in the computer in EBCDIC (Extended Binary-Coded-Decimal Interchange Code) format are illustrated by the following examples. More detailed information about these data formats appears in the publication IBM System/370 Principles of Operation, Order No. GA22-7000.

Numeric DISPLAY (External Decimal): Suppose the value of an item is -1234, and the PICTURE and USAGE are:

```
PICTURE 9999 DISPLAY.

or

PICTURE S9999 DISPLAY.
```

The item appears in the computer in the following forms respectively:



Hexadecimal F is treated arithmetically as plus in the low-order byte. The hexadecimal character D represents a negative sign.

COMPUTATIONAL-3 (Internal Decimal): Suppose the value of an item is +1234, and its PICTURE and USAGE are:

```
PICTURE 9999 COMPUTATIONAL-3.

        or

PICTURE S9999 COMPUTATIONAL-3.
```

The item appears in the computer in the following forms, respectively:

```
| 01 | 23 | 4F |
        Byte

| 01 | 23 | 4C |
        Byte
```

Hexadecimal F is treated arithmetically as positive. The hexadecimal character C represents a plus sign.

Note: Since the low-order byte of an internal decimal number always contains a sign field, an item with an odd number of digits can be stored more efficiently than an item with an even number of digits. Note that a leading zero is inserted in the foregoing example.

COMPUTATIONAL (Binary): Suppose the value of an item is 1234, and its PICTURE and USAGE are:

```
PICTURE S9999 COMPUTATIONAL.
```

The item appears in the computer in the following form:

```
| 0000 | 0100 | 1101 | 0010 |
  sign
 position
```

A 0-bit in the sign position means the number is positive. Negative numbers are represented in two's complement form; thus, the sign position of a negative number will always contain a 1-bit.

For example -1234 would appear as follows:

```
| 1111 | 1011 | 0010 | 1110 |
  sign
 position
```

Binary Item Manipulation: A binary item is allocated storage ranging from one halfword to two words, depending on the number of 9s in its PICTURE. Figure 108 is an illustration of how the compiler allocates this storage. Note that it is possible for a value larger than that implied by the PICTURE to be stored in the item. For example, PICTURE S9(4) implies a maximum value of 9,999, although it could actually hold the number 32,767.

Because most binary items are manipulated according to their allotted storage capacity, the programmer can ignore this situation. For the following reasons, however, there are some cases where he must be careful of his data:

1. When the ON SIZE ERROR option is used, the size test is made on the basis of the maximum value allowed by the picture of the result field. If a size error condition exists, the value of the result field is not altered and control is given to the imperative statements specified by the error option.

2. When a binary item is displayed or exhibited, the value used is a function of the number of 9s specified in the PICTURE clause.

3. When the actual value of a positive number is significantly larger than its picture value, a 1 could result in the sign position of the item, causing the item to be treated as a negative number in subsequent operations.

Figure 109 illustrates three binary manipulations. In each case, the result field is an item described as PICTURE S9 COMPUTATIONAL. One halfword of storage has been allocated; and no ON SIZE ERROR option is involved. Note that if the ON SIZE ERROR option had been specified, it would have been executed for cases B and C.

COMPUTATIONAL-1 or COMPUTATIONAL-2 (Floating Point): Suppose the value of an item is +1234, and that its USAGE is COMPUTATIONAL-1, the item appears in the computer in the following form:

```
|0|100  0011|0100 1101 0010 0000 0000 0000|
 S 1      7 8                            31
```

PICTURE	Maximum Working Value	Assigned Storage
S9 through S9(4)	32,767	one halfword
S9(5) through S9(9)	2,147,483,647	one fullword
S9(10) through S9(18)	9,223,372,036,854,775,807	two fullwords

Figure 108. Relationship of PICTURE to Storage Allocation

Case	Hexadecimal Result of Binary Calculation	Decimal Equivalent	Actual Decimal Value in Halfword of Storage	Display or Exhibit Value
A	0008	8	+8	8
B	000A	10	+10	0
C	C350	50000	-15536	6

Figure 109. Treatment of Varying Values in a Data Item of PICTURE S9

S is the sign position of the number.

A 0-bit in the sign position indicates that the sign is plus.

A 1-bit in the sign position indicates that the sign is minus.

Bits 1 through 7 are the exponent (characteristic) of the number.

Bits 8 through 31 are the fraction (mantissa) of the number.

This form of data is referred to as floating-point. The example illustrates short-precision floating-point data (COMPUTATIONAL-1). In long-precision (COMPUTATIONAL-2), the fraction length is 56 bits. (For a detailed explanation of floating-point representation, see the publication IBM System/370 Principles of Operation, Order No. GA22-7000.

PROCEDURE DIVISION

A program can often be made more efficient or easier to debug in the Procedure Division with some of the techniques described below.

MODULARIZING THE PROCEDURE DIVISION

When the Procedure Division is modularized, programs are easier to maintain and document. In addition, modularization makes it simple to break down a program using the segmentation feature, thereby resulting in a more efficient segmented program. Modularization of the Procedure Division involves organizing it into at least three functional levels: a main-line routine, processing subroutines, and input/output subroutines.

Main-Line Routine

This routine should be short, simple, and contain all the major logical decisions of the program. This routine controls which second-level subroutines are executed and in what order. All second-level subroutines should be invoked from the main-line routine by PERFORM statements.

Processing Subroutines

These should be broken down into as many functional levels as necessary, depending on the complexity of the program. These must be completely closed subroutines, with one entry point and one exit point. The entry point should be the first statement of the subroutine. The exit point should be the EXIT statement. The processing subroutines can perform only lower level subroutines; return to the higher level subroutine (processing subroutine) must be made by a GO TO statement, which references the EXIT statement.

Input/Output Subroutines

These should be the lowest level subroutines, since all higher level subroutines should have access to them. There should be one OPEN subroutine and one CLOSE subroutine for the program, and only one functional (READ or WRITE) subroutine for each file. One READ or WRITE subroutine per file, which is always performed, has several advantages:

1. Coding can be added to count records on a file, transform blanks into zeros, check for 9s padding, etc.

2. Input and output files can be reformatted without changing the logic of the program.

3. DEBUG statements can be added during testing to create input or to DISPLAY formatted output, instead of having to create a test file.

COLLATING SEQUENCES

The combination of the PROGRAM COLLATING SEQUENCE clause and the SPECIAL NAMES alphabet-name clause(s) offers the programmer flexibility in establishing or altering the collating sequence used in the following operations: the various forms of non-numeric comparisons, HIGH/LOW-VALUE, SEARCH ALL, and SORT/MERGE. The alphabet used may be EBCDIC (denoted as NATIVE, which is also the default), ASCII (denoted as STANDARD-1), or one or more programmer-defined alterations of the EBCDIC sequence.

The alphabet identified through the PROGRAM COLLATING SEQUENCE clause will be used for all occurences of non-numeric compares, HIGH/LOW-VALUE, and SEARCH ALL. However, each separate SORT/MERGE operation can override that general specification by including its own COLLATING SEQUENCE clause.

For QSAM files, the CODE-SET clause of the FD statement can be used to identify the file as being either EBCDIC or ASCII. When an ASCII file is identified in this manner, the corresponding DD card need not specify DCB=(OPTCD=Q...) or DCB=(RECFM=D...).

USE OF THE UPSI SWITCHES

The flexibility of programs can be greatly increased through use of the execution-time switches UPSI-0, UPSI-1,...UPSI-7 (see "Options for Execution" in the section "Job Control Procedures"). A program can be written to deal with many potential situations, and then directed (when it is to begin execution) as to which particular situations are current during that run.

For example, a program might be designed to read File A and create File B; these two procedures are always to be done. However, on certain irregular occasions, a special report is also to be written to File C. These special occasions are externally determined--the program could not normally know of or internally determine them. But by using an UPSI switch, this difficulty can be overcome.

The code for the production of File C is written predicated on an IF statement that tests the associated condition name of an UPSI switch: if on, the special report is produced; if off, it is skipped. For everyday executions of the program, the installation supplies the UPSI object-time parameter with that particular switch set to zero. When occasion demands that the special report be printed, the installation sets the UPSI object-time parameter switch to one, and the program's logic then writes out the report.

Because there are eight UPSI switches, controlled variation of a much more complex nature can be obtained by a user.

INTERCEPTING I/O ERRORS

COBOL offers a variety of techniques the programmer can employ to intercept and handle I/O error situations. Use of these techniques (INVALID KEY, USE AFTER ERROR/EXCEPTION, and FILE STATUS) gives a programmer not only the power to prevent abnormal termination, but also flexibility in his response. FILE STATUS--valid for VSAM and QSAM files--can be used separately or in combination with one of the other two techniques. COBOL automatically fills in the key field immediately after every I/O operation, so that the program can be designed to examine it and take action accordingly. (If FILE STATUS is specified but not interrogated by a program after an I/O operation, results are unpredicatble.)

Errors That May Escape Detection

If a logic error occurs because the user attempts a READ or WRITE against an unopened file, an associated USE ERROR declarative will not get control. If such an error occurs when the file has been closed but not reopened, the wrong USE ERROR declarative may get control. However, such a situation can be circumvented by using FILE STATUS to test for successful open before performing the READ/WRITE.

INTERMEDIATE RESULTS

The compiler treats arithmetic statements as a succession of operations and sets up intermediate result fields to contain the results of these operations. Examples of such statements are the arithmetic statements, and statements containing arithmetic expressions. The publication IBM VS COBOL for OS/VS describes the algorithms used by the compiler to determine the number of places reserved for intermediate result fields.

Intermediate Results and Binary Data Items

If an operation involving binary operands requires an intermediate result greater than 18 digits, the compiler converts the operands to internal decimal before performing the operation. If the result field is binary, the result will be converted from internal decimal to binary.

If an intermediate result will not be greater than nine digits, the operation is performed most efficiently on binary data fields.

Intermediate Results and COBOL Library Subroutines

If a decimal multiplication operation requires an intermediate result greater than 30 digits, a COBOL library subroutine is used to perform the multiplication. The result of this multiplication is then truncated to 30 digits.

A COBOL library subroutine is used to perform division if:

1. the scaled divisor is equal to or greater than 15 digits.
2. the length of the scaled divisor plus the length of the scaled dividend is greater than 16 bytes. The lengths of the operands are internal decimal.
3. the scaled dividend is greater than 30 digits. (A scaled dividend is a number that has been multiplied by a power of ten in order to obtain the desired number of decimal places in the quotient.)

Intermediate Results Greater than 30 Digits

Whenever the number of digits in a decimal intermediate result is greater than 30, the field is truncated to 30 digits. A warning message will be generated at compile time, and program flow will not be interrupted at execution time. This truncation may cause a result to be incorrect.

If binary or internal decimal data is in accord with its data description, no interrupt can occur because of an overflow condition in an intermediate result. This is due to the truncation described in the preceding paragraph.

If the possibility exists that an intermediate result field may exceed 30 digits, truncation can be avoided by the specification of floating-point operands (COMPUTATIONAL-1 or COMPUTATIONAL-2); however, accuracy may not be maintained.

Intermediate Results and Floating-Point Data Items

If a floating-point operand has an intermediate result field in which exponent overflow occurs, the job will be abnormally terminated.

If the exponent is a floating-point item or has a PICTURE specifying decimal places, the subroutine ILBOFPW0 is called and the exponentiation is executed in floating-point arithmetic.

Note: The base is always treated as a positive number, regardless of sign, and the answer is always a positive number. Caution should therefore be exercised when using noninteger exponents.

Caution: A COMPUTE or DIVIDE statement which stores its intermediate results in only a floating-point field may produce a slightly different final answer than a similar COMPUTE or DIVIDE statement which stores its intermediate results in both a floating-point and a fixed point field. This is because of the different precisions of the fixed- and floating-point fields.

Intermediate Results and the ON SIZE ERROR Option

The ON SIZE ERROR option applies only to the final calculated results and not to intermediate result fields.

VERBS

CALL Statement

The CALL statement permits communication between a COBOL object program and one or more COBOL subprograms or other language subprograms. A called program may be entered either at the beginning of the Procedure Division or later in the program. When a subprogram is called, it may already be main storage resident and be link-edited with the main program, or it may be specified as dynamic and link-edited into a separate load module. Dynamic loading, via the CALL statement, enables the user to load a subprogram only when it is actually needed.

The first dynamic call to a subprogram brings in a fresh copy of that subprogram. Any subsequent calls to the same subprogram, by either the original caller or another subprogram in the same region/partition, cause a branch to the same copy of the subprogram in its last-used state, until the user deletes it (see the section on the "CANCEL Statement").

The ON OVERFLOW phrase can be included in the CALL statement to circumvent abnormal termination when there is insufficient main storage available for dynamically loading the called program. The imperative statement associated with the ON OVERFLOW will gain control in such a situation, allowing the user to handle the problem in a manner of his own choosing and continue execution. (The conditions handled by ON OVERFLOW are equivalent to the following completion codes: 106C, 804, 80A, and 878.)

For examples of both static and dynamic CALL statements, see the section "Dynamic Subprogram Considerations" in the chapter entitled "Calling and Called Programs."

CANCEL Statement

The CANCEL statement permits dynamic deletion of COBOL subprograms from the COBOL processing environment. That is, a CANCEL statement issued for a subprogram that has been dynamically loaded causes the storage occupied by the subprogram to be freed. As a result, a subsequent call to the subprogram functions as if it were the first.

```
CANCEL CALLED-PROGRAM.
```

Note: A program other than the original caller may issue a CANCEL statement referring to a called program.

CLOSE Statement

There are two ways in which to use the CLOSE statement when closing several files:

```
CLOSE DETAIL-FILE MASTER-FILE.

        or

CLOSE DETAIL-FILE.
CLOSE MASTER-FILE.
```

Each CLOSE statement for a file requires the use of a storage area that is directly proportional to the number of files being closed. Closing more than one file with the same statement is faster than when using a separate statement for each file. However, separate statements require less storage.

COMPUTE Statement

The use of the COMPUTE statement generates more efficient coding than does the use of individual arithmetic statements because the compiler can keep track of internal work areas and does not have to store the results of intermediate calculations. It is the user's responsibility, however, to insure that the data is defined with the level of significance required in the answer.

IF Statement

Nested and compound IF statements should be avoided as the logic is difficult to debug.

MOVE Statement

When a MOVE statement with the CORRESPONDING option is executed, data items are considered CORRESPONDING only if their respective data names are the same, including all implied qualification, up to, but not including, the data-names used in the MOVE statement itself.

For example,

```
01  AA                      01  XX
    05  BB                      05  BB
        10  CC                      10  CC
        10  DD                      10  DD
    05  EE                      05  YY
        10  FF                      10  FF
```

The statement MOVE CORRESPONDING AA TO XX will result in moving CC and DD but not FF because FF of EE does not correspond to FF of YY).

Note: The other rules for MOVE CORRESPONDING, of course, must still be satisfied.

NOTE Statement

An asterisk (*) should be used in place of the NOTE statement, because there is the possibility that when NOTE is the first sentence in a paragraph, it will inadvertently cause the whole paragraph to be treated as part of the NOTE.

OPEN Statement

There are two ways in which to use the OPEN statement when opening several files:

```
OPEN INPUT INFILE UPDATES OUTPUT OUTFILE

        or

OPEN INPUT INFILE
OPEN INPUT UPDATES
OPEN OUTPUT OUTFILE
```

Each OPEN statement for a file requires the use of a storage area that is directly proportional to the number of files being opened. Opening more than one file with the same statement is faster than using a separate statement for each file. However, separate statements require less storage.

PERFORM Verb

PERFORM is a useful verb if the programmer adheres to the following rules:

1. Always execute the last statement of a series of routines being operated on by a PERFORM statement. When branching out of the routine, make sure control will eventually return to the last statement of the routine. This statement should be an EXIT statement. Although no code is generated, the EXIT statement allows a programmer to recognize immediately the extent of a series of routines within the range of a PERFORM statement.

2. Always either PERFORM routine-name THRU routine-name-exit, or PERFORM section-name. A PERFORM paragraph-name can cause trouble for the programmer trying to maintain the program. For example, if a paragraph must be broken into two paragraphs, the programmer must examine every statement to determine whether or not this paragraph is within the range of the PERFORM statement. Then all statements referencing the paragraph-name must be changed to PERFORM THRU statements.

READ INTO and WRITE FROM Options

Use READ INTO and WRITE FROM, and do all processing in the Working-Storage Section. This is suggested for three reasons:

1. Debugging is much simpler. Working-Storage areas are easier to locate in a dump than are buffer areas. And, if files are blocked, it is much easier to determine which record in a block was being processed when the abnormal termination occurred.

2. Trying to access a record area after the AT END condition has occurred (for example, AT END MOVE HIGH-VALUE TO INPUT-RECORD) can cause problems if the record area is only in the File Section.

3. WRITE FROM allows multiple writes of the same record (not possible when using the record area because of buffering techniques).

Note: The programmer should be aware that additional time is used to execute the move operation involved in each READ INTO or WRITE FROM instruction.

WRITE ADVANCING with LINAGE, FOOTING, and END-OF-PAGE

The features LINAGE, WITH FOOTING, and END-OF-PAGE imperative-statement give the programmer added flexibility and control in physical sequential (QSAM) output operations. When these features are used in combination with the BEFORE/AFTER ADVANCING nn LINES clause of the WRITE statement, however, care must be exercised. In the discussion below, notice that END-OF-PAGE imperatives are executed after WRITEs, and the LINAGE-COUNTER may be pointing to the next logical page (instead of to the current footing area) when the imperative gains control.

For ADVANCING nn LINES, COBOL first calculates the sum of LINAGE-COUNTER and nn. (For ADVANCING PAGE, see Case 2 below.) Subsequent actions depend on the size of this value, as follows:

Case 1--If advance would be within the current logical page body (i.e., value is not greater than the established LINAGE value):

a. The WRITE takes place (either before or after advancing nn lines, as specified in the program).

b. LINAGE-COUNTER is incremented by nn.

c. If FOOTING was speciifed, and the advance falls within the footing area (that is, greater than or equal to the established FOOTING value), the END-OF-PAGE imperative is executed (if one was specified).

Case 2--If advance would go beyond the current logical page body (i.e., value is greater than the established LINAGE value):

a. A new value is established for LINES-AT-TOP.

b. The WRITE takes place before or after (as specified by the program) the device is positioned to the first line of the next logical page.

c. LINAGE-COUNTER is set to 1.

d. New values are established fc LINAGE, FOOTING, and LINES-AT-BOTTOM.

e. The END-OF-PAGE imperative is executed (if one was specified).

Note: Files using LINAGE are treated as if the ADV compile option had been specified.

WRITE BEFORE/AFTER ADVANCING

When the ADVANCING option is used for the WRITE statement, library subroutine ILBOSPA is called to handle the spacing. The carriage control characters are determined as follows:

- When only AFTER is specified for the file, ASA control characters are used.
- When only BEFORE is specified for the file, machine control characters are used.
- When both BEFORE and AFTER are specified for output operations, machine control characters are used.

RECEIVE Statement

The RECEIVE statement makes available to the COBOL program a message, a message segment, or part of a message or message segment, as well as information about that message from a queue maintained by the message control program (MCP). The following example of the RECEIVE statement is taken from the sample COBOL communication program shown in Figure 160:

```
RECEIVE CDNAME-IN MESSAGE INTO IDENT-REC.
```

SEND Statement

Specification of the SEND statement in the COBOL program causes a message, a message segment, or part of a message or message segment to be released to the message control program (MCP). The following example of the SEND statement is taken from the sample COBOL communication program shown in Figure 147:

```
SEND CDNAME-OUT FROM IDENT-SEND WITH EMI.
```

Notes:

- Although the COBOL program has access to a message only when the MCP has received it in entirety and placed it in a queue, once several messages have met this requirement the COBOL program can process messages from different MCP queues at the same time.

- If one execution of a RECEIVE statement (or a SEND statement) transmits only part of a message, subsequent executions of RECEIVE statements (or SEND statements) in that run unit are required for transmission of the rest of the message.
- The MCP does not transmit data until the COBOL program has sent it a complete message. This complete message is built in a buffer in the region in which the COBOL program is executing.

ENABLE/DISABLE Statements

These two verbs are used in a COBOL communication program to allow/inhibit data transfer. For output, this transfer is between specified output queues and output destinations; for input, transfer is between specified sources and input queues COBOL provides an interface between the program and the TCAM message control program (MCP), where enabling/disabling actually occurs. For more detail, see the section "Using the Communication Feature."

START Statement

For a sequentially-accessed ISAM file, the START statement must be executed before the READ statement for a given record if either of the following is true:

- Processing begins with other than the first record;
- Processing continues with a record other than the next sequential record.

There are two ways to use the START statement to begin processing a segment of a sequentially accessed ISAM file at a specified key. The programmer may indicate either Method 1, to begin at a specific NOMINAL KEY that matches a RECORD KEY within the file, or Method 2, to start within the first record in a specific generic key class.

Method 1:

```
START file-name
     [INVALID KEY imperative-statement]
```

Method 2:

```
START file-name USING KEY data-name
     {EQUAL TO}
     {   =    } identifier
      [INVALID KEY imperative-statement]
```

where data-name is the data-name given in the RECORD KEY clause and identifier contains the generic key value for the request and may be any data item whose length is less than or equal to that of the RECORD KEY.

Note: For ISAM a problem may result with the generic key facility with binary key if the low-order byte of the search argument is binary zero.

STRING Statement

The STRING statement combines two or more subfields into a single field. When this statement is executed, characters from the sending item(s) are transferred to the receiving item in the same way that moves from alphanumeric to alphanumeric item(s) are effected. The example in Figure 110 illustrates the use of the STRING statement options available to the user. For a discussion of the formats possible with the STRING statement, see the publication IBM VS COBOL for OS/VS.

TRANSFORM Statement

The TRANSFORM statement generates more efficient code than the EXAMINE statement with the REPLACING BY option when only one character is being transformed. TRANSFORM, however, uses a 256-byte table.

```
    STRING SNDFLD5 DELIMITED BY DLMTR
           SNDFLD6 DELIMITED BY SIZE

  *  Combine data in SNDFLD5 up to the delimiter indicated by DLMTR with all the data
  *  in another sending field (as indicated by the SIZE option of the STRING
  *  statement).

           INTO RCDFLD1 WITH POINTER POINTR

  *  Place the result in RCDFLD1 beginning at the relative location designated
  *  by POINTR.

           ON OVERFLOW GO TO OVERFLOW2.

  *  If RCDFLD1 is not large enough to accommodate the combined data-fields, or
  *  if the original contents of the pointer field were less than 1, execute a user-
  *  written checking routine called OVERFLOW2.
```

Figure 110. Using the STRING Statement

```
    UNSTRING SNDFLD

  *  Separate the data in the sending area.

             DELIMITED BY DLMTR1
                OR SPACES
                OR ALL 'E'
             INTO RCFLD

  *  When the character, or set of characters, marking the end of a section of the
  *  sending area is found, move the isolated data into the data-receiving field.

             DELIMITER IN DELIM-IN

  *  Move the delimiter found into the delimiter-receiving area DELIM-IN.

             COUNT IN COUNT-IN

  *  Specify in COUNT-IN the number of characters placed in the RCFLD
  *  data-receiving field.

             WITH POINTER POUNTR

  *  Indicate the relative position in the SNDFLD sending area of the first
  *  character to be examined.  At the end of the operation, POINTR contains a value
  *  equal to the initial value plus the number of characters examined in the sending
  *  field.

             TALLYING IN TALLY-IN

  *  Record the number of data-receiving areas acted upon.  At the end of the
  *  operation, TALLY-IN will contain a value equal to the initial value plus the
  *  number of receiving areas acted upon.

             ON OVERFLOW
                DISPLAY 'OVERFLOW CONDITION'
                GO TO CHECK-ROUTINE.

  *  If the data-receiving fields cannot accommodate the data being sent, or if
  *  the original value of the pointer was less than 1 or greater than the size of the
  *  sending field, execute a user-written checking routine.
```

Figure 111. Using the UNSTRING Statement

UNSTRING Statement

The UNSTRING statement separates contiguous data in a sending field, placing it in multiple receiving fields. The example in Figure 111 illustrates the use of the UNSTRING statement options available to the user.

For a discussion of the formats possible with the UNSTRING statement, see the publication IBM VS COBOL for OS/VS.

USING THE REPORT WRITER FEATURE

REPORT Clause in FD

A given report-name may appear in a maximum of two file description entries. The file description entries need not have the same characteristics. If the same report-name is specified in two file description entries, the report will be written on both files. For example:

```
ENVIRONMENT DIVISION.
    SELECT FILE-1  ASSIGN UR-1403-S-PRTOUT.
    SELECT FILE-2  ASSIGN UT-2400-S-SYSUT1.
    .
    .
    .

DATA DIVISION.
FD  FILE-1  RECORDING MODE F
            RECORD CONTAINS 121 CHARACTERS
```

```
            REPORT IS REPORT-A.
FD  FILE-2  RECORDING MODE V
            RECORD CONTAINS 101 CHARACTERS
            REPORT IS REPORT-A.
```

For each GENERATE statement, the records for REPORT-A will be written on FILE-1 and FILE-2, respectively. The records on FILE-2 will not contain columns 102 through 121 of the corresponding records on FILE-1.

Summing Technique

The object program can be made more efficient with respect to execution time bykeeping in mind the fact that Report Writer source coding is treated as though the programmer had written the program in COBOL without the Report Writer feature. Therefore, a complex source statement or series of statements will generally be executed faster than simple statements that perform the same function. The example below shows two coding techniques for the Report Section of the Data Division. Method 2 uses the more complex statements.

```
RD...CONTROLS ARE YEAR MONTH WEEK DAYY
```

Method 1:

```
01  TYPE CONTROL FOOTING YEAR.
    05  SUM COST.
01  TYPE CONTROL FOOTING MONTH.
    05  SUM COST.
01  TYPE CONTROL FOOTING WEEK.
    05  SUM COST.
01  TYPE CONTROL FOOTING DAYY.
    05  SUM COST.
```

Method 2:

```
01  TYPE CONTROL FOOTING YEAR.
    05  SUM A.
01  TYPE CONTROL FOOTING MONTH.
    05  A SUM B.
01  TYPE CONTROL FOOTING WEEK.
    05  B SUM C.
01  TYPE CONTROL FOOTING DAYY.
    05  C SUM COST.
```

Method 2 will execute faster. One addition will be performed for each day, one more for each week, and one for each month. In Method 1, four additions will be performed for each day.

Use of SUM

Unless each identifier is the name of a SUM counter in a TYPE CONTROL FOOTING report group at an equal or lower position in the control hierarchy, the identifier must be defined in the File, Working-Storage or Linkage Sections, as well as in a TYPE DETAIL report group as a SOURCE item. A SUM counter is algebraically incremented just before presentation of the TYPE DETAIL report group in which the item being summed appears as a source item or the item being summed appeared in a SUM clause that contained an UPON option for this DETAIL report group. This is known as SOURCE-SUM correlation. In the following example, SUBTOTAL is incremented only when DETAIL-1 is generated:

```
FILE SECTION.
        .
        .
        .
    05  NO-PURCHASES       PICTURE 99.
        .
        .
        .
REPORT SECTION.
01  DETAIL-1 TYPE DETAIL.
    05  COLUMN 30          PICTURE 99 SOURCE
        NO-PURCHASES.
        .
        .
        .
01  DETAIL-2 TYPE DETAIL.
        .
        .
        .
01  DAY TYPE CONTROL FOOTING
    LINE PLUS 2.
        .
        .
        .
    05  SUBTOTAL COLUMN 30 PICTURE 999
        SUM NO-PURCHASES.
        .
        .
        .
01  MONTH TYPE CONTROL FOOTING
    LINE PLUS 2 NEXT GROUP
    NEXT PAGE.
```

SUM Routines

A SUM routine is generated by the Report Writer for each DETAIL report group of the report. The operands included for summing are determined as follows:

1. The SUM operand(s) also appears in a SOURCE clause(s) for the DETAIL report group.

2. The UPON detail-name option was specified in the SUM clause. In this case, all the operands are included in the SUM routine for only that DETAIL report group, even if the operand appears in a SOURCE clause in other DETAIL report groups.

When a GENERATE detail-name statement is executed, the SUM routine for that DETAIL report group is executed in its logical sequence. When a GENERATE report-name statement is executed and the report contains more than one DETAIL report group, the SUM routine is executed for each one. The SUM routines are executed in the sequence in which the DETAIL report groups are specified.

The following examples show the SUM routines that are generated by the Report Writer. Example 1 illustrates how operands are selected for inclusion in the routine on the basis of simple SOURCE-SUM correlation. Example 2 illustrates how operands are selected when the UPON detail-name option is specified.

EXAMPLE 1: The following statements are coded in the Report Section:

```
01  DETAIL-1 TYPE DE...
    05 ...  SOURCE A.
              .
              .
              .
01  DETAIL-2 TYPE DE...
    05 ...  SOURCE B.
    05 ...  SOURCE C.
              .
              .
              .
01  DETAIL-3 TYPE DE...
    05 ...  SOURCE B.
              .
              .
              .
01  TYPE CF...
    05 SUM-CTR-1...SUM A, B, C.
              .
              .
              .
01  TYPE CF...
    05 SUM-CTR-2...SUM B.
```

One SUM routine is generated for each DETAIL report group, as follows:

SUM Routine for DETAIL-1

```
REPORT-SAVE
   ADD A TO SUM-CTR-1.
REPORT-RETURN
```

SUM Routine for DETAIL-2

```
REPORT-SAVE
   ADD B TO SUM-CTR-1.
   ADD C TO SUM-CTR-1.
   ADD B TO SUM-CTR-2.
REPORT-RETURN
```

SUM Routine for DETAIL-3

```
REPORT-SAVE
   ADD B TO SUM-CTR-1.
   ADD B TO SUM-CTR-2.
REPORT-RETURN
```

EXAMPLE 2: In this example, the same coding is used as in Example 1, with one exception: the UPON detail-name option is used for SUM-CTR-1, as follows:

```
01  TYPE CF...
    05  SUM-CTR-1...SUM A, B, C UPON
        DETAIL-2.
```

The following SUM routines would then be generated instead of those resulting from the calculations in Example 1.

SUM Routine for DETAIL-1

```
REPORT-SAVE
REPORT-RETURN
```

SUM Routine for DETAIL-2

```
REPORT-SAVE
   ADD A TO SUM-CTR-1.
   ADD B TO SUM-CTR-1.
   ADD C TO SUM-CTR-1.
   ADD B TO SUM-CTR-2.
REPORT-RETURN
```

SUM Routine for DETAIL-3

```
REPORT-SAVE
   ADD B TO SUM-CTR-2.
REPORT-RETURN
```

Output Line Overlay

The Report Writer output line is put together with an internal REDEFINES specification, indexed by integer-1. No check is made to prevent overlay on any line. For example:

```
    05  COLUMN 10           PICTURE X(23)
        VALUE "MONTHLY SUPPLIES REPORT".
    05  COLUMN 12           PICTURE X(9)
        SOURCE CURRENT-MONTH.
```

the length of 23 in column 10, followed by a specification for column 12 will cause field overlay.

Page Breaks

The Report Writer page break routine operates independently of the routines that are executed after any control breaks (except that a page break will occur as the result of a LINE NEXT PAGE clause). Thus, the programmer should be aware of the following facts:

1. A Control Heading is not printed after a Page Heading except for first generation. If the programmer wishes to have the equivalent of a Control Heading at the top of each page, he must include the information and data to be printed as part of the Page Heading. But since only one Page Heading may be specified for each report, he should be selective in considering his Control Heading because this "Control Heading" will be the same for each page, and may be printed at inopportune times (see "Control Footings and Page Format," in this chapter.)

2. GROUP INDICATE items are printed after page and control breaks. Figure 112 contains a GROUP INDICATE clause and shows the execution output.

```
REPORT SECTION.
  .
  .
  .
01  DETAIL-LINE TYPE IS DETAIL LINE
    NUMBER IS PLUS 1.
    05  COLUMN IS 2 GROUP INDICATE
        PICTURE IS A(9) SOURCE IS
        MONTHNAME OF RECORD-AREA (MONTH).
  .
  .
  .
          (Execution Output)
  .
  .
  .

JANUARY    15    A00...
                 A02...

PURCHASES AND COST...

JANUARY    21    A03...
                 A03...
```

Figure 112. Sample Showing GROUP INDICATE Clause and Resultant Execution Output

WITH CODE Clause

When more than one report is being written on a file and the reports are to be selectively written, a unique 1-character code must be given for each report. A mnemonic-name is specified in the RD-level entry for each report and is associated with the code in the Special-Names paragraph of the Environment Division.

Note: If a report is written with the CODE option, the report should not be written directly to a printer device.

This code will be written as the first character of each record that is written on the file. When the programmer wishes to write a report from this file, he needs merely to read a record, check the first character for the desired code, and have it printed if this code is found. The record should be printed starting from the third character, as illustrated in Figure 113. should be printed starting from the second character for ADV, third character for NOADV, as illustrated in Figure 113.

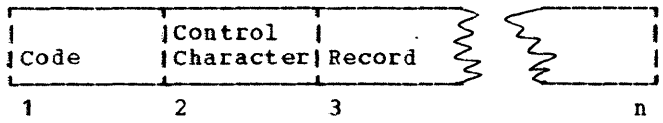


Figure 113. Format of a Report Record When the CODE Clause is Specified

The following example shows how to create and print a report with a code of A. A Report Writer program contains the following statements:

```
ENVIRONMENT DIVISION.
.
.
.
SPECIAL-NAMES.   'A' IS CHR-A
                 'B' IS CHR-B.
.
.
.
DATA DIVISION.
FILE SECTION.
FD  RPT-OUT-FILE
    RECORDS CONTAIN 122 CHARACTERS
    LABEL RECORDS ARE STANDARD
    REPORTS ARE REP-FILE-A REP-FILE-B.
    .
    .
    .
REPORT SECTION.
RD  REP-FILE-A   CODE CHR-A...
.
.
.
RD  REP-FILE-B  CODE CHR-B...
.
.
.
```

The RPT-OUT-FILE must be written on a tape or mass storage device. A second program could then be used to print only the report with the code of A, as follows:

```
DATA DIVISION.
FD  RPT-IN-FILE
    RECORD CONTAINS 122 CHARACTERS
    LABEL RECORDS ARE STANDARD
    DATA RECORD IS RPT-RCD.
01  RPT-RCD.
    05  CODE-CHR         PICTURE X.
    05  PRINT-PART.
        10  CTL-CHR      PICTURE X.
        10  RECORD-PART PICTURE X(120).
FD  PRINT-FILE
    RECORD CONTAINS 121 CHARACTERS
    LABEL RECORDS ARE STANDARD
    DATA RECORD IS PRINT-REC.
01  PRINT-REC.
    05  FILLER           PICTURE X(121).
.
.
PROCEDURE DIVISION.
.
.
LOOP.  READ RPT-IN-FILE AT END
         GO TO CONTINUE.
         IF CODE-CHR  = "A"
         WRITE PRINT-REC FROM
           PRINT-PART
         AFTER POSITIONING CTL-CHR
           LINES.
         GO TO LOOP.
CONTINUE.
         .
         .
```

Control Footings and Page Format

Depending on the number and size of Control Footings (as well as the page depth of the report), all of the specified Control Footings may not be printed on the same page if a control break occurs for a high-level control. When a page condition is detected before all required Control Footings are printed, the Report Writer will print the Page Footing (if specified), skip to the next page, print the Page Heading (if specified), and then continue to print Control Footings.

If the programmer wishes all of his Control Footings to be printed on the same page, he must format his page in the RD-level entry for the report (by setting the LAST DETAIL integer to a sufficiently low line number) to allow for the necessary space.

```
RD  EXPENSE-REPORT CONTROLS ARE FINAL,
    MONTH, DAYY
      .
      .
      .
01  TYPE CONTROL FOOTING DAYY
    LINE PLUS 1 NEXT GROUP
    NEXT PAGE.
      .
      .
      .
01  TYPE CONTROL FOOTING MONTH
    LINE PLUS 1 NEXT GROUP
    NEXT PAGE.
      .
      .
      .
(Execution Output)

EXPENSE REPORT
      .
      .
      .
January 31.....29.30
    (Output for CF DAYY)

January total.....131.40
    (Output for CF MONTH)
```

Note: The NEXT GROUP NEXT PAGE clause for the control footing DAY is not activated.

Floating First Detail Rule

The first presentation of a body group (CH, CF, or DE) that contains a relative line as its first line, will have its relative line spacing suppressed, and the first line will be printed on either the value of FIRST DETAIL or INTEGER PLUS 1 of a NEXT GROUP clause from the preceding page. For example:

A. If the following body group was the last to be printed on a page

```
01  TYPE CF NEXT GROUP NEXT PAGE
```

Then this next body group

```
01  TYPE DE LINE PLUS 5
```

would be printed on value of FIRST DETAIL (in PAGE clause).

B. If the following body group was the last to be printed on a page

```
01  TYPE CF NEXT GROUP LINE 12
```

and after printing, line-counter = 40, then this next BODY GROUP

```
01  TYPE DETAIL LINE PLUS 5
```

would be printed on line 12 + 1 (i.e., line 13).

Report Writer Routines

At the end of the analysis of a report description entry (RD), the Report Writer routines are generated, based on the contents of the RD. Each routine refers to the compiler-generated card number of its own respective RD.

TABLE HANDLING CONSIDERATIONS

Subscripts

If a subscript is represented by a constant and if the subscripted item has a fixed length, the location of the subscripted data item within the table or list is resolved at compile time.

If a subscript is represented by a data-name, the location is resolved at execution time. The most efficient format, in this case, is COMPUTATIONAL, with PICTURE size less than five integers.

The value contained in a subscript is an integer that represents an occurrence number within a table. Every time a subscripted data-name is referred to in a program, the compiler generates up to 16 instructions to calculate the correct displacement. Therefore, if a subscripted data-name is to be processed extensively, move the subscripted item to an unsubscripted work area, do all necessary processing, and then move the item back into the table. Even when subscripts are described as computational, subscripting takes time and main storage.

There is, however, compiler optimization in the computation of displacements. If a subscripted data item is referred to more than once in the same paragraph, the displacement for the data item is computed only once, then saved and used again for the subsequent references. For example:

```
MOVE ITEM (A, B) TO D.
MOVE ITEM (A, B,) TO E.
```

If these two statements occur in the same paragraph, and no statements between them

could change the values stored in A or B, then the location of ITEM (A, B) will be computed only once and saved, and used again for the second reference.

Index-Names

Index-names are compiler-generated items, one fullword in length, assigned storage in the TGT. An index-name is defined by the INDEXED BY clause. The value in an index-name represents an actual displacement from the beginning of the table that corresponds to an occurrence number in the table. Address calculation for a direct index takes a maximum of four instructions; address calculation for a relative index takes a few more. Therefore, the use of index-names in referencing tables is more efficient than the use of subscripts. The use of direct indexes is faster than the use of relative indexes.

Index-names can only be referenced in the PERFORM, the SEARCH, and the SET statements.

Index Data Items

Index data items are compiler-generated storage positions, one fullword in length, that are assigned storage within the COBOL program area. An index data item is defined by the USAGE IS INDEX clause. The programmer can use index data items to save values of index-names for later reference.

Great care must be used when setting values of index data items. Since an index data item is not part of any table, the compiler places the value contained in the index-name or other index data item into the index data item (see the example given in "SET Statement"). Index data items can only be referenced in SEARCH and SET statements, a relational condition, or the USING phrase of a PROCEDURE DIVISION CALL or ENTRY statement.

OCCURS Clause

A table element is represented by the subject of an OCCURS clause, and is equivalent to one level of a table. If indexing is to be used to reference a table element, and the Format 2 (SEARCH ALL) statement is also to be used, the KEY option must be specified in the OCCURS clause. The table element must then be ordered upon the key(s) data-name(s) specified.

DEPENDING ON Option

If a data item described by an OCCURS clause with the DEPENDING ON data-name option[1] is followed by nonsubordinate data items, a change in the value of data-name during the course of program execution will have the following effects:

1. The size of any group described by or containing the related OCCURS clause will reflect the new value of data-name.
2. Whenever a MOVE to a field containing an OCCURS clause with the DEPENDING ON option is executed, the MOVE is made on the basis of the current contents of the object of the DEPENDING ON option.
3. The location of any nonsubordinate items following the item described with the OCCURS clause will be affected by the new value of data-name. If the user wishes to preserve the contents of these items, the following procedure can be used: prior to the change in data-name, move all nonsubordinate items following the variable item to a work area; after the change in data-name, move all the items back.

Note: The value of data-name may change because a move is made to it or to the group in which it is contained; or the value of data-name may change because the group in which it is contained is a record area that has been changed by execution of a READ statement.

For example, assume that the Data Division of a program contains the following coding:

```
01  ANYRECORD.
    05  A PICTURE S999 COMPUTATIONAL-3.
    05  TABLEA PICTURE S999 OCCURS 1 TO
        100 TIMES DEPENDING ON A.
    05  GROUPB.

        (Subordinate data items.)

           (End of record.)
```

---

[1]For a discussion of the use of the OCCURS DEPENDING ON clause in a sort program, see "Sorting Variable-Length Records."

GROUPB items are not subordinate to TABLEA, which is described by the OCCURS clause. Assuming that WORKB is a work area with the same data structure as GROUPB, the following procedural coding could be used:

1. MOVE GROUPB TO WORKB
2. Calculate new value of A
3. MOVE WORKB TO GROUPB

The above statements can be avoided by putting the OCCURS clause with the DEPENDING ON option at the end of the record.

Note: Data-name can also change because of a change in the value of an item that redefines it. In this case, the group size and the location of nonsubordinate items as described in the two preceding paragraphs cannot be determined.

SET Statement

The SET statement is used to assign values to index-names and to index data items.

When the SET statement assigns to an index-name the value of a literal, identifier, or an index-name from another table element, it is set to an actual displacement from the beginning of the table element that corresponds to the occurrence number indicated by the second operand in the SET statement. The compiler performs all the necessary calculations. If the SET statement is used to assign an index-name to another index-name for the same table element, the compiler need make no conversion of the actual displacement value contained in the second operand.

However, when an index data item is set to another index data item or to an index-name, or when an index-name is set to an index data item, the compiler is unable to change any displacement value it finds, since an index data item is not part of any table. Thus, no conversion of values can take place. If the programmer forgets this, programming errors can occur.

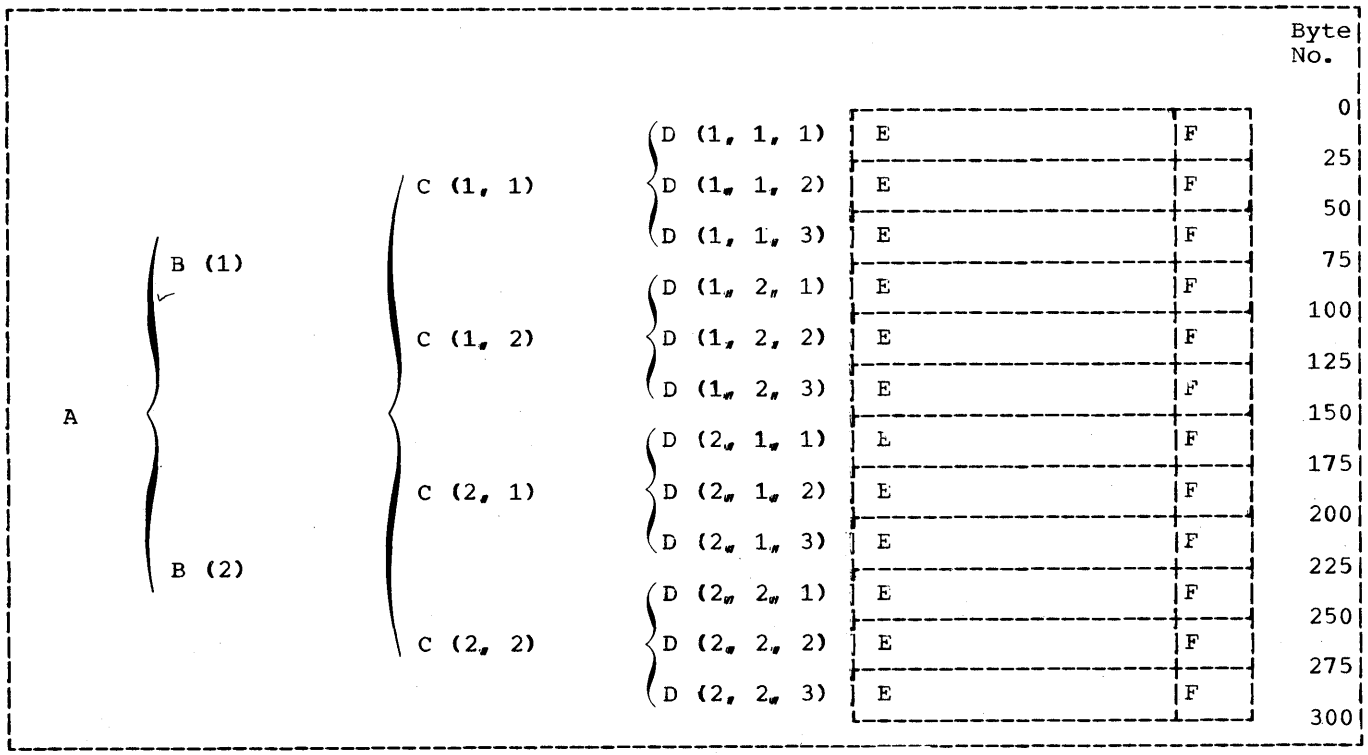


Figure 114. Storage Layout for Table Reference Example

For example, suppose that a table has been defined as:

```
01 A.
   02 B OCCURS 2 INDEXED BY I1, I5.
      03 C OCCURS 2 INDEXED BY I2, I6.
         04 D OCCURS 3 INDEXED BY I3, I4.
            05 E PIC X(20).
            05 F PIC 9(5).
```

Figure 114 shows how the table is laid out in main storage.

Now, suppose it is necessary to reference D (2,2,3). The following steps are incorrect:

```
SET I3 TO 2.
SET INDX-DATA-ITM TO I3.
SET I2, I1 TO INDX-DATA-ITM.
SET I3 UP BY 1.
MOVE D(I1, I2, I3) TO WORKAREA.
```

The value contained in I3 after the first SET statement is 25, which represents the beginning point (in bytes) of the second occurrence of D. When the second SET statement is executed, the value 25 is placed in INDX-DATA-ITM, and the third SET statement moves the value 25 into I2 and I1. The fourth SET statement increases the value in I3 to 50. The calculation for the address D (I1, I2, I3) would then be as follows:

```
(address of D(1,1,1)) + 25 + 25 + 50 =
  (address of D(1,1,1)) + 100
```

where D(1,1,1) represents the first occurrence of D. This is not the address of D (2,2,3).

The following steps will find the correct address:

```
SET I3 TO 2.
SET I2, I1 TO I3.
SET I3 UP BY 1.
```

In this case, the first SET statement places the value 25 in I3. Since the compiler is able to calculate the lengths of B and C, the second SET statement places the value 75 in I2, and the value 150 in I1. The third SET statement places the value 50 in I3. The correct address calculation will be:

```
(address of D(1,1,1)) + 150 + 75 + 50 =
   (address of D(1,1,1)) + 275.
```

The rules for the SET statement are shown in Figure 115.

SEARCH Statement

Only one level of a table (a table element) can be referenced with one SEARCH statement. Note that SEARCH statements cannot be nested, since an imperative-statement must follow the WHEN condition, and the SEARCH statement is itself conditional.

There are two formats for the SEARCH statement.

Format 1 SEARCH statements perform a serial search of a table element. If the programmer knows that the "found" condition will come after some intermediate point in the table element, to speed up execution, he can use the SET statement to set the index-names at that point and search only part of the table element. If the table element is large, and must be searched from the first occurrence to the last, the use of Format 2 (SEARCH ALL) is more efficient

Receiving \ Sending	Index-name	Index Data Item	Identifier or Literal
Index-name	Set to value corresponding to occurrence number[1]	Move without conversion	Set to value corresponding to occurrence number
Index Data Item	Move without conversion	Move without conversion	--
Identifier	Set to occurrence number represented by index-name	--	--

[1]If index-name refer to the same table element move without conversion

Figure 115. Rules for the SET Statement

than Format 1, since it uses a binary search technique; however, the table must then be ordered.

In Format 1, the VARYING option allows the programmer to:

- Vary an index-name other than the first index-name stated for this table element. Thus, with two SEARCH statements each using a different index-name, reference can be made to more than one value in the same table element for comparisons, etc.

- Vary an index-name from another table element. In this case, the first index-name specified for this table element is used for the search, and the index-name specified in the VARYING option is incremented at the same time. Thus, it is possible to step through two table elements at once.

In Format 1, the WHEN condition can be any relation condition, and can be multiple. If multiple WHEN conditions are stated, the implied logical connective is OR -- that is, if any one of the WHEN conditions is satisfied, the imperative-statement following the WHEN condition is executed. If all conditions of the SEARCH statement are to be satisfied before exiting from the search, a compound WHEN condition with an AND logical connective must be written.

In Format 2, the SEARCH ALL statement, the table must be ordered on the KEY(S) specified in the OCCURS clause. Any KEY may be specified in the WHEN condition, but all preceding data-names in the KEY option must also be tested. The test must be an "equal to" (=) condition, and the KEY data-name must be either the subject of the condition or the name of a conditional variable with which the tested condition-name is associated. The WHEN condition can also be a compound condition, formed from one of the simple conditions listed above, with AND as the only logical connective. The KEY and its object of comparison must be compatible, as given in the rules of the relation test.

To write a series of statements that will search the three-dimensional table discussed in the section "The SET Statement," the programmer could write:

```
77 COMPARAND1 PIC X(5).
77 COMPARAND2 PIC 9(5).
01 A.
   05 B OCCURS 2 INDEXED BY I1 I5.
      10 C OCCURS 2 INDEXED BY I2 I6.
         15 D OCCURS 3 INDEXED BY I3, I4.
            20 E PIC X(5).
            20 F PIC 9(5).
                        .
                        .
                        .
    (initialize comparand1 and comparand2)
                        .
                        .
                        .
   PERFORM SEARCH-TEST1 THRU SEARCH-EXIT1
   VARYING I1 FROM 1 BY 1 UNTIL I1 GREATER
   THAN 2 AFTER I2 FROM 1 BY 1 UNTIL I2
   GREATER THAN 2.
ENTRY-NOENTRY1.  GO TO ERROR-RECOVERY1.
                        .
                        .
                        .
SEARCH-TEST1.  SET I3 TO 1.
   SEARCH D WHEN E (I1, I2, I3) =
      COMPARAND1 AND
      F (I1, I2, I3) = COMPARAND2
      SET I5 TO I1
      SET I6 TO I2
      SET I2 TO 3
      SET I1 TO 3
      ALTER ENTRY-NOENTRY1 TO PROCEED TO
      ENTRY-PROCESSING1.
   SEARCH-EXIT1.  EXIT.
                        .
                        .
                        .
   ERROR-RECOVERY1.
                        .
                        .
                        .
   ENTRY-PROCESSING1.
      MOVE E(I5, I6, I3) TO OUT-AREA1.
      MOVE F(I5, I6, I3) TO OUT-AREA2.
                        .
                        .
                        .
```

The PERFORM statement varies the indexed (I1 and I2) associated with table elements B and C; the SEARCH statement varies I3, which is associated with table element D.

The values of I1 and I2 that satisfy the WHEN conditions of the SEARCH statement are saved in I5 and I6. I1 and I2 are then both set to 3 using the SET statement, so that upon return from the SEARCH statement control will fall through the PERFORM statement to the GO TO statement.

Subsequent references to the desired occurrence of table elements E and F make use of the index-names I5 and I6 in which the correct value was saved.

For example, a user-defined table may be the following:

```
01  TABLEA.
    05  ENTRY-IN-TABLEE OCCURS 90 TIMES
        ASCENDING KEY-1, KEY-2
        DESCENDING KEY-3
        INDEXED BY INDEX-1.
        10  PART-1  PICTURE 9(2).
        10  KEY-1   PICTURE 9(5).
        10  PART-2  PICTURE 9(6).
        10  KEY-2   PICTURE 9(4).
        10  PART-3  PICTURE 9(33).
        10  KEY-3   PICTURE 9(5).
```

A search of the entire table can be initiated with the following instruction:

```
SEARCH ALL ENTRY-IN-TABLEE AT
  END GO TO NOENTRY
WHEN KEY-1 (INDEX-1) = VALUE-1 AND
  KEY-2 (INDEX-1) = VALUE-2
  AND KEY-3 (INDEX-1) = VALUE-3
MOVE PART-1 (INDEX-1) TO
  OUTPUT-AREA.
```

The foregoing instructions will execute a search on the given array TABLEA which contains 90 elements of 55 bytes and 3 keys. The primary and secondary keys (KEY-1 and KEY-2) are in ascending order whereas the least significant key (KEY-3) is in descending order. If an entry is found in which three keys are equal to the given values (i.e., VALUE-1, VALUE-2 VALUE-3) PART-1 of that entry will be moved to OUTPUT-AREA. If matching keys are not found in any of the entries in TABLEA, the NOENTRY routine is entered.

If a match is found between a table entry and the given values, the index (INDEX-1) is set to a value corresponding to the relative position within the table of the matching entry. If no match is found, the index remains at the setting it had when execution of the SEARCH ALL statement began.

Compilation is faster if KEY(S) are tested in the SEARCH statement in the same order they appear in the KEY option.

Note that if KEY entries within the table do not contain valid values, then the results of the binary search will be unpredictable.

Building Tables

When reading in data to build an internal table:

1. Check to make sure the data does not exceed the space allocated for the table.
2. If the data must be in sequence, check the sequence.
3. If the data contains the subscript determining its position in the table, check the subscript for a valid range.

When testing for the end of a table, use a named value giving the item count, rather than using a literal. Then, if the table must be expanded, only one value need be changed, instead of all references to a literal.

QUEUE STRUCTURE CONSIDERATIONS

In a COBOL teleprocessing (TP) program, a CD FOR INPUT allows the specification of one through three levels of sub-queues from which data can be received; this allows the COBOL object program, at execution time, to make use of pre-defined queue structures, and to access all or parts of such structures. For TP programs, such queue structures are analogous in function and form to the File Description (FD) entry and its associated 01 record description for file processing programs. If pre-defined queue structures are used, each lowest level sub-queue name in the structure corresponds to a DD name; the associated DD card must specify a TPROCESS entry in the message control program (MCP) terminal table). Figure 116 shows the configuration of one queue structure such that queue A is made up of sub-queues B and C, sub-queue B is made up of sub-queues D and E, and sub-queue D is made up of sub-queues H and I (where sub-queue H contains messages Z1 and X2 and sub-queue I contains messages X3, X4, and X5), and so on.

During program execution, when the user wishes to receive a message from a queue (or sub-queue) he need not place the names of all sub-queues in the input CD; he need specify only the SYMBOLIC QUEUE name, which may be the name of a pre-defined queue structure, or he may specify that name plus one or more sub-queue names -- which allows him to access only that part of the entire structure that is needed. A COBOL object-time subroutine uses the name(s) placed in the input CD to determine which

lowest-level sub-queue(s) and corresponding TCAM queue(s) can be used to fill the request.

In order to do this, the user must have previously defined his queue structures in a form that is acceptable to the COBOL object-time subroutine. A utility program that functions as the Queue Structure Description routine (included in the OS/VS COBOL library) makes this possible. Input to the Queue Structure Description routine consists of a series of statements that define queue structures. The statements are written in a COBOL-like format, similar to an FD entry and its associated record description entry. The Queue Structure Description routine produces as output a partitioned data set with one member for each complete queue structure. The sample listing shown in Figure 117 provides the queue definition statements that correspond to the queue structure. At the right of each statement, in parenthesis, is each FD entry equivalent.

Each logical record in a queue structure description may include only a queue or sub-queue definition; it may not include, for example, the usual COBOL sequence number. (For a detailed description of the possible formats for input to the Queue Structure Description routine, see the Section "Rules for Queue Structure Description" in this chapter.)

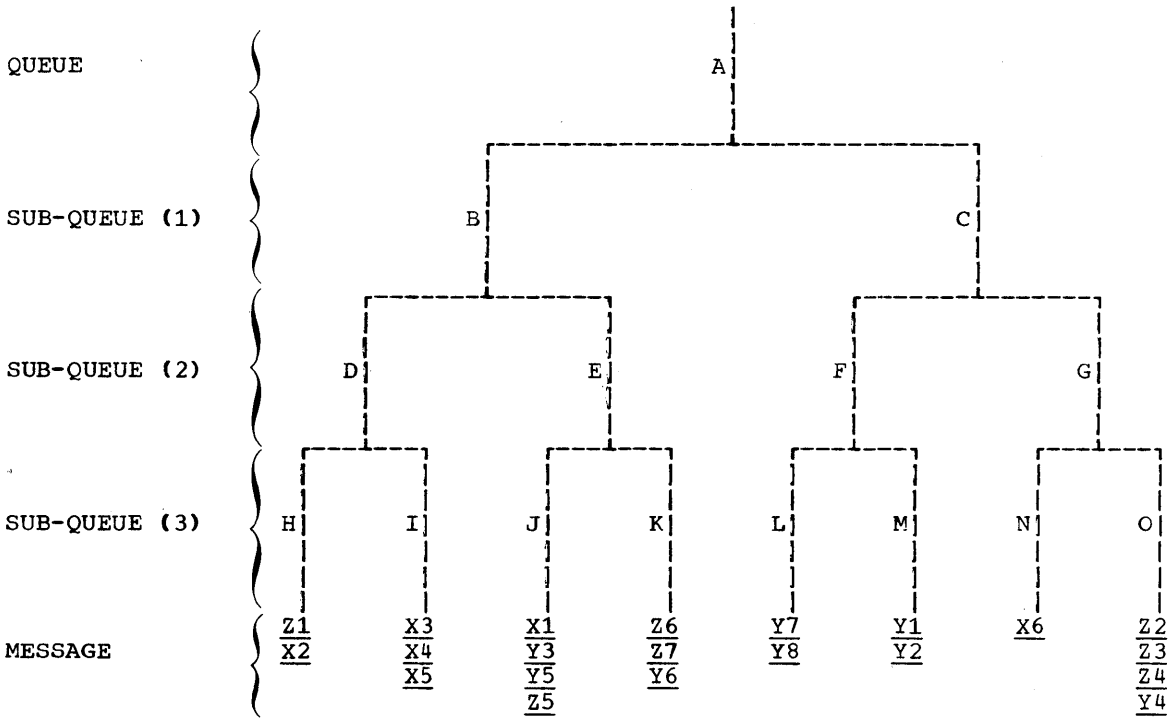


Figure 116. A Queue Structure with Three Levels of Sub-Queues

```
//BLDRDS    JOB  user information
//JOBLIB[1] DD   DSN=SYS1.COBLIB,UNIT=2314,VOL=SER=DC160,DISP=OLD
//SUBQS[2]  EXEC PGM=ILBOQSU0,REGION=96K
//COBTPQD[3] DD  DSN=SUBQPDS,UNIT=2314,VOL=SER=DC160,                    X
//              SPACE=(4000,(50,20,1)),DISP=(OLD,KEEP)
//SYSPRINT[4] DD SYSOUT=A
//SYSIN[5]  DD   *
//
```

QUEUE STRUCTURE DEFINITIONS FOR USE IN COBOL PROGRAMS WHICH PROCESS QUEUES AND SUB-QUEUES.

	Note: The parenthetical entries below are for illustrative purposes only, they may not appear in the program itself.
QUEUE IS A.	(FD clause)
SUB-QUEUE-1 IS B.	(01 entry)
SUB-QUEUE-2 IS D.	(02 entry)
SUB-QUEUE-3 IS H.	(03 entry)
SUB-QUEUE-3 IS I.	(03 entry)
SUB-QUEUE-2 IS E.	(02 entry)
SUB-QUEUE-3 IS J.	(03 entry)
SUB-QUEUE-3 IS K.	(03 entry)
SUB-QUEUE-1 IS C.	(01 entry)
SUB-QUEUE-2 IS F.	(02 entry)
SUB-QUEUE-3 IS L.	(03 entry)
SUB-QUEUE-3 IS M.	(03 entry)
SUB-QUEUE-2 IS G.	(02 entry)
SUB-QUEUE-3 IS N.	(03 entry)
SUB-QUEUE-3 IS O.	03 entry)

Notes:

1. The data-set name SYS1.COBLIB represents the OS/VS COBOL Library.
2. The utility program ILBOQSU0 (called the Queue Structure Description routine) creates a partitioned data set with one member for each complete queue structure defined. It has an alias name of BLDQS, which may be specified on the EXEC card instead.
3. The partitioned data set must be described on a DD card with the reserved name //COBTPQD.
4. The SYSPRINT DD statement defines the output message and listing data set.
5. The SYSIN DD statement defines the input to the program. The SYSIN data set must consist of 80-character records.

Figure 117. A Sample Queue Structure Descripion

ACCESSING QUEUE STRUCTURES THROUGH COBOL

Once the user has defined and stored the queue structures, COBOL TP programs can utilize these structures. At execution time, the partitioned data set is described on a DD card with the name COBTPQD. If, for example, the user wanted to access messages described in the queue structure defined in Figure 117, a DD card specifying the partitioned data set SUBQPDS, as below, would be required.

```
//COBTPQD DD DSN=SUBQPDS,DISP=SHR,
          UNIT=3350,VOL=SER=DC160
```

Additional DD cards would be required to link the message control program terminal table entries and the lowest-level sub-queue names. (For a description of terminal table entries, see the section "Terminal and Line Control Areas" in the chapter "Using the Teleprocessing Feature".) The name of the DD card may be defined either as the sub-queue name itself (for example, as H, I, J, K, L, M, N, or O) or as a ddname that is equivalent to the lowest-level sub-queue name. This alternative approach permits the COBOL program to reuse SYMBOLIC SUB-QUEUE names without ambiguity. These two approaches are illustrated below.

Method 1: The DD card associated with the queue definition SUB-QUEUE-3 IS H would be:

```
//H DD QNAME=Q1
```

Method 2: The DD card associated with the queue definition SUB-QUEUE-3 IS H(FIRSTMSG) would be:

```
//FIRSTMSG DD QNAME=Q1
```

where Q1 is an entry in the terminal table.

Before a RECEIVE statement is executed, the user places (via a MOVE statement) the needed queue and sub-queue name(s) in the CD entry. When the RECEIVE statement is executed, the RECEIVE subroutine checks for the presence of the partitioned data set describing these queue structures. If the data set is present, the RECEIVE subroutine invokes a Queue Analyzer routine, which searches the partitioned data set for a member corresponding to the name in the SYMBOLIC QUEUE field, reads that member into main storage, and uses it to validate the SYMBOLIC SUB-QUEUE name(s) in the COBOL program input CD entry. The Queue Analyzer routine then determines the first valid name for the structure specified and gives this name to the RECEIVE routine.

Names at the SUB-QUEUE-1 level take priority over names at the SUB-QUEUE-2 level. Names at the SUB-QUEUE-2 level take priority over names at the SUB-QUEUE-3 level. At any given level, names at the left take priority over, and are completely evaluated before, names at the right. (Taking advantage of this retrieval technique, the user can improve object-time performance by defining the most frequently used sub-queues at the left of the structure.) Figure 118 illustrates this process.

The RECEIVE subroutine then attempts to access the queue specified. If there are no messages in the associated MCP queue, the Queue Analyzer provides the RECEIVE routine with another valid name. The procedure is repeated until the RECEIVE routine accesses a message, or until there are no more queues to access.

During a RECEIVE operation, a COBOL program using queue structures need not specify all levels of sub-queues. The highest level (QUEUE) must be specified; that level plus a SUB-QUEUE-1 may also be specified; or all four levels may be specified. If a lower level is specified, then all higher levels must also be specified.

If the COBOL programmer wishes to access the next message in the queue structure, regardless of which sub-queue that message may be in, he specifies the queue name only, and initializes the sub-queue-names to SPACES. The Queue Analyzer, when supplying the message, returns to the COBOL object program any applicable sub-queue names via the data items in the associated input CD. In this way, if the entire message was not returned as a result of the current RECEIVE, additional RECEIVE requests may be issued using the complete queue and subqueue names in the CD to retrieve the balance of the message. If, however, the programmer wishes the next message in a given sub-queue, he must specify both the queue name and any applicable sub-queue names. Figure 118 illustrates the relationship between the information contained in the input CD at object time and the message(s) accessed when the RECEIVE statement is executed (where each example refers to the queue structure pictured in Figure 116).

Input CD	Message Returned by the MCP
CD CDNAME-IN FOR INPUT SYMBOLIC QUEUE IS data-name-1. (where data-name-1 contains 'A')	Message Z1
CD CDNAME-IN FOR INPUT SYMBOLIC QUEUE IS data-name-1. SYMBOLIC SUB-QUEUE-1 IS data-name-2. (where data-name-1 contains 'A' and data-name-2 contains 'C')	Message Y7
CD CDNAME-IN FOR INPUT SYMBOLIC-QUEUE IS data-name-1, SYMBOLIC-QUEUE-1 IS data-name-2, SYMBOLIC-QUEUE-2 IS data-name-3. (where data-name-1 contains 'A', data-name-2 contains 'B', and data-name-3 contains 'E')	Message X1

Note: Data-name-1, data-name-2, and so on, refer to the optional clauses of a queue structure defined under "Rules for Queue Structure Description" in this chapter.

Figure 118. Sample Message Retrieval Options

Specifying ddnames with Elementary Sub-Queues

Suppose that an application program is written to accept TP messages as input to an inventory control process. Each of five different locations transmits data on four different parts. The diagram in Figure 119 illustrates the relationship between the input messages and the four different parts for each location.

Each elementary, or lowest-level, queue in the structure must specify the name of a DD card, which in turn names a TPROCESS entry. While the example, as shown in Figure 107, is not ambiguous (that is, INVENTORY.CHICAGO.PARTA is distinct from INVENTORY.LOS-ANGELES.PARTA), the elementary queues by themselves are not (that is, the elementary name PARTA, which corresponds to a ddname, can be any one of five different PARTA's). To eliminate this ambiguity, whenever there are duplicate names in the lowest level of a queue structure, the user must define ddnames in addition to the sub-queue names at the lowest level when he defines the structure to the Queue Structure Description routine. Then the object-time Queue Analyzer routine automatically associates the fully qualified queue structure names with the DD names required. Accordingly, in this example:

NEW-YORK.PARTA could have ddname DD1.

NEW-YORK.PARTB could have ddname DD2.

NEW-YORK.PARTC could have ddname DD3.

NEW-YORK.PARTD could have ddname DD4.

CHICAGO.PARTA could have ddname DD5.

CHICAGO.PARTB could have ddname DD6.

and so forth. In this way, each elementary queue has a unique designation; yet the COBOL program can refer to the sub-queue names without ambiguity.

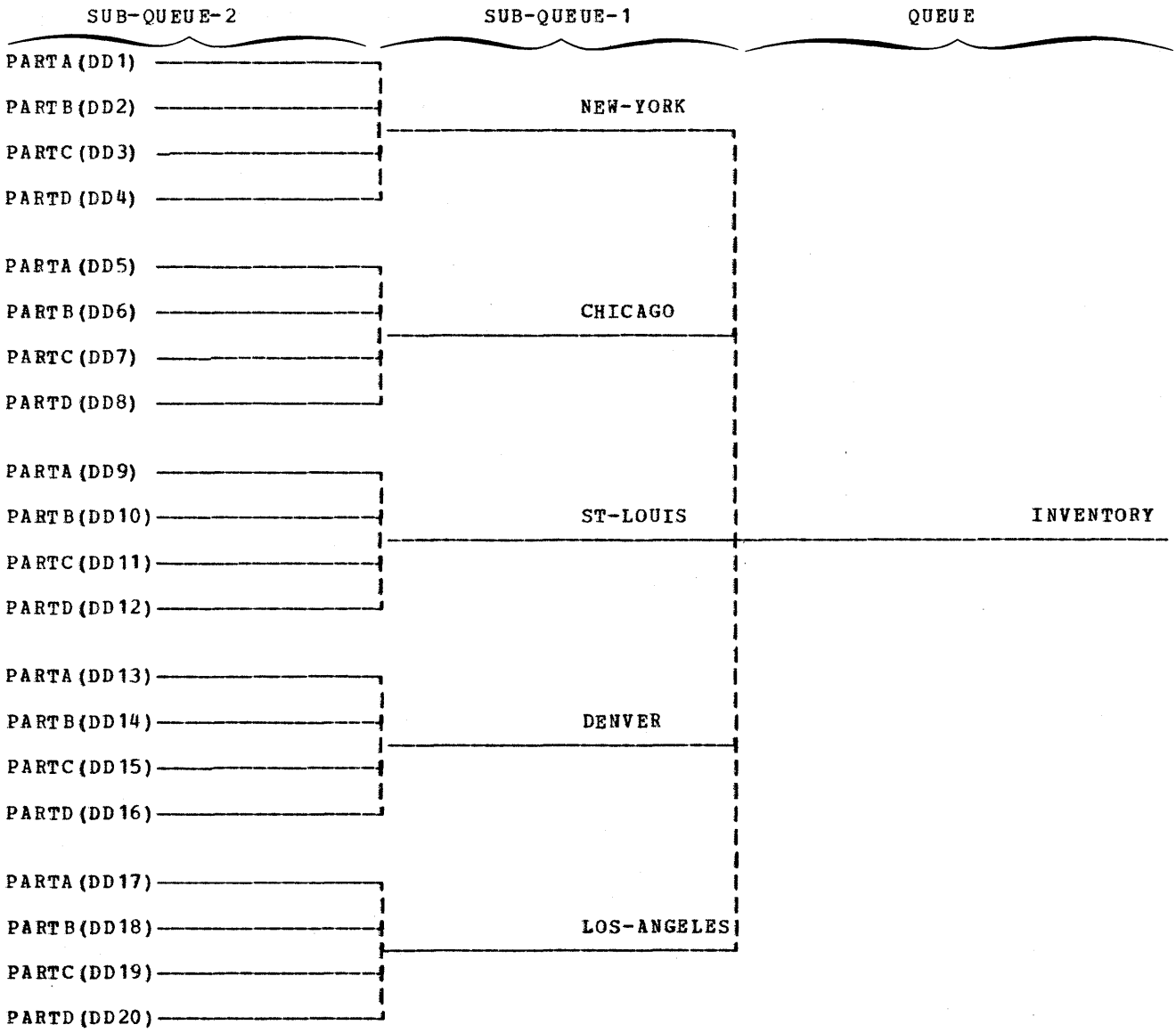


Figure 119. Using ddnames with Queue Structures

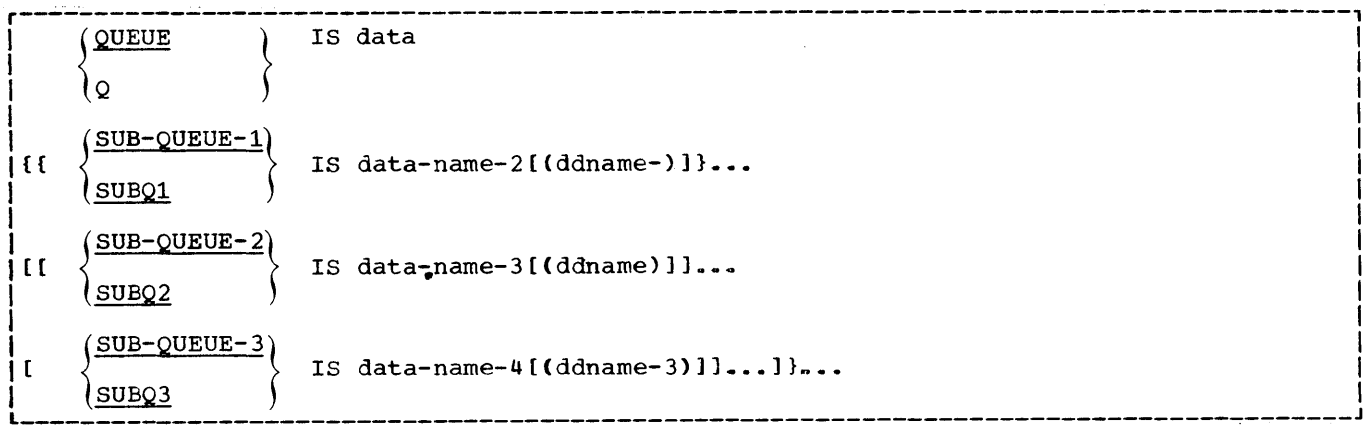


Figure 120. Format for Input to Queue Structure Description Routine

Rules for Queue Structure Description

For each member of the partitioned data set, the input to the Queue Structure Description Routine must take the format shown in Figure 120.

The clauses of the queue structure may be written free form; however, only one clause may appear on each 80-character record. At least one sub-queue level must be specified; no more than 200 sub-queue names may be specified in one queue structure.

The sub-queues at each level must be specified to the Queue Structure Description routine in left-to-right order. When the queue structure is referred to at object program execution time, names at a higher level take priority over names at a lower level. At a given level in the queue structure, names to the left take priority over names to the right.

A queue structure need not include all levels of sub-queues. However, if a lower level is included in one leg of a queue structure, then that leg must include all higher levels.

Each clause of the structure may optionally be followed by a period.

Data-name-1 is the name of the queue structure, and becomes the name of that member of the partitioned data set.

Data-name-2 though data-name-4 are sub-queue names within the data set member.

Note: A data-name cannot contain more than 12 characters.

Each data-name at the lowest (elementary) level of a leg of the queue structure may be a ddname; alternatively, each such data-name may be followed by a parenthesized ddname. If a parenthesized ddname follows a sub-queue name, the left parenthesis must immediately follow the sub-queue name with no intervening spaces. There must be no spaces between the parentheses and the ddname.

CALLING AND CALLED PROGRAMS

A COBOL program can refer to and pass control to other COBOL programs, or to programs written in other languages. A program in another language can refer to and pass control to a COBOL program. A program that refers to another program is a calling program. A program that is referred to is a called program. Control is returned from a called program to the first instruction following the calling sequence in the calling program.

A called program can also be a calling program; that is, a called program can, in turn, call another program. However, a called program cannot call the program that called it, an earlier calling program, or itself. In Figure 121, for instance, program A calls program B; program B calls program C. Therefore:

1. A is considered a calling program by B.
2. B is considered a called program by A.
3. B is considered a calling program by C.
4. C is considered a called program by B.

Control is returned in the same order of calling; that is, a called program (program C) returns control to its own calling program (program B), not to an earlier calling program (program A). Compiler-generated switches (e.g., ON and ALTER) are not reinitialized upon each entrance to the called program, that is, the program is in the last executed state unless it has been the object of a CANCEL statement.

Usually called and calling programs to be executed as a single job step are link-edited together; they must all be included in the same load module. However, with the COBOL dynamic call feature a programmer can request that a called program be link-edited into a separate module and called only if it is needed (see the section "Dynamic Subprogram Linkage", in this chapter).

This chapter describes the accepted linkage conventions for calling and called programs in both COBOL and assembler language and discusses how such programs are link-edited. An example is provided to illustrate the coding required to have proper interface between both COBOL and assembler language calling and called programs. In addition, it includes a discussion of overlay design in which different called programs may, at different times, occupy the same area in main storage. Another example is provided to illustrate one method of accomplishing program linkage using the dynamic overlay technique.

SPECIFYING LINKAGE

Whenever a program calls another program, a link must be established between the two. The calling program must state the entry point of the called program and must specify any identifiers to be passed. The called program must have an entry point and must be able to accept the identifiers. In addition, the called program must establish the linkage for the return of control to the calling program. See Figure 122 for an example of the linkage statements required in a typical calling/called situation.

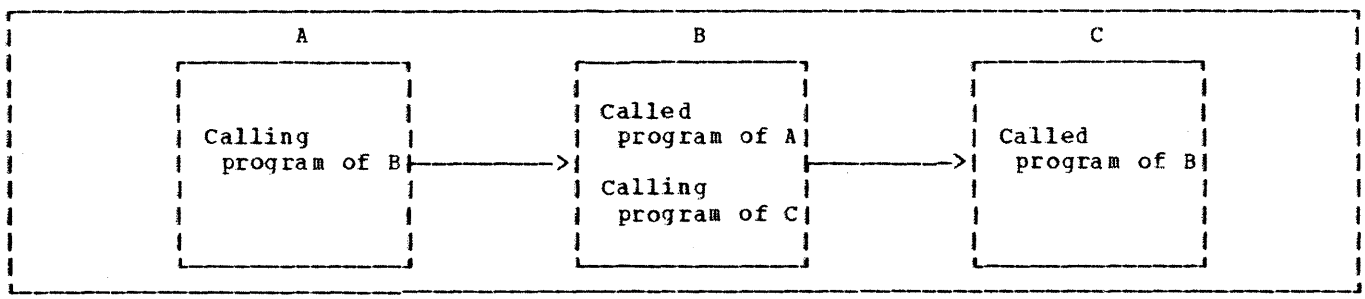


Figure 121. Calling and Called Programs

LINKAGE IN A CALLING COBOL PROGRAM

A calling COBOL program must contain the following statement at the point at which another program is to be called:

```
CALL {literal-1   }
     {identifier-1}

  [USING identifier-list]
  [ON OVERFLOW imperative-statement].
```

Literal-1 or the contents of identifier-1 must be either the name of the program that is being called or the name of an entry point in the called program. The first eight characters of literal-1 or identifier-1 are used to make the correspondence between the calling program and the called program. The identifier-list is one or more data-names, called identifiers and separated by blanks, that are to be passed to the called program.

If the called program is an assembler-language program, the identifier in the USING phrase may also be a file-name or a procedure-name. If the identifier in the USING phrase is a file-name, the COBOL compiler passes the address of the DCB for a queued file, or the address of the DECB for a basic file, as this entry of the identifier-list. The identifier may not be a VSAM file name. This can be used to test bits in the DCB or DECB or to enter some options in the DCB. However, when changing a field of the DCB, precautions should be taken not to contradict the information in other fields or the information in the object code supplied by the compiler, job control language, or other sources. When the identifier in the USING phrase is a procedure-name, the value passed is the beginning address of the procedure. If no identifiers are passed, the USING clause is omitted.

If a non-standard return is executed to a procedure name in a program compiled with OPT, RES, or DYN options, unpredictable results may occur.

The ON OVERFLOW imperative statement will gain control if the called program cannot be dynamically loaded during execution time (situation equivalent to completion code 106-C, 804, 80A, or 878). This allows the user to continue processing. ON OVERFLOW only has meaning if a program is to be dynamically loaded (see the "Dynamic Subprogram Linkage" section later in this chapter).

LINKAGE IN A CALLED COBOL PROGRAM

A called COBOL program must contain two statements.

One of the following statements must be inserted to name the point where the program is to be entered:

```
ENTRY literal-1
  [USING identifier-list].

          or

PROCEDURE DIVISION [USING
  identifier-list].
```

The literal-1 or PROGRAM-ID is the name of the entry point in the called program. It is the same name that appears in the CALL statement of the program that calls this program that the compiler uses. The identifier-list is one or more data-names that correspond to the identifier-list of the CALL statement of the calling program. Each data name of the identifier-list must be defined in the Linkage Section of the Data Division and must have a level number of 01 or 77.

One of the following statements must be inserted at the point at which control is to be returned to the calling program:

```
GOBACK.

    or

EXIT PROGRAM.
```

The GOBACK or EXIT PROGRAM statement enables restoration of necessary registers and returns control to the point in the calling program immediately following the calling sequence.

Note: The GOBACK and EXIT PROGRAM statements may be used in a main program, with the result that any COBOL program can be used as either a calling or a called program, if written with this end in mind. If a GOBACK statement appears within the main program, control is returned immediately to the system; if an EXIT PROGRAM statement appears, it is simply regarded as a null instruction.

A called program may pass a completion code to its caller by storing a value in RETURN-CODE. The calling program may interrogate RETURN-CODE after a return is made from a called program to determine the completion code.

Note: RETURN-CODE may also be used to pass a completion code to the system at the end of a run unit.

Dynamic Subprogram Linkage

With the dynamic subprogram linkage feature, a called program need not be link-edited with the main program. It may instead be link-edited into a separate load module, so that at execution time it is loaded if and only if it is called. Accordingly, the first dynamic call to a subprogram obtains a fresh copy of the subprogram. Subsequent calls to the same subprogram, by either the original caller or any other subprogram in the same region/partition, result in a branch to the same copy of the subprogram in its last-used state until the subprogram is canceled. The first call following a CANCEL statement results in a branch to a fresh copy of the subprogram.

Specification of the DYNAM option in the PARM field of the EXEC statement (see the section on "Compiler Options" in the chapter entitled "Job Control Procedures") makes all calls dynamic. If NODYNAM is in effect, through either user specification at compile time or as the default option, only CALL identifier statements are dynamic; when NODYNAM is in effect, CALL literal statements are static. (For a discussion of the formats possible with the CALL statement, see the publication IBM VS COBOL for OS/VS.)

For an example of a COBOL program that takes advantage of the dynamic CALL/CANCEL feature, see Figure 122.

Notes:

1. When the dynamic CALL is used, the main program and all subprograms in one region/partition should take advantage of the COBOL Library Management Facility (see the "Libraries" chapter). Programs compiled under pre-Version 4 COBOL compilers cannot be dynamically called. Even when the DYNAM option is not specified a program with CALL identifier or CANCEL identifier statements requires the Library Management Feature.

2. The USING option should be included in the CALL statement only if there is a USING option in the called entry point.

3. It is recommended that calling programs use the ON OVERFLOW facility.

```
//CALLJOB       JOB   user information
//STEP1         EXEC    COBUCL,PARM.COB='DYNAM,RESIDENT'
//COB.SYSIN     DD    *
                IDENTIFICATION DIVISION
                PROGRAM-ID. SUBPROG1.
                AUTHOR. J. SMITH
                REMARKS.
                      THIS SUBPROGRAM IS CALLED BY THE MAIN PROGRAM.
                      IT ISSUES A MESSAGE TO INDICATE WHETHER IT IS
                      IN INITIAL OR LAST-USED STATE, AND THEN RETURNS
                      TO THE MAIN PROGRAM.
                ENVIRONMENT DIVISION.
                CONFIGURATION SECTION.
                SOURCE-COMPUTER.  IBM-370.
                OBJECT-COMPUTER.  IBM-370.
                DATA DIVISION.
                WORKING-STORAGE SECTION.
                77  SWITCH  PIC 9 VALUE 0.
                PROCEDURE DIVISION.
                      IF SWITCH=0 DISPLAY 'SUBPROG1 CALLED -- IN
                        INITIAL STATE'
                      GO TO RETURN-POINT.
                      DISPLAY 'SUPROG1 CALLED -- IN LAST-USED STATE'.
                RETURN-POINT.
                      ADD 1 TO SWITCH.
                      EXIT PROGRAM.
/*
//LKED.SYSLMOD DD     DSN=SUBPROGS(SUBPROG1),UNIT=2314,VOL=SER=XXXXXX,
//              DISP=(NEW,KEEP),SPACE=(TRK,(5,1,1))
/*
```

Note: When a subprogram is called dynamically, the (NAME and/or ALIAS) option of the linkage editor is used to identify the module that is accessed by an OS/VS LOAD macro at execution time (see the section entitled "Link-editing COBOL Programs").

Figure 122. Sample Calling and Called Programs Using Dynamic CALL and CANCEL Statements (Part 1 of 3)

```
//CALLJOB2      JOB user information
//STEP1         EXEC COBUCL,PARM.COB='DYNAM,RESIDENT'
//COB.SYSIN     DD  *
                IDENTIFICATION DIVISION.
                PROGRAM-ID. SUBPROG2.
                AUTHOR. J. SMITH
                REMARKS.
                    THIS SUBPROGRAM IS CALLED BY THE MAIN PROGRAM.
                    IF IT IS IN INITIAL STATE, IT ISSUES A MESSAGE
                    TO THAT EFFECT AND RETURNS TO THE MAIN PROGRAM.
                    IF NOT, IT ISSUES A MESSAGE THAT IT IS IN THE
                    LAST-USED STATE, CANCELS SUBPROG1 VIA A CANCEL
                    IDENTIFIER, AND RETURNS TO THE MAIN PROGRAM.
                ENVIRONMENT DIVISION.
                CONFIGURATION DIVISION.
                SOURCE-COMPUTER. IBM-370.
                OBJECT-COMPUTER. IBM-370.
                DATA DIVISION.
                WORKING-STORAGE SECTION.
                77  SWITCH  PIC 9  VALUE 0.
                77  CANCL-ID  PIC X(8).
                PROCEDURE DIVISION.
                    IF SWITCH=0 DISPLAY 'SUBPROG2 CALLED -- IN INITIAL STATE'
                      GO TO RETURN-POINT.
                    DISPLAY 'SUBPROG2 CALLED -- IN LAST-USED STATE'.
                    DISPLAY 'SUBPROG2 CANCELLING SUBPROG1'.
                    MOVE 'SUBPROG1' TO CANCL-ID.
                    CANCEL CANCL-ID.
                RETURN-POINT.
                    ADD 1 TO SWITCH.
                    EXIT PROGRAM.
/*
//LKED.SYSLMOD DD  DSN=SUBPROGS(SUBPROG2),UNIT=2314,VOL=SER=XXXXXX,DISP=OLD
/*
```

Figure 122. Sample Calling and Called Programs Using Dynamic CALL and CANCEL Statements (Part 2 of 3)

```
//CALLJOB3     JOB user information
//STEP1        EXEC COBUCL,PARM.COB='DYNAM,RESIDENT'
//COB.SYSIN    DD  *
             IDENTIFICATION DIVISION.
             PROGRAM-ID. MAINPROG.
             AUTHOR. J. SMITH
             REMARKS.
                 THIS IS A MAIN PROGRAM.  IT CALLS SUBPROG1 AND
                 SUBPROG2 TWICE.  ON THE FIRST CALL, EACH SUBPROGRAM
                 SHOULD BE A FRESH COPY (THAT IS, IN INITIAL STATE).
                 ON THE SECOND CALL, EACH SUBPROGRAM SHOULD BE IN ITS
                 LAST-USED STATE.  WHEN SUBPROG2 IS CALLED THE SECOND
                 TIME, IT CANCELS SUBPROG1.  THEN MAINPROG CALLS
                 SUBPROG1 AGAIN, AND AGAIN A FRESH COPY OF THIS
                 SUBPROGRAM SHOULD BE MADE AVAILABLE.
                 THE OUTPUT FROM THIS RUN SHOULD READ AS FOLLOWS:
                 'BEGIN MAINPROG.
                 MAINPROG CALLING SUBPROG1.
                 SUBPROG1 CALLED -- IN INITIAL STATE.
                 MAINPROG CALLING SUBPROG2.
                 SUBPROG2 CALLED -- IN INITIAL STATE.
                 MAINPROG CALLING SUBPROG1.
                 SUBPROG1 CALLED -- IN LAST-USED STATE.
                 MAINPROG CALLING SUBPROG2.
                 SUBPROG2 CALLED -- IN LAST-USED STATE.
                 SUBPROG2 CANCELLING SUBPROG1.
                 MAINPROG CALLING SUBPROG1.
                 SUBPROG1 CALLED -- IN INITIAL STATE.
                 MAINPROG CANCELLING SUBPROG1 AND SUBPROG2.
                 END MAINPROG.'
             ENVIRONMENT DIVISION.
             CONFIGURATION SECTION.
             SOURCE-COMPUTER. IBM-370.
             OBJECT-COMPUTER. IBM-370.
             DATA DIVISION.
             WORKING-STORAGE SECTION.
             77  SWITCH  PIC 9 VALUE 0.
             77  CALLID  PIC X(8).
             PROCEDURE DIVISION.
                 DISPLAY 'BEGIN MAINPROG'.
             START-CALLS.
                 IF SWITCH IS LESS THAN 2 PERFORM CALL1,
                 PERFORM CALL2,
                 GO TO START-CALLS.
                 PERFORM CALL1.
                 DISPLAY 'MAINPROG CANCELLING SUBPROG1 AND SUBPROG2'.
                 CANCEL 'SUBPROG1','SUBPROG2'.
                 DISPLAY 'END MAINPROG'.
                 STOP RUN.
             CALL1.
                 MOVE 'SUBPROG1' TO CALLID.
                 DISPLAY 'MAINPROG CALLING SUBPROG1'.
                 CALL CALLID.
             CALL2.
                 MOVE 'SUBPROG2' TO CALLID.
                 DISPLAY 'MAINPROG CALLING SUBPROG2'.
                 CALL CALLID.
                 ADD 1 TO SWITCH.
/*
//LKED.SYSLMOD DD  DSN=SUBPROGS(MAINPROG),UNIT=2314,VOL=SER=XXXXXX,DISP=OLD
//GO   EXEC PGM=MAINPROG
//STEPLIB      DD  DSN=SUBPROGS,UNIT=2314,VOL=SER=XXXXXX,DISP=OLD
//SYSOUT       DD  SYSOUT=A
/*
```

Figure 122. Sample Calling and Called Programs Using Dynamic CALL and CANCEL Statements (Part 3 of 3)

Correspondence of Identifiers in Calling and Called Programs

The number of data-names in the identifier list of a calling program must be the same as the number of data-names in the identifier list of the called program. There is a one-to-one correspondence; that is, the first identifier of the calling program is passed to the first identifier of the called program, the second identifier of the calling program is passed to the second identifier of the called program, and so forth.

Only the address of an identifier list is passed. Consequently, the data-name that is an identifier of the calling program and the data-name that is the corresponding identifier of the called program both refer to the same locations in main storage. The pair of names, however, need not be identical, but the data descriptions must be equivalent. For example, if an identifier of the calling program is a level-77 data-name of a character string of length 30, its corresponding identifier of the called program could also be a level-77 data-name of a character string of length 30, or the identifier of the called program could be a level-01 name with subordinate names representing character strings whose combined length is 30.

Although all identifiers of the called program in the ENTRY statement must be described with level numbers of 01 or 77, there is no such restriction made for identifiers of the calling program in the CALL statement. An identifier of the calling program may be a qualified name or a subscripted name.

FILE-NAME ARGUMENTS

A calling COBOL program that calls an assembler-language program can pass file-names, in addition to data-names, as identifiers. For a queued file, the file-name is passed as the address of the DCB (Data Control Block) and for a basic file, the file-name is passed as the address of the DECB (Data Event Control Block). A VSAM file name cannot be passed.

LINKAGE IN A CALLING OR CALLED ASSEMBLER-LANGUAGE PROGRAM

In a COBOL program, the expansions of the linkage statement provide the save and return coding that is necessary to establish linkage between the calling and the called programs. Assembler-language programs must be prepared in accordance with the basic linkage conventions of the operating system. Figure 123 shows the conventions for use of general registers as linkage registers.

Register Number	Register Use	Contents
1	Identifier	Address of the list that is passed to the called program.
13	Save Area	Address of an area (of 18 fullwords) to be used by the called program to save registers.
14	Return	Address of the location in the calling program to which control should be returned after execution of the called program.
15	Entry Point[1]	Address of the entry point in the called program to which control is to be transferred.

[1]Register 15 is also used as a return code register. The return code indicates whether or not any exceptional conditions occurred during execution of the called program. When control is returned to the COBOL program, it automatically moves the contents of register 15 into the special register RETURN-CODE.

Figure 123. Linkage Registers

Conventions Used in a Calling Assembler-Language Program

A calling assembler-language program must reserve a save area of 18 words, beginning on a fullword boundary, to be used by the called program for saving registers. It must load the address of this area into register 13. If the program is to pass identifiers, an identifier list must be prepared, and the address of the identifier list must be loaded into register 1. The calling program must load the address of the return point into register 14, and it must load the address of the entry point of the called program into register 15.

The identifier list is a group of contiguous fullwords, each of which is an address of a data item to be passed to the called program. The identifier list must begin on a fullword boundary. The high-order bit of the last identifier, by convention, is set as a flag of one to indicate the end of the list. Figure 125 shows a portion of an assembler-language program that illustrates the conventions used in a calling program.

The Assembler-Language Caller As Main Program: In an all-COBOL environment, the first COBOL program in the run unit that is invoked for execution becomes the main program. It may be invoked via an EXEC JCL card, or by linkage from some other program. The fact that it is the first one invoked under the task establishes it as a main program. The main program calls a library subroutine which (1) initializes the subroutine communications data area ILBOCOM, and (2) saves a pointer to the register save area of the main program. (A main program also executes a GOBACK statement is if STOP RUN had been coded. See below.)

Any COBOL program invoked after the COBOL main program has begun to execute is considered a subprogram. A COBOL subprogram knows it is a subprogram rather than a main program because when it gets control, ILBOCOM has already been initialized.

Execution of the STOP RUN statement by any COBOL program causes a library subroutine to be called which terminates the run unit. This subroutine closes COBOL DCBs (such as SYSIN, SYSOUT, and SYSDBOUT), and resets certain flags and fields in ILBOCOM. If ENDJOB (see below) is in effect, it performs further cleanup. Then it picks up the main program save area (register 13) pointer that was saved when ILBOCOM was initialized, follows the save area back chain pointer from there, and executes a RETURN to the caller of the main program.

Execution of the GOBACK statement in a COBOL subprogram causes a return to the routine that called it. However, execution of this statement in a COBOL main program causes STOP RUN processing to occur (see above).

The compilation option ENDJOB causes the STOP RUN subroutine to free all main storage acquired during the run unit, and to delete any subprograms or subroutine library modules that were loaded during the run unit. The only parts of the run unit left after a STOP RUN with ENDJOB has executed are any load modules that were loaded by an assembler-language program rather than a COBOL program, and the library subroutine ILBOSTT. These must be deleted by an assembler-language calling program.

If the compilation option RESIDENT is used, it must be used by all COBOL programs in a run unit. It causes linkage to library subroutines to be established at execution time, rather than at link edit time, and maintains a list of all subroutine library modules for which it has issued a LOAD.

The compilation option DYNAM causes dynamic LOADs and DELETEs to be done for subprograms specified in CALL and CALL CANCEL statements in COBOL programs in a run unit. It maintains a list of all subprograms it loads in the run unit.

When a COBOL program is invoked by Job Management via an EXEC JCL card, the situation is straightforward. The COBOL program so invoked is the main program and begins the run unit. A STOP RUN statement causes a return to the system. RESIDENT and DYNAM may be used to improve efficiency of main storage utilization. The ENDJOB option is not important, in this case, because Job Management frees main storage and deletes load modules used by the jobstep task when it terminates.

A COBOL program may also be invoked by use of CALL, LOAD and CALL, LINK, or ATTACH. If this is done, then the following cautions must be observed:

1. Use the ENDJOB option to make sure the region or partition is cleared of GETMAIN-acquired storage and loaded modules when the run unit ends. Then, after the run unit has ended, have the calling program issue a DELETE for module ILBOSTT. If LOAD was used in an assembler language program to bring in any subprograms, a DELETE should also be issued for those modules.

2. COBOL does not support concurrent running of multiple COBOL subtasks in one region or partition. If a COBOL subtask is attached, it must terminate before the next one is attached. Multiple subtasks may work in very limited circumstances. The following would definitely prevent multiple subtasking from working: RESIDENT or DYNAM options, or use of the same data set by two different subtasks.

The COBOL subroutine library provides a means whereby an assembler-language program may become a main program; this is accomplished by a call to the library subroutine entry point ILBOSTP0, which causes ILBOCOM to be loaded and initialized. If this is done first, all COBOL programs subsequently called will behave as subprograms. This provides several other advantages for certain types of applications: execution of a GOBACK to the calling assembler-language routine will not cause STOP RUN processing--COBOL DCBs will remain open, no storage will be freed, no subroutines deleted; the assembler-language main program can continue calling its subprograms and adding to its open data sets until one of the subprograms executes a STOP RUN. Execution of a STOP RUN will cause return to the caller of the assembler-language program, rather than to the assembler-language program itself.

If this type of processing is desired, the assembler-language program must issue a CALL to the library subroutine entry point ILBOSTP0 before making any call to a COBOL program. (Please note that ILBOSTP0 destroys the contents of Registers 0 and 1.) The following considerations apply to the use of the CALL to ILBOSTP0 (within a single run unit):

1. The subroutine ILBOSRV must be included in the link edit of the calling program.
2. If RESIDENT is used, ILBOCOM must not be included in any link edit. Please note that the link edit of ILBOSRV to serve the ILBOSTP0 interface must leave a weak external reference for ILBOCOM unresolved.
3. If NORESIDENT is used, all programs and subroutines must be link edited together.
4. The CALL to ILBOSTP0 may not be used if the COBOL program is invoked by LINK or ATTACH.
5. If DYNAM is used, all COBOL programs must be link edited in separate load modules.

The following table distinguishes the procedures for initialization with several representative types of program linkage.

In Cases 1, 2, and 3, the CALL to ILBOSTP0 interface is being used. In Case 4, it is not.

Case 1. Caller should have a V-type address constant for ILBOSTP0 and should call ILBOSTP0 using it. Library subroutine ILBOSRV must be included in the link edit, but ILBOCOM must not be. The COBOL program may then be called as a subroutine.

Case 2. Caller should have a V-type address constant for ILBOSTP0 and should call ILBOSTP0 using it. Library subroutine ILBOSRV must be included in the link edit of the caller, but ILBOCOM must not be. No library subroutines should be included in the link edit of the called COBOL program. After the call to ILBOSTP0, caller should issue a LOAD for the called COBOL program. The COBOL program may then be called as a subroutine.

Case 3. Caller should have a V-type address constant for ILBOSTP0 and should

	Caller and COBOL Program Linked Together	Caller and COBOL Program Linked Separately
Called COBOL Program Compiled with RESIDENT option	Case 1	Case 2
Called COBOL Program Compiled with NORESIDENT option	Case 3	Case 4

call ILBOSTP using it. The COBOL program may then be called as a subroutine. With the NORES option, all subroutines will be link edited into the load module.

Case 4. Caller must LOAD the COBOL module, then call it. Caller may use LINK or ATTACH instead of LOAD, CALL, and DELETE.

Figure 124 is an example of Case 2 and illustrates one fairly typical and useful configuration for assembler-language programs calling COBOL-language routines. COBOL1 is a transaction processor of some type. ASM1 collects transactions and passes them to ASM2, which passes them one at a time to COBOL1. When all transactions are done, ASM2 calls COBOL2 to execute a STOP RUN, perform ENDJOB cleanup, and return to ASM1.

Because ASM2 has first called ILBOSTP0, all COBOL programs in the run unit will behave as subprograms. Thus when COBOL1 returns control to ASM2 after a call, it remains in main storage in its last-used state. There is no overhead for clean-up or for reinitialization on the succeeding call. COBOL1 can return control by either an EXIT PROGRAM or a GOBACK. (Because it is behaving as a subprogram, GOBACK is equivalent to EXIT PROGRAM.) ENDJOB processing is not done for any of the exits from COBOL1. It is only done once, in response to the STOP RUN in COBOL2.

Two of the several possible ways to compile and link edit this application are:

1. Compiling with ENDJOB and NORESIDENT. ASM2, COBOL1, COBOL2, and all required library subroutines would be link edited together. ASM1 would issue a LOAD for ASM2, then CALL it. Upon receiving control again, ASM1 would DELETE ASM2.

2. Compiling with ENDJOB and RESIDENT. ASM2 and ILBOSRV would have to be link edited together. COBOL1 could also be included. In that case, ASM1 would issue a LOAD for ASM2, then CALL it. ASM2 would have to issue a LOAD for COBOL2 before calling it. When control is returned to ASM1, it must issue a DELETE for ASM2, COBOL2, and ILBOSTT.

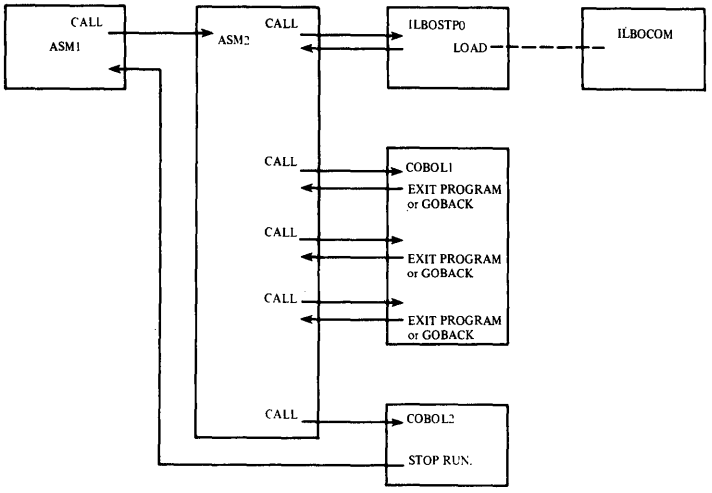


Figure 124. Effect of STOP RUN Statement

Conventions Used in a Called Assembler-Language Program

A called assembler-language program must save the registers and store other pertinent information in the save area passed to it by the calling program (the layout of the save area is shown in Figure 127). A called program must also contain a return routine that (1) loads the address of the save area back into register 13, (2) restores the contents of other registers, loading the return address in register 14, and (3) optionally, sets flags in the high-order eight bits of word 4 of the save area to 1's to indicate that the return occurred. It can then branch to the address in register 14 to complete the return.

Figure 126 (Part 6) shows a portion of an assembler-language program that illustrates the conventions used in called programs that are also calling programs. Figure 133 shows the JCL suggested for compiling, link-editing, and executing a calling assembler-language program and a called COBOL program.

```
            LA    13,AREA          LOADS THE ADDRESS OF THIS PROGRAM'S SAVE AREA INTO
*                                  REGISTER 13.
            .
            .
            .
AREA        DS    18F              RESERVES 18 WORDS FOR THE SAVE AREA
            .
            .
            .
*  CALLING SEQUENCE
            LA    1,ARGLST         LOADS INTO REGISTER 1 THE ADDRESS OF THE IDENTIFIER
*                                  LIST TO BE PASSED.  TRANSFERS CONTROL TO THE ENTRY
            CALL  COBREGN2         POINT OF THE CALLED PROGRAM.  (THE CALL MACRO
*                                  INSTRUCTION GENERATES CODING THAT LOADS A V-TYPE
*                                  ADDRESS CONSTANT -- COBREGN2 -- INTO REGISTER 15 AND
*                                  PLACES INTO REGISTER 14 THE RETURN ADDRESS, THAT IS,
*                                  THE ADDRESS OF THE FIRST BYTE FOLLOWING THE MACRO
*                                  EXPANSION.
            .
            .
            .
*  PARAMETER LIST
            DS    0F               THIS PARAMETER LIST CONTAINS ONLY ONE ARGUMENT.
ARGLST      DC    X'80'            FIRST BYTE OF LAST ARGUMENT (ONLY
            DC    AL3(ARGUMENT)    ARGUMENT IN THIS PROGRAM) SETS BIT 0
ARGUMENT    DC    C'1'             TO 1.
```

Note: Since the calling program containing this coding could previously have been called by another program, it also could establish linkage between the save area it has received and the save area it passes to the called program. It would store in word three of the old save area the address of the new save area, and it would store in word two of the new save area the address of the old save area.

Figure 125. Sample Linkage Coding Used in a Calling Assembler-Language Program

COMMUNICATION WITH OTHER LANGUAGES

An American National Standard COBOL program may communicate at object time with programs written in other source program languages, such as COBOL F, PL/I, and FORTRAN. However, such communication must follow the same conventions discussed above under "Linkage in a Called or Calling Assembler-Language Program." The relatively few problems that may arise in using American National Standard COBOL with COBOL F usually have to do with slightly different boundary alignments, slack-byte insertion, different meanings for the same reserved word, and so on.

There is a greater disparity between American National Standard COBOL and FORTRAN, much of it stemming from the basic differences in the applications for which these languages were developed. (FORTRAN is process oriented and does comparatively little file processing; COBOL, on the other hand, is definitely file oriented and is not mathematically self-sufficient.) Care must be taken, therefore, in attempting to pass arguments between American National Standard COBOL and FORTRAN programs.

The use of COBOL and PL/I together requires a number of important considerations. See the publications OS/VS Linkage Editor and Loader and either OS PL/I Optimizing Compiler Programmer's Guide, Order No. SC33-0006 or OS PL/I Checkout Compiler: Programmer's Guide, Order No. SC33-0007.

Abnormal terminations in non-COBOL programs calling COBOL programs compiled with either the STATE or the SYMDMP option (see the chapter entitled "Symbolic Debugging Features") cause generation of the following misinformation:

- Incorrect number for the statement responsible for the abnormal termination. The last COBOL statement in the called program executed before the return to the calling non-COBOL program is given in the "Last Card Number Executed" message.

- Incorrect PROGRAM-ID when such an abnormal termination occurs after return from the called COBOL program. The PROGRAM-ID message contains the user-specified name for the called COBOL program.

SAMPLE CALLING AND CALLED PROGRAMS

The following set of programs (Figure 126) contains a sample COBOL main-line program, COBMAIN, which calls COBOL and assembler-language programs using arguments that represent a data-item and a file-name.

Some of the called programs (COBOL1, COBOL1B, and ASSMPGM) are themselves calling programs. Program COBREGN0 is called by several programs, each of which enters at a different entry point within the program.

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  COBMAIN.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.  IBM-370.
OBJECT-COMPUTER.  IBM-370.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT FILE-X ASSIGN TO UR-2540R-S-INFILE.
I-O-CONTROL.
DATA DIVISION.
FILE SECTION.
FD  FILE-X
    RECORD CONTAINS 80 CHARACTERS
    LABEL RECORD IS OMITTED.
01  IN-REC.
    05  TYPEN PIC X.
    05  HOLDER PIC X.
    05  FILLER PIC X(78).
WORKING-STORAGE SECTION.
77  SIGNAL PIC X(8).
PROCEDURE DIVISION.
    .
    .
    .
    OPEN INPUT FILE-X.
    READ FILE-X AT END GO TO CLOSE-FILE.
    .
    .
    .
    CALL 'COBOL1' USING IN-REC.
    .
    .
    .
    CALL 'COBREGN1' USING IN-REC.
    .
    .
    .
    CALL 'ASSMRTN' USING SIGNAL.
    .
    .
    .
CLOSE-FILE.  CLOSE FILE-X.
    .
    .
    .
    STOP RUN.
```

Figure 126. Sample Calling and Called Programs (Part 1 of 7)

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  COBOL1.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.  IBM-370.
OBJECT-COMPUTER.  IBM-370.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
I-O-CONTROL.
DATA DIVISION.
FILE SECTION.
WORKING-STORAGE SECTION.
77  TRANS-COBL PIC X(7).
LINKAGE SECTION.
01  PASS-REC.
    05  FILLER PIC X.
    05  TRANS-VALUE PIC X.
    05  FILLER PIC X(78).
PROCEDURE DIVISION USING PASS-REC.
    .
    .
    .
    CALL 'COBOL1A' USING TRANS-COBL.
    .
    .
    .
    CALL 'COBOLIB' USING TRANS-COBL.
    .
    .
    .
    GOBACK.
```

Figure 126. Sample Calling and Called Programs (Part 2 of 7)

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  COBOL1A.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.  IBM-370.
OBJECT-COMPUTER.  IBM-370.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
I-O-CONTROL.
DATA DIVISION.
FILE SECTION.
WORKING-STORAGE SECTION.
LINKAGE SECTION.
77  TRANS-COB1A PIC X(7).
PROCEDURE DIVISION USING TRANS-COB1A.
    .
    .
    .
    GOBACK.
```

Figure 126. Sample Calling and Called Programs (Part 3 of 7)

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  COBOL1B.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.  IBM-370.
OBJECT-COMPUTER.  IBM-370.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
I-O-CONTROL.
DATA DIVISION.
FILE SECTION.
WORKING-STORAGE SECTION.
77  TRANS-COBREGN PIC X(7).
LINKAGE SECTION.
77  TRANS-COB1B PIC X(7).
PROCEDURE DIVISION USING TRANS-COB1B.
    .
    .
    .
    CALL 'COBREGN0' USING TRANS-COBREGN.
    .
    .
    .
    GOBACK.
```

Figure 126. Sample Calling and Called Programs (Part 4 of 7)

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  COBREGN0.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.  IBM-370.
OBJECT-COMPUTER.  IBM-370.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
I-O-CONTROL.
DATA DIVISION.
FILE SECTION.
WORKING-STORAGE SECTION.
LINKAGE SECTION.
77  TRANS-COB PIC X(7).
77  TRANS-ASSM PIC X(4).
01  PASS-REC.
    05  FILLER PIC X.
    05  TRANS-VALUE PIC X.
    05  FILLER PIC X(78).
PROCEDURE DIVISION USING TRANS-COB.
    .
    .
    .
    GOBACK.
B.  ENTRY 'COBREGN1' USING PASS-REC.
    .
    .
    .
    GOBACK.
C.  ENTRY 'COBREGN2' USING TRANS-ASSM.
    .
    .
    .
    GOBACK.
```

Figure 126. Sample Calling and Called Programs (Part 5 of 7)

```
ASSMPGM   START 0
          PRINT NOGEN
          ENTRY ASSMRTN          ESTABLISHES ASSMRTN AS AN EXTERNAL NAME THAT CAN BE
*                                 REFERRED TO IN ANOTHER PROGRAM.
          USING ASSMRTN,15

*  SAVE ROUTINE
ASSMRTN   SAVE  (14,12)          STORES THE CONTENTS OF REGISTERS 14, 15, 0, AND 1
*                                   IN WORDS 4, 5, 6, AND 7 OF THE SAVE AREA.
*                                   THESE ARE CONVENTIONAL LINKAGE REGISTERS.
*                                   REGISTERS 2 THROUGH 12, WHICH ARE NOT
*                                   ACTUALLY USED FOR LINKAGE, ARE SAVED IN SUBSEQUENT
*                                   WORDS OF THE SAVE AREA.  THE EXPANDED CODE OF THE
*                                   SAVE MACRO INSTRUCTION USES REGISTER 13, WHICH
*                                   CONTAINS THE ADDRESS OF THE SAVE AREA, IN
*                                   EFFECTING THE STORAGE OF REGISTERS.
          LR    10,15
          DROP  15
          USING ASSMRTN,10
          LR    11,13            LOADS THE ADDRESS OF THE SAVE AREA INTO REGISTER 11,
*                                   WHICH WILL SUBSEQUENTLY BE USED TO REFER TO THE
*                                   SAVE AREA.

          LA    13,AREA          LOADS THE ADDRESS OF THIS PROGRAM'S SAVE AREA INTO
*                                   REGISTER 13.

          ST    13,8(11)         STORES THE ADDRESS OF THIS PROGRAM'S SAVE AREA INTO
*                                   WORD 3 OF THE SAVE AREA OF THE CALLING PROGRAM.

          ST    11,4(13)         STORES THE ADDRESS OF THE PREVIOUS SAVE AREA INTO
*                                   WORD 2 OF THIS PROGRAM'S SAVE AREA.
          B     PROCESS
AREA      DS    18F              RESERVES 18 WORDS FOR THE SAVE AREA.

PROCESS   L     2,0(1)           LOADS INTO REGISTER 2 THE ADDRESS OF THE IDENTIFIER-
*                                   LIST PASSED TO THE PROGRAM.  THE ADDRESS OF THE
*                                   IDENTIFIER-LIST IS ALWAYS PASSED IN REGISTER 1,
*                                   WHICH IS USED HERE AS THE BASE REGISTER TO GET THE
*                                   ADDRESS.  SUBSEQUENT REFERENCES TO THE IDENTIFIER
*                                   WILL USE REGISTER 2 AS THE BASE REGISTER FOR THAT
*                                   ADDRESS.  (IF A VARIABLE-LENGTH IDENTIFIER-LIST
*                                   COULD BE USED IN CALLING THIS PROGRAM, EACH
*                                   IDENTIFIER WOULD BE TESTED FOR A ONE IN THE
*                                   HIGH-ORDER BIT.)

          {User-written program statements}

*  CALLING SEQUENCE
          LA    1,ARGLST         LOADS INTO REGISTER 1 THE ADDRESS OF THE IDENTIFIER-
*                                   LIST TO BE PASSED.

          CALL  COBREGN2         TRANSFERS CONTROL TO THE ENTRY POINT OF THE CALLED
*                                   PROGRAM.  [THE CALL MACRO INSTRUCTION GENERATES
*                                   CODING THAT LOADS A V-TYPE ADDRESS CONSTANT --
*                                   COBREGN2 -- INTO REGISTER 15 AND PLACES INTO
*                                   REGISTER 14 THE RETURN ADDRESS (THAT IS, THE
*                                   ADDRESS OF THE FIRST BYTE FOLLOWING THE MACRO
*                                   EXPANSION) ].

          {User-written program statements}
```

Figure 126. Sample Calling and Called Programs (Part 6 of 7)

```
*  CALLING SEQUENCE
          LA    1,ARGLST          LOADS INTO REGISTER 1 THE ADDRESS OF THE
                                  IDENTIFIER-
*                                   LIST TO BE PASSED.

          CALL  COBREGN2          TRANSFERS CONTROL TO THE ENTRY POINT OF THE CALLED
*                                   PROGRAM.  [THE CALL MACRO INSTRUCTION GENERATES
*                                   CODING THAT LOADS A V-TYPE ADDRESS CONSTANT --
*                                   COBREGN2 -- INTO REGISTER 15 AND PLACES INTO
*                                   REGISTER 14 THE RETURN ADDRESS (THAT IS, THE
*                                   ADDRESS OF THE FIRST BYTE FOLLOWING THE MACRO
*                                   EXPANSION) ].

          {User-written program statements}

*  RETURN ROUTINE
          L     13,4(13)          LOADS THE ADDRESS OF THE PREVIOUS SAVE AREA
*                                   BACK INTO REGISTER 13.

          RETURN(14,12),T,RC=(15) THIS RETURN MACRO INSTRUCTION RESTORES THE SAVED
*                                   REGISTERS (14, 15, AND 0 THROUGH 12).  THE RETURN
*                                   ADDRESS IS RESTORED TO REGISTER 14, AND THE
*                                   EXPANSION INCLUDES A BRANCH TO THAT INSTRUCTION.
*                                   THE 'T' IN THE RETURN MACRO INSTRUCTION CAUSES
*                                   THE EIGHT HIGH-ORDER BITS OF WORD 4 OF THE SAVE
*                                   AREA TO BE SET TO ONES AS AN INDICATION THAT THE
*                                   RETURN HAS OCCURRED.  THE RC=(15) PARAMETER
                                    INDICATES THAT THIS PROGRAM IS PASSING A RETURN
                                    CODE IN REGISTER 15.

*  PARAMETER LIST
          DS    0F                THIS PARAMETER LIST CONTAINS ONLY 1 ARGUMENT.
ARGLST    DC    X'80'
          DC    AL3(ARGUMENT)     FIRST BYTE OF LAST ARGUMENT (ONLY ARGUMENT IN
ARGUMENT  DC    C'1'                THIS PROGRAM) SETS BIT 0 TO 1.
          END
```

Figure 126. Sample Calling and Called Programs (Part 7 of 7)

LINK-EDITING PROGRAMS

Each time an entry point is specified in a called program, an external name is defined (except when a program is compiled using the DYNAM and RESIDENT compiler options). An external name is a name that can be referred to by another separately compiled or assembled program. Each time an entry name is specified in a calling program, an external reference is defined except when a program is compiled using the DYNAM and RESIDENT compiler options. An external reference is a symbol that is defined as an external name in another separately compiled or assembled program. The linkage editor resolves external names and references and combines calling and called programs into a format suitable for execution together, i.e., as a single load module except when programs are compiled with dynamic CALL statements and/or the RESIDENT option (see the section entitled "Programs Compiled with the DYNAM and/or RESIDENT Options").

Load modules of both calling and called programs are used as input to the linkage editor. There are two kinds of input, primary and additional. Primary input consists of a sequential data set that contains one or more separately compiled object modules and/or linkage editor control statements. The primary input can contain object modules that are either calling or called programs or both. Additional input consists of object modules or load modules that are not part of the primary input data set but are to be included in the load module. The additional input may be in the form of (1) a sequential data set consisting of one or more object modules with or without linkage editor control statements, or (2) libraries containing object modules with or without linkage editor control statements, or (3) libraries consisting of load modules. Note that the secondary input (all libraries and/or data sets) must be composed of either all object modules or all load modules, but it cannot contain both types. The additional input is specified by

Word No.	Area No.	Contents
1	AREA	Used by COBOL.
2	AREA +4	Address (passed by the calling program) of the save area used by the calling program. This is the address of a save area that was passed to the called program by the program that called the called program.
3	AREA +8	Address (stored by the called program) of the next save area, that is, the save area that the called program provides for a program that it calls. The called program need not reserve a save area if it does not, in turn, call another program.
4	AREA +12	Return address (contents of register 14) stored by the called program.
5	AREA +16	Entry point address (contents of register 15) stored by the called program.
6	AREA +20	Contents of register 0 (stored by the called program).
7	AREA +24	Contents of register 1 (stored by the called program); that is, the address of the identifier list passed to the called program.
8	AREA +28	Contents of registers 2 through 12 (stored by the called program).
.	.	
.	.	
.	.	
18	AREA +68	

Figure 127. Save Area Layout and Contents

linkage editor control statements in the primary input and a DD statement for each additional input data set. Additional input may contain either calling or called programs or both.

Note: Each additional input data set may itself contain external references or names and linkage editor control statements that specify more additional input.

SPECIFYING PRIMARY INPUT

The primary input data set is specified for linkage editor processing by the SYSLIN DD statement. The linkage editor must always have a primary input data set specified by a SYSLIN DD statement whether or not there are called or calling programs and even if the primary input data set contains only linkage editor control statements. The SYSLIN DD statement that specifies the primary input is discussed in "Linkage Editor Data Set Requirements" (see "Example of Linkage Editor Processing" for a discussion of how to specify a primary input data set that contains more than one object module along with linkage editor control statements).

SPECIFYING ADDITIONAL INPUT

Additional input data sets are specified by linkage editor control statements and a DD statement for each additional input data set.

The linkage editor control statements that specify additional input are INCLUDE and LIBRARY.[1] A primary input data set may consist entirely of such statements. The INCLUDE and LIBRARY statements may be placed before, between, or after object modules or other control statements in either primary or additional input data sets. One method of using these statements is shown in Figure 134.

Note: Additional input often contains members of libraries (see "Specifying Libraries as Additional Input" in "Libraries").

---

[1]The operation field in a linkage editor control statement must start after column 1. The operand field must be preceded by at least one blank.

INCLUDE Statement

The INCLUDE statement is used to include an additional input data set that is either a member of a library or a sequential data set. The format is:

```
| Operation | Operand                                  |
|-----------|------------------------------------------|
| INCLUDE   | ddname[(member-name                      |
|           |   [,member-name]...)]                    |
|           |   [,ddname[(member-name                  |
|           |   [,member-name...])]]...                |
```

where ddname indicates the name of the DD statement that specifies the library or sequential data set, and member-name is the name of the library member that is to be included. Member-name is not used when the additional input data set is not a member of a partitioned data set.

LIBRARY Statement

The LIBRARY statement is used to include additional input that may be required to resolve external references. The format is:

```
| Operation | Operand                                  |
|-----------|------------------------------------------|
| LIBRARY   | ddname(member-name                       |
|           |   [,member-name]...)                     |
|           |   [,ddname(member-name                   |
|           |   [,member-name...])]...                 |
```

where ddname indicates the name of the DD statement that specifies the library, and member-name is the name of the member of the library.

The LIBRARY statement differs from the INCLUDE statement in that libraries specified in the LIBRARY statement are not searched for additional input until all other processing, except references reserved for the automatic library call, is completed by the linkage editor. Any additional module specified by an INCLUDE statement is incorporated immediately, whenever the INCLUDE statement is encountered.

ALIAS Statement

The ALIAS statement specifies additional names for the output library member, and can also display names of additional entry points. If a load module has more than one entry point or more than one CSECT and the user wishes to access that alternate entry at execution time via a dynamic CALL, he should specify an ALIAS with the same symbolic name as the desired entry point or CSECT. The format is:

```
|Operation| Operand                                          |
|---------|--------------------------------------------------|
|         | {symbol       } [,symbol       ]                 |
|ALIAS    | {external name} [,external name]                 |
```

where symbol specifies an alternate name for the load module, and external name specifies a name that is defined as a control section name or entry name in the output module.

If the linkage-editor input includes an ALIAS statement, the symbolic name specified is identified with the relative location of the entry point or CSECT name that matches the ALIAS. If there is no matching entry point or CSECT name, the ALIAS is identified with relative location zero in the load module.

Note: If the user plans to dynamically call a subprogram at an ENTRY point, an ALIAS card is required in the link-edit step for that entry point.

NAME Statement

The NAME statement specifies the name of the load module created from the preceding input modules and serves as a delimiter for input to the load module. The NAME statement may be used to assign a symbolic name to a load module. This symbolic name is entered in the directory of the partitioned data set that contains the module, and allows the module to be accessed at execution time by an OS/VS LOAD, LINK, XCTL, or ATTACH macro. The format is:

```
| Operation | Operand                                  |
|-----------|------------------------------------------|
| NAME      | member-name [(R)]                        |
```

where member-name specifies the name to be assigned to the load module that is created

from the preceding input modules, and (R) indicates that this load module replaces an identically named module in the input module library. (If the module is not a replacement, the parenthesized value (R) has no effect.)

If the linkage-editor input includes a NAME statement, the symbolic name specified is always identified with relative location zero in the load module, unless there is an ENTRY statement specifying a different location.

ENTRY Statement

The ENTRY statement specifies the symbolic name of the first instruction to be executed when the program is called by its module name for execution. An ENTRY statement should be used whenever a module is reprocessed by the linkage editor. If more than one ENTRY statement is encountered, the first statement specifies the main entry point; all other ENTRY statements are ignored. Its format is:

Operation	Operand
ENTRY	externalname

where externalname is defined as either a control section name or an entry name in a linkage editor input module.

ORDER Statement

The ORDER statement specifies the sequence in which control sections or named common areas are to appear in the output load module. When multiple ORDER statements are used, their sequence further determines the sequence of the control sections or named common areas in the output load module; those named on the first statement appear first, and so forth. Its format is:

Operation	Operand
ORDER	{common-area-name / csectname} [(P)] [
	{common-area-name / csectname} [(P)]] ...

where common-area-name is the name of the common area to be sequenced, csectname is the name of the control section to be sequenced, and P specifies that the starting address of the control section or named common area is to be on a page boundary within the load module. When P is specified, the control sections or common areas are aligned on 4K page boundaries unless the ALIGN2 attribute is specified on the EXEC statement.

PAGE Statement

The PAGE statement aligns a control section or named common area on a 4K page boundary in the load module. For OS/VS2, if the ALIGN2 attribute is specified on the EXEC statement for the linkage editor job step, use of the PAGE statement aligns the specified control sections or common areas on 2K page boundaries within the load module. Its format is:

Operation	Operand
PAGE	{common-area-name / csectname} [
	{common-area-name / csectname}] ...

where common-area-name is the name of the common area to be aligned on a page boundary, and csectname is the name of the control section to be aligned on a page boundary.

PROGRAMS COMPILED WITH THE DYNAM AND/OR RESIDENT OPTIONS

In the usual called/calling situation, all references to any subprogram or library subroutines generated in an object program result in a V-type address constant (VCON) that must be resolved by the linkage editor. Therefore, at link-edit time, the modules referred to by VCONs are made a part of a single load module containing the object program and all required subprograms and library routines. When the object program is executed, all those required routines are present in the user region for the entire execution step, even though they may have been used only at the beginning of the main program and never invoked again. With dynamic linkage, on the other hand, the user can invoke a called program when it is needed and retain it for only the period needed.

Subprograms invoked through the CALL literal statement are dynamically loaded using the Operating System LOAD macro if DYNAM is specified. Before the CALL Subprogram is executed, linkage is effected for all COBOL library subroutines required by the subprogram. Similarly, use of the CANCEL statement makes it possible to dynamically delete subprograms at object time.

Figure 122 is an example of a job compiled with the DYNAM and RESIDENT options. Figures 128 through 131 in this section illustrate for called/calling programs the relationship between the possible combinations of the DYNAM/RESIDENT options and the identifier and literal options of the CALL and CANCEL statements. Figure 131 shows the JCL necessary for compiling, link-editing, and executing a calling COBOL program and a called COBOL program when both of the programs invoke the DYNAM and RESIDENT compiler options.

When a program is compiled with DYNAM and RESIDENT, no external references are generated. Therefore, while the program may refer to other modules, no references are resolved by the linkage editor. In such a case, the only input to the linkage editor is the program itself. Any module the program refers to must exist in load module form in a library that is available to the system at execution time.

The link-editing that takes place varies with the combinations of the DYNAM(NODYNAM) and RESIDENT(NORESIDENT) options in effect. What would seem to be the most representative link-edit situations are discussed in the sections that follow.

Specifying DYNAM/RESIDENT

When both DYNAM and RESIDENT are specified for the called/calling situation pictured in Figure 128, first the main program COBA is compiled and link-edited; then each of the two subprograms COBB and COBC is compiled and link-edited separately, thereby producing three separate modules. Then the main program is executed.

In this situation, all external references are dynamically resolved. Therefore, no VCONs are generated for the address of an external symbol that would be used in a static situation (that is, a CALL literal without the DYNAM option) to effect branches to other programs.

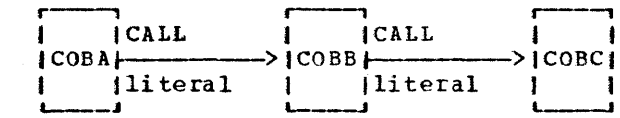


Figure 128. CALL with DYNAM and RESIDENT

Specifying NODYNAM/RESIDENT

When NODYNAM and RESIDENT are specified for the called/calling situation pictured in Figure 129, a dynamic situation occurs because of the inclusion of CALL identifier in the calling programs. That is, because the name of the called subprogram is not available until execution time, a CALL identifier statement cannot be used in a static situation.

In Figure 129, if NORESIDENT had been either specified or implied by default, it would have been overridden with RESIDENT. The compiler automatically recognizes the requirement for the library management feature by the presence of either a CALL identifier or CANCEL identifier in the source program.

Note: A printed indication of the compiler options in effect appears in the statistics section of the compiler output. (For examples of compiler statistics, see the chapter entitled "Output.")

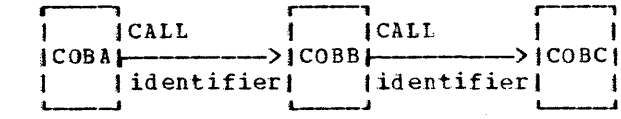


Figure 129. CALL With NODYNAM and RESIDENT

In contrast with Figure 128, the called/calling situation pictured in Figure 130 invokes the CALL literal option. Again the programs are compiled in the order COBA, COBB, and COBC. The CALL literal statements included in programs COBA and COBB result in static calls that must be resolved by the linkage editor. Therefore, the three program units are enjoined as a single module. However, with the COBOL Library Management Feature in effect, linkage to the library is dynamic. That is, the required COBOL object-time library subroutines are not link-edited, but linkage is effected dynamically at object time.

Note: When including both dynamic and static CALL statements in the same run unit, the programmer should not dynamically

call any subprograms that are otherwise called statically. To do so might cause multiple copies of the called program to be created and, therefore, produce unpredictable results.

The combination of RESIDENT and NODYNAM should be used only if required library subroutines are in fact permanently resident, or if all calls to COBOL subprograms are dynamic. If the library subroutines are not permanently resident, they will be loaded into the region or partition during program execution. This could cause storage fragmentation if a static call is made to a COBOL subprogram, because the subroutines required by a subprogram can only be removed from the region by a CANCEL statement for the subprogram, and a CANCEL is invalid for a subprogram that is statically called.

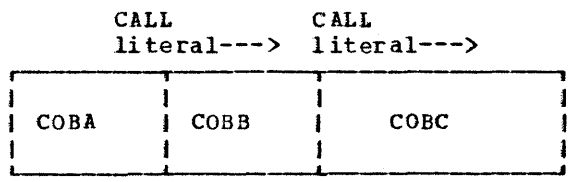


Figure 130. CALL With NODYNAM and RESIDENT With CALL Literal Option

Management Feature is not in effect, and all CALL statements result in static calls that must be resolved by the linkage editor. One load module is produced for the programs COBA, COBB, COBC, and all of the necessary COBOL library subroutines.

The NODYNAM/NORESIDENT set of options should be used only when the user does not intend to use the CALL or CANCEL identifier statement or the Library Management Feature. If either a CALL identifier or a CANCEL identifier statement appears in any one program, the Library Management Feature is in effect for that program only. This situation may result in a duplication of subprograms and COBOL library subroutines within the user region/partition, thereby causing unpredictable results.

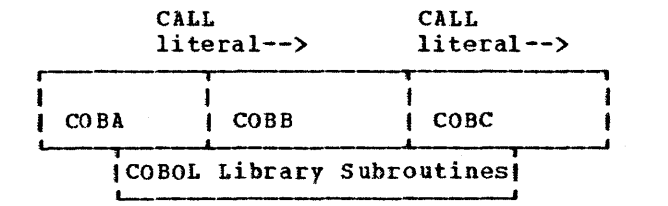


Figure 131. CALL With NODYNAM and NONRESIDENT

Specifying NODYNAM/NORESIDENT

For the called/calling situation pictured in Figure 131, the COBOL Library

```
//JOBY        JOB
//STEP1       EXEC   PGM=IKFCBL00,PARM='LOAD,DYNAM,RESIDENT'
              .
              .
              .
//SYSLIN      DD     DSNAME=&&LINKDS1,DISP=(MOD,PASS),UNIT=SYSSQ,SPACE=(TRK,(10,10))
//SYSIN       DD     *

              {Source module for COBMAIN, a calling COBOL program}

              CALL 'COBSUB'
/*
//STEP2       EXEC   PGM=HEWL
//SYSLMOD     DD     DSNAME=&&GOFILE,DISP=(MOD,PASS),UNIT=SYSSQ,SPACE=(TRK,(10,10,10))
//SYSLIN      DD     DSNAME=&&LINKDS1,DISP=(OLD,DELETE)
//SYSIN       DD     *
  NAME        COBMAIN
/*
//STEP3       EXEC   PGM=IKFCBL00,PARM='LOAD,DYNAM,RESIDENT'
//SYSLIN      DD     DSNAME=&&LINKDS2,DISP=(MOD,PASS),UNIT=SYSSQ,SPACE=(TRK,(10,10))
//SYSIN       DD     *

              {Source module for COBSUB, a called COBOL program}
/*
//STEP4       EXEC   PGM=HEWL
//SYSLMOD     DD     DSNAME=&&GOFILE,DISP=(OLD,PASS)
//SYSLIN      DD     DSNAME=&&LINKDS2,DISP=(OLD,DELETE)
//SYSIN       DD     *
  NAME        COBSUB
/*
//STEP5       EXEC   PGM=COBMAIN
//STEPLIB     DD     DSNAME=&&GOFILE,DISP=(OLD,DELETE)
/*
```

Figure 132. Sample JCL for Called/Calling Programs Compiled with the DYNAM and RESIDENT Options

```
//CALLPROG    JOB
//STEP1       EXEC    PGM=IKFCBL00,PARM=(LOAD,NODECK)
              .
              .
              .
//SYSLIN      DD      DSN=&&TEMPLIB1,UNIT=SYSSQ,DISP=(NEW,PASS),     X
//            SPACE=(TRK,(10,1))
//SYSIN       DD      *
            (Source module for COBSUB, a called COBOL program)
/*
//STEP2       EXEC    PGM=ASMBLR,PARM=(LOAD,NODECK),                 X
//            COND=(9,LT,STEP1)[1]
//SYSGO       DD      DSN=&&TEMPLIB1,DISP=(MOD,PASS)
//SYSIN       DD      *
            (Source module for ASSMMAIN, a calling assembler-
              language program)
/*
//STEP3       EXEC    PGM=HEWL,PARM=(LIST,XREF,LET),                 X
//            COND=((9,LT,STEP1),(5,LT,STEP2))
              .
              .
              .
//PROGLIB1    DD      DSN=&&TEMPLIB1,DISP=(OLD,PASS)
//SYSLIN      DD      *
              INCLUDE PROGLIB1[2]
              ENTRY   ASSMMAIN[3]
/*
//STEP4       EXEC    PGM=*.STEP3.SYSLMOD,COND=((9,LT,STEP1),       X
//            (5,LT,STEP2),(5,LT,STEP3))
//SYSOUT      DD      SYSOUT=A
```

[1] This example was chosen to illustrate the testing of condition codes.
[2] See the discussion under the INCLUDE statement.
[3] Because the COBOL program is compiled first and the linkage editor cannot identify the proper entry point, the ENTRY statement must be included.

Figure 133. Sample JCL Used for a Calling Assembler-Language Program and a Called COBOL Program

LINKAGE EDITOR PROCESSING

The linkage editor first processes the primary input and any additional input specified by INCLUDE statements. All external references in the primary that refer only to other modules in the included input are resolved first. If there are still unresolved references after this input is processed, the automatic call library, which includes libraries specified by the SYSLIB DD statement and by the LIBRARY statements, is searched to resolve the references. The automatic call library generally will contain the COBOL library subroutines. (External references to these subroutines are generated by the COBOL compiler when statements in the source module require certain functions to be performed, such as some data conversions.)

If the additional input contains external references and/or linkage editor control statements, the references are resolved in the same way. Data sets specified by the INCLUDE statement are incorporated when the statement is encountered. Data sets specified by the LIBRARY statement are used only when there are unresolved references after all of the other processing is completed.

```
//JOBX      JOB
//STEP1     EXEC    PGM=IKFCBL00,PARM=LOAD
            .
            .
            .

//SYSLIN    DD      DSNAME=&&GOFILE,DISP=(MOD,PASS),UNIT=SYSSQ,SPACE=(TRK,(10,10))
//SYSIN     DD      *

            (Source module for COBMAIN)

/*
//STEP2     EXEC    PGM=IKFCBL00,PARM=LOAD
            .
            .
            .

//SYSLIN    DD      DSNAME=*.STEP1.SYSLIN,DISP=(MOD,PASS)
//SYSIN     DD      *

            (Source module for COBOL1)

/*
//STEP3     EXEC    PGM=IKFCBL00,PARM=LOAD
            .
            .
            .

//SYSLIN    DD      DSNAME=*.STEP2.SYSLIN,DISP=(MOD,PASS)
//SYSIN     DD      *

            (Source module for COBOL1A)

/*
//STEP4     EXEC    PGM=IEWL
            .
            .
            .

//SYSLIB    DD      DSNAME=SYS1.COBLIB,DISP=OLD
//SYSLMOD   DD      DSNAME=PGMLIB(CALPGM),DISP=(NEW,KEEP),UNIT=3340,X
//                  SPACE=(1024,(50,20,2)),VOLUME=SER=LIBPAK
//OBLIB     DD      DSNAME=OBJLIB,DISP=OLD
//ADDLIB    DD      DSNAME=MYLIB,DISP=OLD
//SYSLIN    DD      DSNAME=&&GOFILE,DISP=(OLD,DELETE)                X
//          DD      *
            INCLUDE OBLIB(COBOL1B,ASSMPGM)
            LIBRARY ADDLIB(COBREGN0)
/*
```

Figure 134. Specifying Primary and Additional Input to the Linkage Editor

Example of Linkage Editor Processing

Figure 134 shows the control statements for a job that separately compiles three source modules (one is a calling program and two are called programs) and places them in one data set as primary input for the linkage editor. The linkage editor then links them together with additional input (called programs that are members of the specified libraries) to form one load module.

STEP1 compiles a source module called COBMAIN, STEP2 compiles a source module called COBOL1, and STEP3 compiles a source module called COBOL1A. The object module from each step is placed in the sequential data set called &&GOFILE. (Since MOD and PASS are specified for &&GOFILE in the SYSLIN DD statement in STEP1, the object modules COBOL1 and COBOL1A are placed in the data set behind the object module COBMAIN. When SYSSQ is not a mass storage device, the SPACE parameter is ignored.)

In STEP4, the linkage editor uses the &&GOFILE data set as primary input, and thecataloged libraries MYLIB, OBJLIB, and SYS1.COBLIB as additional input. (The INCLUDE and LIBRARY statements become part of the primary input through the DD * statement following the SYSLIN DD statement.

The object modules of the data set &&GOFILE and the members COBOL1B and ASSMPGM of OBJLIB are processed first. If there are unresolved references after this input is processed, the linkage editor searches the automatic call library, which includes the COBOL subroutine library and member COBREGN0 of MYLIB, to resolve these references. OBJLIB is specified in the OBLIB DD statement and MYLIB in the ADDLIB DD statement.

After linkage editor processing is completed, a new library, PGMLIB, is created with CALPGM as a member. CALPGM contains COBMAIN, COBOL1, COBOL1A, COBOL1B, ASSMPGM, and, possibly, COBOL subroutines and COBREGN0.

OVERLAY STRUCTURES

If it is necessary to conserve main storage, it can be accomplished by applying the overlay technique to called and calling programs. Called programs that do not need to be in main storage at the same time can be given the same relative storage address and then loaded at different times during execution when they are needed. In this way, the same storage space can be used for more than one called program.

Note: The use of execution-time debugging aids (SYMDMP, STATE, FLOW, COUNT, and so forth) when utilizing overlay structures is not recommended. Doing so may result in unpredictable operations within the STAE and debugging routines, or even abends.

Considerations for Overlay

Assume that the six programs illustrated in Figure 125 have the following load module sizes:

Program	Module Size (in Bytes)
COBMAIN	11,000
COBOL1	4,000
COBOL1A	6,000
COBOL1B	5,000
COBREGN0	3,000
ASSMPGM	13,000

Through the linkage mechanism, CALL COBOL1..., all subprograms plus COBMAIN must be link-edited together to form one module 42,000 bytes in size. Therefore, COBMAIN would require 42,000 bytes of storage in order to be executed.

If the subprograms needed do not fit into main storage, the following two techniques of overlay are available to the COBOL programmer:

- Preplanned overlay using the linkage editor
- Dynamic overlay using assembler language macro instructions during execution

Note: The largest load module that can be processed under OS/VS1 is 524,288 bytes. If a load module exceeds this limit, it should be divided. Under OS/VS2, larger load modules are permitted. The ON OVERFLOW phrase should be used with the CALL statement to handle any such size errors.

Linkage Editing with Preplanned Overlay

The preplanned linkage editor facility permits the reuse of storage locations already occupied. By judiciously modularizing a program and using the linkage editor overlay facility, a program that is too large to fit into storage at one time can be executed.

In using the preplanned overlay technique, the programmer specifies to the linkage editor which subprograms are to overlay each other. The subprograms specified are processed as part of the program by the linkage editor, so they can be automatically placed in main storage for execution when requested by the program. The resulting output of the linkage editor is called an overlay structure.

It is possible, at linkage edit time, to set up an overlay structure by using the COBOL source language linkage statement and the linkage editor OVERLAY statement. These statements enable a user to call a subprogram that is not actually in storage. The details for setting up the linkage editor control statements for accomplishing this procedure can be found in the publication OS/VS Linkage Editor and Loader.

In a linkage editor run, the programmer specifies the overlay points in a program by using OVERLAY statements. The linkage editor treats the entire input as one program, resolving all symbols and inserting tables into the program. These tables are used by the control program to bring the overlay subprograms into storage automatically when called.

Figure 135 is an overlay tree structure illustrating how the six programs in Figure

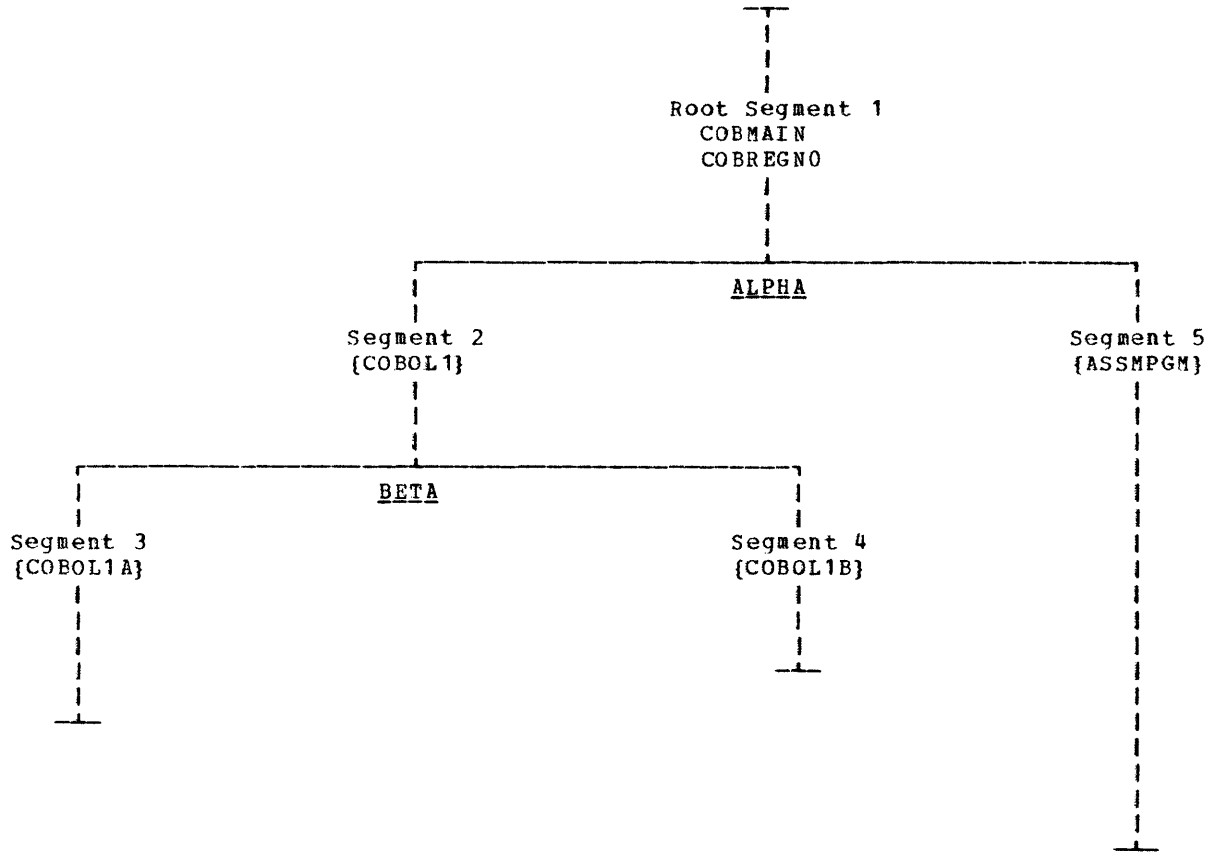


Figure 135. Overlay Tree Structure

126 could be positioned in main storage at execution time using preplanned linkage editor overlay.

Figure 136 shows the deck arrangement required to achieve the overlay illustrated in Figure 135. The OVERLAY statements specify to the linkage editor that the overlay structure to be established is one in which the called programs of COBOL1 (COBOL1A and COBOL1B) overlay each other when called for execution, and that ASSMPGM and COBOL1 and its called program overlay each other when called.

Routine COBREGN0 is placed with COBMAIN in the root segment of the overlay structure because it is called by three of the routines in the program, the largest of which is ASSMPGM. Utilizing COBREGN0 as an individual overlay segment would not have resulted in a net decrease in the amount of main storage required for execution because the minimum amount of main storage needed would have to contain COBMAIN, ASSMPGM, and COBREGN0 at the same time. Creating another overlay segment for COBREGN0 would only have added to the amount of time required for program execution.

Dynamic Overlay Technique

In preparation for the dynamic overlay technique, each part of the program brought into storage independently should be processed separately by the linkage editor. (Hence, each part must be processed as a separate load module.) To execute the entire program, the programmer must:

1. Specify the main program in the EXEC statement.
2. Bring the separately processed load modules into storage when they are required, by using the appropriate supervisor linkage macro instructions. This is accomplished during execution.

```
//OVERLAY     JOB         NY83937800,COSMO,MSGLEVEL=1
//STEP1       EXEC        PGM=IEWL,PARM='OVLY,LIST,XREF,LET'
//SYSLIB      DD          DSNAME=SYS1.COBLIB,DISP=SHR
//SYSPRINT    DD          SYSOUT=A
//SYSUT1      DD          UNIT=SYSDA,SPACE=(1024,(50,20))
//SYSLMOD     DD          DSNAME=&GODATA(RUN),DISP=(NEW,PASS),UNIT=SYSDA, X
//                        SPACE=(1024,(50,20,1))
//SYSLIN      DD          *

         {COBMAIN         object deck}
         {COBREGNO        object deck}

OVERLAY  ALPHA
         {COBOL1          object deck}

OVERLAY  BETA
         {COBOL1A         object deck}

OVERLAY  BETA
         {COBOL1B         object deck}

OVERLAY  ALPHA
         {ASSMPGM         object deck}
/*
```

Figure 136. Sample Deck for Linkage-Editor Overlay Structure

The dynamic overlay technique can be used to overlay subprograms during execution. To accomplish dynamic overlay of subprograms, the programmer must write an assembler language subprogram that employs the LINK macro instruction to call each COBOL subprogram. For a detailed description of the LINK macro instruction, see the publication OS/VS Supervisor Services and Macro Instructions.

In using the dynamic overlay technique, the main program communicates with the assembler language subprogram by using the COBOL language CALL statement. The CALL statement can be used to pass the name of the COBOL subprogram (to be linked) and the specified parameter list to the assembler language subprogram. This procedure is the same for each CALL used in the main program. Hence, each CALL results in linking with a subprogram through the assembler language subprogram.

When the COBOL subprogram is finished executing, it returns control to the assembler language subprogram, which in turn returns to the main program. The process is repeated for each CALL to the assembler-language subprogram.

Dynamic overlay requires that a programmer have detailed knowledge of the linkage conventions, assembler language, and the LINK macro instruction with its features and restrictions.

Figure 137 contains an example of a COBOL main program, PROGMAST, and an assembler language subprogram, LINKRTN. The two programs are link-edited together as a single load module. At execution time, the assembler-language subprogram dynamically fetches COBOL subprograms (OPN, BILL, CRDT, TRNF, and LCK, none of which are shown in the example) for the main program using the LINK macro instruction. The COBOL subprograms are stored in a private library, DYNLINK.

The parameter list passed to LINKRTN contains three identifiers, TRANS-REC, COM-WORD, and SWITCH, two of which (TRANS-REC and SWITCH) are referenced by LINKRTN, and two of which (TRANS-REC and COM-WORD) are referenced by the COBOL subprograms fetched. LINKRTN passes the same parameter list it receives to the COBOL subprograms fetched.

LINKRTN determines from identifier TRANS-REC which subprograms to fetch, and from SWITCH when to open and close the library DYNLINK.

Note: In structuring a program with either the preplanned overlay technique or the dynamic overlay technique, special consideration must be given to the presence of the TRANSFORM table and the class test tables, which are members of the COBOL object-time library (see "Appendix B: COBOL Library Subroutines"). The TRANSFORM table is link-edited with a COBOL program if the TRANSFORM statement is used. Similarly, one or more of the class test

tables is present in a COBOL load module if a class test is performed or if the OCCURS DEPENDING ON option is used.

For these tables, which contain no executable code and are not branched to but are merely referenced, the compiler designates A-type address constants (ADCONs) and EXTRN references, rather than V-type address constants (VCONs). Accordingly, the overlay structure segment containing the table(s) must be either the root segment or a segment that is higher in the same leg as the segment containing the reference(s) to the table(s).

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  PROGMAST.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.  IBM-370.
OBJECT-COMPUTER.  IBM-370.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT FILE-Y ASSIGN TO UR-2540R-S-INFILE.
I-O-CONTROL.
DATA DIVISION.
FILE SECTION.
FD  FILE-Y
    RECORD CONTAINS 80 CHARACTERS
    LABEL RECORD IS OMITTED.
01  TRANS-REC.
    05  ACCOUNT-NUMBER PIC 9(10).
    05  TRANSACTION PIC 9(4).
    05  NAME PIC X(20).
    05  LOCATION PIC X(20).
    05  METER-READING PIC 9(6).
    05  DAYTE PIC 9(6).
    05  FILLER PIC X(8).
    05  AMOUNT PIC 9(6).
WORKING-STORAGE SECTION.
77  COM-WORD PIC X(12).
77  SWITCH PIC 9 VALUE ZERO.
PROCEDURE DIVISION.
    .
    .
    .
    OPEN INPUT FILE-Y.
B.  READ FILE-Y AT END GO TO END-RUN.
C.  CALL 'GETUM' USING TRANS-REC COM-WORD SWITCH.
    .
    .
    .
END-RUN.  CLOSE FILE-Y.
    MOVE 2 TO SWITCH.
    PERFORM C.
    STOP RUN.
```

Figure 137. Sample COBOL Main Program and Assembler-Language Subprogram Using Dynamic Overlay Technique (Part 1 of 3)

```
LINKRTN   START 0
          PRINT NOGEN
          ENTRY GETUM                 UPON ENTRY TO THIS PROGRAM, REGISTER 1 POINTS
*                                     TO A FIXED-LENGTH PARAMETER LIST OF THREE
*                                       WORDS.
*                                       THE FIRST WORD CONTAINS THE ADDRESS OF
*                                         RECORD TRANS-REC.
*                                       THE SECOND WORD CONTAINS THE ADDRESS OF
*                                         COM-WORD, TO WHICH THIS PROGRAM DOES NOT
*                                         REFER BUT WHICH IS USED BY ROUTINES THIS
*                                         PROGRAM LATER LINKS TO.
*                                       THE THIRD WORD CONTAINS THE ADDRESS OF
*                                         SWITCH USED BY THIS PROGRAM TO CHECK THE
*                                         STATUS OF THE PRIVATE LIBRARY DYNLINK
          USING GETUM,15
GETUM     SAVE  (14,12)
          LR    10,15
          DROP  15
          USING GETUM,10
          LR    11,13
          LA    13,SAVEAREA
          ST    13,8(11)
          ST    11,4(13)
          L     5,0(1)                REGISTER 5 LOADED WITH ADDRESS OF TRANS-REC
          USING PARAMLST,5            REGTER 5 IS USED AS THE BASE REGISTER TO
*                                       REFERENCE TRANS-REC.
          B     OPENLIB
SAVEAREA  DS    18F

OPENLIB   L     6,8(1)                REGISTER 6 LOADED WITH ADDRESS OF SWITCH.
          CLI   0(6),C'1'             CHECK SWITCH STATUS.
          BE    INITREG               IF SWITCH = 1, DYNLINK IS ALREADY OPEN;
*                                       INITIALIZE REGISTERS.
          BH    CLOSLIB               IF SWITCH > 1, DYNLINK IS NO LONGER NEEDED;
*                                       CLOSE DYNLINK.
          OPEN  (DYNLINK)             IF SWITCH = 0 THE FIRST TIME THROUGH, OPEN
*                                       DYNLINK.
          OI    0(6),C'1'             SET SWITCH SO THAT OPEN IS BYPASSED ON FUTURE
*                                       ENTRY.
*  TABLE LOOK-UP ROUTINE

INITREG   LA    2,RTNLST              INITIALIZE REGISTERS 2 AND 3 FOR LOOK-UP.
          LA    3,6
FINDRTN   CLC   TRANSACT,0(2)         TRANSACT CONTAINS THE TRANSACTION CODE THAT
*                                       DETERMINES WHICH ROUTINE TO FETCH.
          BE    GETRTN
          LA    2,12(0,2)
          BCT   3,FINDRTN
          MVC   ERRMSG+28(4),TRANSACT PRODUCE ERROR MESSAGE IF TRANSACT CONTAINS
ERRMSG    WTO   'INVALID TRANSACTION'   AN INVALID CODE.
EXIT      L     13,4(13)
          SR    15,15                 SET REGISTER 15 TO ZERO.
          RETURN(14,12),T,RC=(15)     THE RC=(15) PARAMETER INDICATES THAT THIS
                                        PROGRAM IS PASSING A RETURN CODE IN REGISTER
                                        15.

*  DYNAMIC OVERLAY ROUTINE
GETRTN    L     1,24(11)              RESTORE REGISTER 1 TO ORIGINAL STATUS.
          LA    4,4(0,2)              PASS REGISTER 4 TO NAME OF ROUTINE TO BE
          LINK  EPLOC=(4),DCB=DYNLINK   FETCHED.  HAVE THE CONTROL PROGRAM
*                                       FETCH THE ROUTINE POINTED TO BY
*                                       REGISTER 4 FROM PRIVATE LIBRARY DYNLINK.
          B     EXIT
```

Figure 137. Sample COBOL Main Program and Assembler-Language Subprogram Using Dynamic Overlay Technique (Part 2 of 3)

```
CLOSLIB   CLOSE (DYNLINK)              CLOSE PRIVATE LIBRARY.
          B     EXIT

          DS    0F
RTNLST    EQU   *                      AS THE TABLE SEARCHED BY THE TABLE LOOK-UP
*                                        ROUTINE, RTNLST CONTAINS A LIST OF ALL VALID
*                                        TRANSACTION CODES AND THE NAMES OF THE
*                                         ROUTINES FETCHED TO HANDLE THE TRANSACTIONS

          DC    C'0100'                TRANSACTION CODE
          DC    CL8'OPN'               ROUTINE NAME ASSOCIATED WITH ABOVE TRANSACTION
          DC    C'0200'
          DC    CL8'BILL'
          DC    C'0300'
          DC    CL8'CRDT'
          DC    C'0400'
          DC    CL8'TRNF'
          DC    C'0500'
          DC    CL8'LCK'

DYNLINK   EQU   *
          DCB   DDNAME=SYNLNKDD,DSORG=PO,MACRF=(R)
*                                      DCB TO DEFINE PRIVATE LIBRARY REFERRED TO IN
*                                        LINK MACRO INSTRUCTION.

PARAMLST  DSECT                        DSECT USED BY REGISTER 5 TO REFER TO TRANS-
*                                        REC.  THE RECORD DESCRIPTION CORRESPONDS TO
*                                        THAT OF TRANS-REC IN PROGMAST.
TRANSREC  DS    0CL80
ACCTNUM   DS    CL10
TRANSACT  DS    CL4
NAME      DS    CL20
LOCATION  DS    CL20
METERRD   DS    CL6
DATE      DS    CL6
          DS    CL8
AMOUNT    DS    CL6
          END
```

Note: Had a job or step library (requiring either a JOBLIB or STEPLIB DD statement in the job control for execution of the main program) been used instead of a private library (which for this example requires a DD statement named DYNLNKDD), responsibility for the opening and closing of the library would have been with the control program and not with LNKRTN.

The use of a private library, in conjunction with the LINK DCB parameter, reduces to a minimum the amount of search time needed to retrieve member modules from a library.

Figure 137. Sample COBOL Main Program and Assembler-Language Subprogram Using Dynamic Overlay Technique (Part 3 of 3)

LOADING PROGRAMS

The loader resolves external names and references and combines calling and called programs into a format suitable for execution as a single load module. For information on invoking the loader, see "Using the Cataloged Procedures."

When the dynamic call is used, all subprograms to be called dynamically must have been processed by the linkage editor. The loader may be used only to resolve references to subprograms invoked by static calls. Otherwise, load modules of both calling and called programs are used as input to the loader. There are two kinds of input, primary and additional. Primary input consists of one or more separately compiled object modules and/or load modules. Additional input consists of object modules or load modules that are not part of primary input data sets but are to be included in the load module. The additional input may be in the form of (1) libraries containing object modules, or (2) libraries containing load modules. Additional input may contain either calling or called programs or both.

If these subroutines reside in the Link Pack Area, their external references are not resolved. When the RES loader option is specified, the loader always searches the Link Pack Area for modules before searching the SYSLIB data set. If the RES compiler option is specified, subroutine ILBONTR0 controls all subroutine intercommunication so that unresolved external references in the Link Pack Area present no problem. If, however, the NORES compiler option is specified, ILBONTR0 is not invoked and unresolved external references in the Link Pack Area may prevent successful subroutine execution. Therefore, in order to prevent the loader from searching the Link pack Area, the NORES loader option should be specified in conjunction with the NORES compiler option. This will cause the loader to search for required modules in the SYSLIB data set, bypassing the Link Pack Area search. If COBOL subroutines that have external references which would need to be resolved do not reside in the Link Pack Area, either the RES or NORES loader option is acceptable. (Subroutine external references are listed in Figure 174.)

SPECIFYING PRIMARY INPUT

The primary input data set is specified for loader processing by the SYSLIN DD statement. The loader must always have a primary input data set whether or not there are calling or called programs. The SYSLIN DD statement that specifies primary input is discussed in the section "Data Set Requirements."

SPECIFYING ADDITIONAL INPUT

Additional input data sets are specified by the SYSLIB DD statement. The SYSLIB DD statement is discussed in the section "Data Set Requirements."

Note: The overlay facility can not be used with the loader.

Libraries are an integral part of the operating system. Some libraries have system-supplied names and system-supplied data. Other libraries have system-supplied names but may contain user-specified data. Still other libraries have both user-supplied names and user-supplied data.

Libraries, in general, are made up of partitioned data sets. Any library with a user-supplied name and user-supplied data is always a single partitioned data set, which is a collection of independent sets of sequentially organized data, called members. All of the members within a partitioned data set have the same characteristics as that of record format. When used to store programs, a partitioned data set containing load modules can contain only load modules; it cannot contain both load modules and object modules.

Each partitioned data set is headed by a directory of entries pointing to the members that make up the library. Each member has a unique member name. A partitioned data set must reside on a single mass storage device, but some libraries can consist of a concatenation of more than one partitioned data set.

Figure 138 shows the format of a library that is a single partitioned data set of four members. Space for the members of such a library and its directory is requested in the SPACE parameter of the DD statement when the library is created. Additional members can be added to a library at a later time. Additional space cannot be allocated for the directory, however. Directory space is allocated for the entire library when the library is created. If the original allocation was not large enough, the IEHMOVE utility program can be used to expand the directory size. If the directory is filled, no additional members can be added to the library. Following is an example of a DD statement that might be used to create a library:

```
//DD1       DD DSNAME=FILELIB(FILE1),      X
//               DISP=(NEW,CATLG),          X
//               UNIT=2314,                 X
//               SPACE=(TRK,(40,10,3)),     X
//               VOLUME=SER=111111
```

This statement specifies that a library named FILELIB is to be created and cataloged in this job step. Its first member is named FILE1. Initial space allocated for data sets is to be 40 tracks, with additional allocation to be made, as necessary, in units of 10 tracks. In addition, space for three 256-byte records is to be allocated for the directory. The volume serial number is 111111.

A member of a partitioned data set can be replaced or deleted. The system actually accomplishes this by modifying or deleting the directory pointer to the member. The space occupied by the original member is not available for reuse either until the MOVE or COPY control statement of the IEHMOVE utility program is used or the compress facility of the IEBCOPY utility program is used. The space previously occupied by the replaced or deleted member is thus made available. (For further details, see the publication OS/VS Utilities.)

KINDS OF LIBRARIES

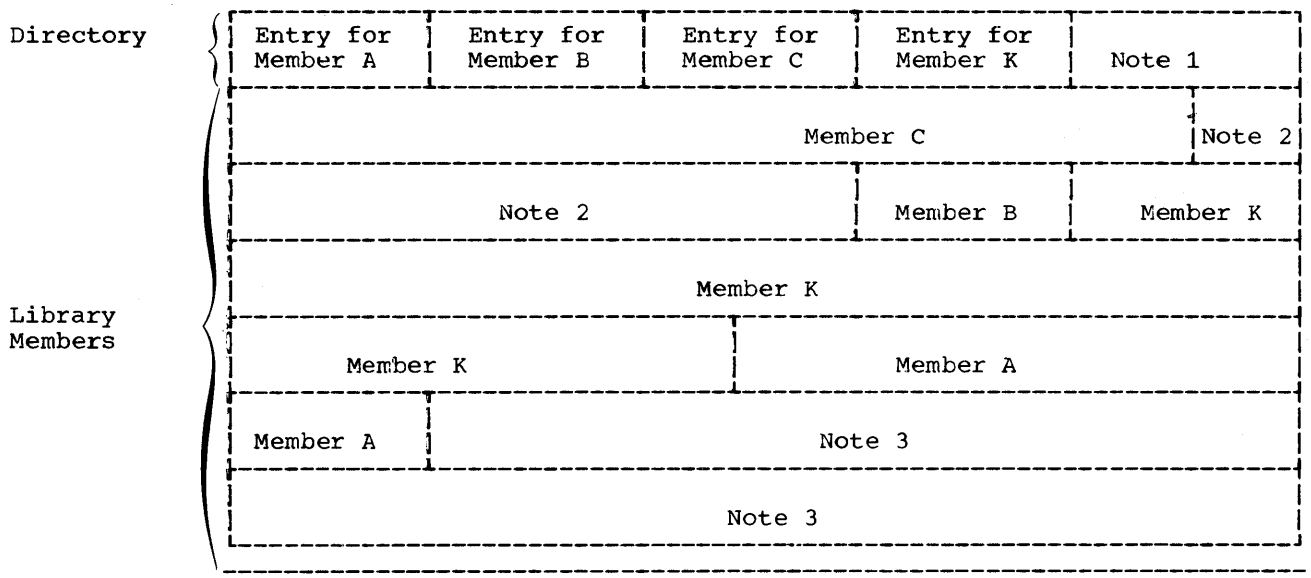
A programmer can use libraries already provided by the system, or he can create libraries of his own. In addition, certain library names recognized by the system may be assigned to partitioned data sets provided by the system, by the programmer, or both. These libraries and their uses are discussed in the following paragraphs.

SYSTEM LIBRARIES USED IN COBOL APPLICATIONS

Link Library

The link library is a partitioned data set that contains load modules to be executed. Unless non-resident control program routines and IBM-supplied processing programs specified otherwise, a load module name in an EXEC statement is to be fetched from the link library. Operating system programs, such as the COBOL compiler, are usually contained in this library.

The link library can be used by the programmer to store executable load modules at link-edit time. However, any attempt to



Notes:
1. Space available in directory.
2. Space available from deleted members. Space can be recovered through utility programs IEHMOVE and IEBCOPY.
3. Space available in library.

Figure 138. Format of a Library

write in this library will cause a message to be issued to the operator if the library is write-protected or if its expiration or retention date has not yet occurred. Under these conditions, no data may be written unless authorized.

The link library is identified in a job control statement as SYS1.LINKLIB, or by default as specified by the system link list.

Procedure Library

The procedure library is a partitioned data set whose members are the cataloged procedures at an installation. They may include the cataloged procedures provided by IBM. Procedures written at the installation can be added to the procedure library with the IEBUPDTE utility program (see "Using the Cataloged Procedures").

The system name for the procedure library is SYS1.PROCLIB.

Sort Library

The sort library is a partitioned data set that contains load modules from which the sort program is produced.

It may be identified by the name SYS1.SORTLIB (see "Using the Sort/Merge Feature").

COBOL Subroutine Library

The COBOL subroutine library is a partitioned data set that contains the COBOL library subroutines in load module form. These subroutines may be included in a COBOL load module or dynamically loaded to perform such functions as data conversion and double precision arithmetic. The COBOL programmer does not refer directly to these subroutines; in most cases calling sequences to them are generated at compile time from certain Procedure Division statements, and they are incorporated into the load module at link-edit time or loaded at program initialization time. A listing of subroutine names, functions, entry points, and size is given in Appendix B; also noted are those subroutines explicitly called by the COBOL program.

Availability at Execution-Time: Usually, COBOL library subroutines needed in the user's program will be automatically link-edited into the load module, and the user need not concern himself with their availability. However, certain situations will require that subroutines not normally linked be available during program execution. To satisfy this requirement, the user can either make sure that the COBOL library is available to the system loader at execution time, or specifically include the subroutines in his load module--by using an INCLUDE SYSLIB (subroutine names) in the link edit SYSLIN data stream.

The situations, and the subroutines required, are as follows:

1. STATE compiler option was specified. Subroutine ILBOSTN0 will be required if an abend occurs.

2. TEST compiler option was specified. ILBOCOM0 will be required.

3. COUNT compiler option was specified. ILBOTC2 and ILBOTC3 will be required.

4. SYMDMP compiler option was specified. ILBOD01, ILBOD10 through ILBOD14, and ILBOD20 through ILBOD26 will be required.

5. COUNT, FLOW, STATE, or SYMDMP compiler option was specified. If an informative message needs to be issued, ILBODBE0 will be required.

6. Execution-time options (PARM field on EXEC statement) will be passed. ILBOPRM0 will be required.

In the event the DYNAMIC or RESIDENT compiler option was specified, then all subroutines will be loaded (and required to be available) at execution time.

The system name for the COBOL subroutine library may be SYS1.COBLIB.

LIBRARIES CREATED BY THE USER

A programmer can create members of the link library, the procedure library, and the job library. He can also create partitioned data sets for use in the copy library, the automatic call library, and the job library. In addition, he can create partitioned data sets to be used as libraries for additional input to the linkage editor, and he can create libraries whose members are source program entries.

Automatic Call Library

The automatic call library, defined by the SYSLIB DD statement in the link-edit job step, contains load modules or object modules that may be used as secondary input to the linkage editor. If the library contains object modules, it may also contain control statements. External symbols that are undefined after all primary input has been processed cause the automatic library call mechanism to search the automatic call library for modules that will resolve the references, unless the NCAL option is specified. The COBOL subroutine library must be specified for the automatic call library if any of the subroutines will be needed to resolve external references. Other partitioned data sets may be concatenated as shown in the following example:

```
//SYSLIB   DD DSNAME=SYS1.COBLIB,DISP=SHR
//         DD DSNAME=MYLIB,DISP=SHR
```

In this case, both the COBOL subroutine library and the partitioned data set named MYLIB are available to the automatic library call.

```
//CATALOG          JOB    user information
//                 EXEC   PGM=IEBUPDTE,PARM=(NEW)
//SYSUT2           DD     DSNAME=COPYLIB,UNIT=3330,                    X
//                            DISP=(NEW,KEEP),                        X
//                            VOLUME=SER=111111,                      X
//                            SPACE=(TRK,(15,10,2)),                  X
//                       DCB=(LRECL=80,BLKSIZE=80,RECFM=F)
//SYSPRINT         DD     SYSOUT=A
//SYSIN            DD     *
./                 ADD    NAME=CFILEA,LEVEL=00,SOURCE=0,LIST=ALL
./                 NUMBER NEW1=10,INCR=5
                     BLOCK CONTAINS 13 RECORDS
                     RECORD CONTAINS 120 CHARACTERS
                     LABEL RECORDS ARE STANDARD
                     DATA RECORD IS FILE-OUT.
./                 ENDUP
/*
```

Figure 139. Entering Source Statements into the COPY Library

```
//UPDATE           JOB    user information
//                 EXEC   PGM=IEBUPDTE,PARM=(MOD)
//SYSUT1           DD     DSNAME=COPYLIB,UNIT=3340,                    X
//                            DISP=(OLD,KEEP),                        X
//                            VOLUME=SER=111111,                      X
//                            DCB=(RECFM=F,BLKSIZE=80)
//SYSUT2           DD     DSNAME=COPYLIB,UNIT=3340,                    X
//                            DISP=(OLD,KEEP),                        X
//                            VOLUME=SER=111111                       X
//SYSPRINT         DD     SYSOUT=A
//SYSIN            DD     *
./                 CHANGE NAME=CFILEA,LEVEL=01,SOURCE=0,LIST=ALL
                     BLOCK CONTAINS 20 RECORDS                        00000010
./                 ENDUP
/*
```

Figure 140. Updating Source Statements in a COPY Library

Note: If the partitioned data set named in the SYSLIB DD statement contains load modules, any data set concatenated with it must also be a load module partitioned data set. If the first contains object modules, the others must also contain object modules.

The linkage editor LIBRARY control statement has the effect of concatenating any specified member names with the automatic call library.

COBOL Copy Library

The COBOL copy library is a user-created library consisting of statements or entire COBOL programs frequently used by the programmer. The programmer can include these statements or programs into a program at compile time. He calls them with the COBOL COPY statement or BASIS card.

To enter or update source statements in the copy library, a utility program must be used. IEBUPDTE is the IBM-supplied utility program used to catalog procedures. A full discussion of the statements used in this program may be found in the publication OS/VS Utilities.

Maximum Block Size:

The maximum block size for the copy library is restricted to 16K.

Multiple Libraries: If more than one copy library is being used, the COPY statement must include the phrase IN/OF library-name, where library-name is the ddname

identifying the particular partitioned data set to be copied from. (If this qualifying phrase is omitted, the default ddname becomes SYSLIB or its alternate.)

Entering Source Statements: Figure 139 illustrates the method to insert source statements into a copy library member.

The ./ ADD statement is a utility statement that copies CFILEA into the library called COPYLIB. CFILEA describes an FD entry. The NUMBER statement assigns a sequential numbering system to the statements in the library. The first statement is assigned number 000010 and each succeeding statement is incremented by 5. The entries following the utility statements are the actual source statements to be stored. The ENDUP statement signals the end of the entries to be inserted.

The same procedure can be used to store entire source programs.

Updating Source Statements: Figure 140 illustrates the method to update source statements in a copy library member inserted in the previous example.

SYSUT1 and SYSUT2 describe the data sets. Note that changes may be made on the same data set (identified on the DSNAME parameter). The utility statement CHANGE indicates that the new entry of CFILEA replaces the old entry. Alternatively, the UPDATE IN PLACE parameter of the change statement could have been used to avoid moving the entire member, (CILEA, to the first available area in the file. The sequence number of the altered statement must be supplied. This number, 00000010, is indicated in columns 73 through 80 of the replacement source statement. Note that, although in the insert example (see Figure 139 -- NUMBER statement) the number was coded as 10 without leading zeros, the program assigns an 8-character field to a sequence number and pads with leading zeros if necessary. When updating a sequence number in a library, these leading zeros must be included.

At compile time, COPYLIB is identified on a DD statement; for example:

```
//SYSLIB   DD    DSNAME=COPYLIB,              X
//                VOLUME=SER=111111,          X
//                DISP=SHR,UNIT=2314
```

Retrieving Source Statements: Members of the cataloged library can be retrieved using the COPY statement or BASIS card.

COPY Statement

The COPY statement permits the programmer to include stored source statements in any of the four divisions. If the programmer wishes to retrieve the member, CFILEA, stored in the previous examples, he writes the statement:

```
FD FILEA COPY CFILEA
```

The compiler translates this instruction to read:

```
FD FILEA
   BLOCK CONTAINS 20 RECORDS
   RECORD CONTAINS 120 CHARACTERS
   LABEL RECORDS ARE STANDARD
   DATA RECORD IS FILE-OUT.
```

Note that CFILEA itself does not appear in the statement. CFILEA is a name ide tifying the entries. It acts as a header record but is not itself retrieved. The compiler source listing, however, will print out the COPY statement as the programmer wrote it.

The COPY statement permits the programmer to include previously stored source statements into any portion of the program.

Assume a procedure named DOWORK was stored with the following statements:

```
./  ADD          NAME=DOWORK,LEVEL=00,
                   SOURCE=0,LIST=ALL
./  NUMBER       SEQ1=400,INCR=10
      COMPUTE QTY-ON-HAND =
        TOTAL-USED-NUMBER-ON-HAND.
      MOVE-QTY-ON-HAND TO PRINT-AREA.
./  ENDUP
```

To retrieve the stored member, DOWORK, the programmer writes:

```
paragraph-name.  COPY DOWORK.
```

The statements included in the DOWORK procedure will immediately follow the paragraph-name, replacing the words COPY DOWORK.

Notes:

1. The SUPPRESS option of the COPY statement will be ignored if LISTER or FIPS is requested.
2. Results may be unpredictable if a CURRENCY SIGN IS = is specified (only allowed with LANGLVL(1)) and a PICTURE character string is part of pseudo-text and contains a floating currency sign.
3. In order for the text copied to have a D inserted in column 7 (debugging line indicator), the D must appear on the first line of the COPY statement itself. A copy statement itself can never be a debugging line; if it contains a D and WITH DEBUGGING mode is not specified, the COPY statement will nevertheless be processed.
4. No more than 150 COPY-REPLACING pairs may be specified in a source program. If this limit is exceeded, message IKFI20I is issued by the compiler, and COPY statements over the limit are ignored.

BASIS Card

Frequently used source programs, such as a payroll program, can be inserted into the copy library. The BASIS card brings in an entire source program at compile time. Calling in a program eliminates the need for the programmer to handle a program each time he wants to compile it. The programmer may, however, alter any statement in the source program by referring to its COBOL sequence number with an INSERT or DELETE statement. The INSERT statement will add new source statements after the sequence number indicated. The DELETE statement will eliminate the statements indicated by the sequence numbers. The programmer may delete a single statement with one sequence number, or he may delete more than one statement, separating by a hyphen the first and last sequence numbers to be deleted.

Note: The COBOL sequence number is the 6-digit number that the programmer assigns in columns 1 through 6 of the source cards. This sequence number has nothing to do with the sequence numbers assigned in simulated columns 73 through 80 by the IEBUPDTE utility program. The sequence numbers

```
COBOL                                                                    IEBUPDTE
Sequence                                                                 Sequence
Numbers                                                                  Numbers
000730            IF ANNUAL-PAY GREATER THAN 15,000 GO TO PAY-WRITE.     00000105
000735            IF ANNUAL-PAY GREATER THAN 15,000 - BASE-PAY GO TO LAST-TAX. 00000110
000740 TAX-PAYR.  COMPUTE TAX-PAY = BASE-PAY * .025                      00000115
000750            MOVE TAX-PAY TO OUTPUT-TAX.                            00000120
000760 PAY-WRITE. MOVE BASE-PAY TO OUTPUT-BASE.                          00000125
000770            ADD BASE-PAY TO ANNUAL-PAY.                            00000130
   .              .                                                        .
   .              .                                                        .
   .              .                                                        .
000850            STOP RUN.                                              00000240
```

Figure 141. COBOL Statements to Deduct Old Age Tax

```
   .
   .
   .
BASIS PAYROLL
DELETE 000730-000740
000730             IF ANNUAL-PAY GREATER THAN 17800 GO TO PAY-WRITE.
000735             IF ANNUAL-PAY GREATER THAN 17800 - BASE-PAY GO TO LAST-TAX.
000740 TAX-PAYR.  COMPUTE TAX-PAY = BASE-PAY * .044.
```

Figure 142. Programmer Changes to Source Program

assigned by IEBUPDTE are used to update source statements in the copy library. Changes made using these numbers are intended to be permanent changes. The COBOL sequence numbers are used to update COBOL source statements at compile time. Such changes are in effect for the one run only.

Assume that a company payroll program is kept as a source program in the copy library. The name of the program is PAYROLL. During a particular year, old age tax is taken out at a rate of two and a half percent each week for all personnel until earnings exceed $15,000. The coding to accomplish this is shown in Figure 141.

Now, however, due to a change in the old age tax laws, tax is to be taken out until earnings exceed $17,800 and a new percentage is to be placed. The programmer can code these changes as shown in Figure 142.

The altered program will contain the coding shown in Figure 143.

Note that changes made through use of the INSERT and DELETE statements remain in effect for the one run only.

Note: If both the COPY statement and the BASIS card are used, the library containing the member specified in the BASIS card must be defined first. The COPY libraries concatenated with the BASIS library may be defined and referenced in any order (see "Appendix I: Checklist for Job Control

```
000730            IF ANNUAL-PAY GREATER THAN 17800 GO TO PAY-WRITE.
000735            IF ANNUAL-PAY GREATER THAN 17800 - BASE-PAY GO TO LAST-TAX.
000740 TAX-PAYR.  COMPUTE TAX-PAY = BASE-PAY * .044.
000750            MOVE TAX-PAY TO OUTPUT-TAX.
000760 PAY-WRITE. MOVE BASE-PAY TO OUTPUT-BASE.
000770            ADD BASE-PAY TO ANNUAL-PAY.
   .              .
   .              .
   .              .
000850            STOP RUN.
```

Figure 143. Changed COBOL Statements to Source COPY Library Statements

Procedures"). For a discussion of special considerations when using BASIS with the BATCH option, see "Batch Compilation."

JOB Library

The job library consists of one or more partitioned data sets that contain load modules to be executed. It is specified by the JOBLIB DD statement that must precede the EXEC statement of the first step of a job. Partitioned data sets assigned to the job library are concatenated with the link library so that any load module is obtained automatically when its name appears in the PGM= parameter of the EXEC statement. The following statements illustrate how three cataloged partitioned data sets can be assigned to the job library:

```
//MYJOB   JOB ...
//JOBLIB  DD  DSNAME=MYLIB1,DISP=(OLD,PASS)
//        DD  DSNAME=MYLIB2,DISP=(OLD,PASS)
//        DD  DSNAME=MYLIB3,DISP=(OLD,PASS)
//STEP1   EXEC ...
          .
          .
          .
//STEP2   EXEC ...
          .
          .
          .
```

These statements specify that the job library containing the data sets MYLIB1, MYLIB2, and MYLIB3 is to be concatenated with the link library. When a load module is named in an EXEC statement in any step of the job, the directories of the job library will be searched for the name. When a job library is specified for a job, the link library is searched for a named load module only when the module is not found in the job library.

Partitioned data sets used in the job library can be created by specifying the partitioned data set name and the member name in the SYSLMOD DD statement when each member is processed by the linkage editor.

Additional Input to the Linkage Editor: Libraries of object modules (with or without linkage editor control statements) and libraries of load modules can be used as additional input to the linkage editor. Members are specified by use of the INCLUDE and LIBRARY linkage editor control statements.

A library of object modules and control statements can be created by use of the IEBUPDTE utility program.

A library of load modules can be created by use of the SYSLMOD DD statement in the linkage editor job step, as discussed in "Job Library."

SHARING COBOL LIBRARY SUBROUTINES

Use of the COBOL Library Management Feature makes it possible for all programs in the same or different regions/partitions to share one copy of the COBOL library subroutines. That is, the most economical use of main storage is made when the most frequently used COBOL library subroutines are placed in the OS/VS2 link pack area (LPA), or the OS/VS1 resident reenterable routine (RRR) area, rather than in each region/partition. To make the most effective use of the Library Management Feature, and to use the IBM cataloged procedures whether or not Library Management is needed, the user should concatenate the COBOL subroutine library with the system link library, or specify it to be used as such a library in the appropriate system parameter library member.

The user may request the COBOL Library Management Feature at compile time, via the RESIDENT option (see the section "Options for the Compiler" in the chapter entitled "Job Control Procedures").

CONCATENATING THE SUBROUTINE LIBRARY

To concatenate the subroutine library with the link library, the user executes the IEBUPDTE utility program to add a member named LNKLST00 to SYS1.PARMLIB, specifying the library desired (that is, either the entire COBOL subroutine library or a private library containing user-selected COBOL library subroutines). Note that the library containing the subroutines must be cataloged.

An installation that is planning to use the Library Management Feature will find it convenient to include frequently used COBOL library subroutines in the OS/VS2 LPA or the OS/VS1 RRR area. Infrequently used subroutines are then brought into the region/partition as required. To add COBOL subroutines to the RRR area, the user invokes the IEBUPDTE utility program to add a member named IEAIGGXX (see Note 2 in Figure 143) to SYS1.PARMLIB, specifying all names and aliases for the COBOL library subroutines to be included. Then, at an initial program load (IPL) time, the operator identifies the link list to the

system, which subsequently places the identified COBOL subroutines in main storage in the RRR area.

Figure 144 illustrates how an installation can accomplish both these functions in one operation. The encircled letters in the figure refer to the JCL suggested A to concatenate the COBOL subroutine library (SYS1.COBLIB) with the system link library (SYS1.LINKLIB), and then B to place the user list of desired COBOL library subroutines and their aliases to the RRR. (For further information, see the publication OS/VS COBOL Compiler and Library Installation Reference Material.)

Notes:

1. If the user does not wish to place any COBOL subroutines in the RRR area, he need not execute the portion of the IEBUPDTE utility program that adds IEAIGGXX to SYS1.PARMLIB shown above. He may still make use of the Library Management Feature. However, all required library subroutines will be loaded into his own region/partition when they are needed by one or more programs, and deleted when they are no longer needed. Thus, not all library subroutines needed by all programs in the region need be resident at the same time. In this case, however, the user must supply a job control card at execution time pointing to the COBOL subroutine library or to his own private library of COBOL subroutines. (For a discussion of the various COBOL library subroutines available to the programmer, see "Appendix B: COBOL Library Subroutines.")

2. If one or more programs in a given region/partition request the COBOL Library Management Feature, then the main program and all subprograms in that region/partition must use it. Otherwise, the multiple copies of COBOL library subroutines resident at one time may cause unpredictable results.

CREATING AND CHANGING LIBRARIES

A programmer can create or change a partitioned data set in one of three ways: (1) through the use of DD statements, (2) through the use of utility programs, and (3) through the use of certain linkage editor control statements.

```
//CATLG     JOB   user information
//          EXEC  PGM=IEBUPDTE,PARM=MOD
//SYSPRINT  DD    SYSOUT=A
//SYSUT1    DD    DSN=SYS1.PARMLIB,DISP=SHR
//SYSUT2    DD    DSN=SYS1.PARMLIB,DISP=SHR
//SYSIN     DD    *
./          REPL  NAME=LNKLST00,LIST=ALL
                        SYS1.LINKLIB,SYS1.COBLIB
./          ADD   NAME=IEAIGG01,LIST=ALL
                        SYS1962(562B1,NAME1,ALIAS1,...
                        SYS1.LINKLIB,NAME,ALIAS
./          ENDUP
/*
```

Notes

1. The name used on the card after the REPL statement must identify the data set (SYS1.COBLIB) to be concatenated with the system link library, and is selected by the installation. (Note that this data set must be cataloged.)

2. The last two digits of the member-name specified in the ADD statement can vary, but the digits specified here must also be specified in the RAM= parameter used at IPL time. For example, if IEAIGG02 were specified, 'RAM=02' would be required at IPL time. For OS/VS2 Release 1, modify the IEAIGG00 member of SYS1.PARMLIB as explained in OS/VS2 Initialization and Tuning Guide, GC28-0681. For OS/VS2 Release 2 and later, use IEALP00 instead of IEAIGG00.

3. The names and aliases of the COBOL library subroutine members to be made resident must be specified by the installation in the ADD statement. The system searches the last name first; in this case, ALIAS1 is searched last. The user should, therefore, specify the most frequently used name last.

Figure 144. Concatenating the Subroutine Library

The DD statement can be used to create libraries as is discussed at the beginning of this chapter. In addition, DD statements can be used to add members to existing libraries, including the link library, and to retrieve members of existing libraries.

Utility programs can be used to create libraries such as those used in the copy library or as secondary input to the linkage editor. In addition, utility programs can be used to move, copy, and replace members of an existing library; to add, delete, and renumber the records within an existing library; and to assign sequence numbers to the records of a new library.

Linkage editor control statements can be used to make changes to members of a library of load modules. The name of a member can be changed or additional names can be specified. Additional entry points can be identified, existing entry points can be deleted, and portions of a load module can be deleted or replaced. For further information, see the publication OS/VS Linkage Editor and Loader.

USING THE CATALOGED PROCEDURES

A cataloged procedure is a set of job control statements placed in a partitioned data set called the procedure library (SYS1.PROCLIB). It can be retrieved from the library by using its member name in an EXEC statement of a job step in the input stream. Frequently used procedures, such as those used for compiling and linkage editing, can be cataloged to simplify their subsequent use.

A cataloged procedure can contain statements for the processing of an entire job, or it can contain statements to process one or more steps of a job, with the remaining steps defined by job control statements in the input stream. A job can use several cataloged procedures, each processing one or more of the job steps. A job can also call for execution of the same cataloged procedure in more than one job step.

This chapter describes the following:

- How to call cataloged procedures
- The types of cataloged procedures, including those supplied by IBM for use with COBOL source programs
- How to add procedures to the procedure library
- How to modify existing procedures for the current job step only
- How to override and add to cataloged procedures
- How to use the DDNAME parameter in cataloged procedures

CALLING CATALOGED PROCEDURES

A cataloged procedure is called by a job that appears in the input stream. The job must consist of a JOB statement and an EXEC statement that specifies the cataloged procedure name in the positional parameter (either procname or PROC=procname). For example:

```
//STEPQ EXEC COBUC
//STEPQ EXEC PROC=COBUC
```

Either of these EXEC statements could be used to call the IBM-supplied cataloged procedure COBUC to process the job step STEPQ.

A job step that calls for execution of a cataloged procedure can also contain DD statements that are applicable to the job steps of the cataloged procedure. A job that calls for execution of a cataloged procedure may, in other steps, call for execution of other cataloged procedures, call for other executions of the same cataloged procedure, or call directly for execution of load modules. The following example shows a job control procedure that calls both cataloged procedures and load modules.

```
//JOB1          JOB
//STEPA         EXEC   COBUC
//COB.SYSIN     DD     *

          (source module)

/*
//STEPL         EXEC   PGM=IEWL
                .
                .
                .
   (DD statements for the linkage editor)
                .
                .
                .
//STEPE         EXEC   PGM=*.STEPL.SYSLMOD
                .
                .
                .
   (DD statements for user-defined files)
                .
                .
                .
```

The IBM-supplied cataloged procedure COBUC for compilation is used to process STEPA. The COB.SYSIN DD statement is required to define the input to the compiler. The remaining statements in the procedure refer to execution of the linkage editor and the subsequent load module.

Data Sets Produced by Cataloged Procedures

Data sets produced during execution of a cataloged procedure can be used in subsequent job steps. They can also be called as follows:

```
//jobname     JOB  1234,J.SMITH
//STEPA       EXEC PROCED
//PROC1.SYSIN DD  *

          (source module)

/*
//stepname    EXEC PGM=*.STEPA.PROC2.SYSLMOD
                .
                .
                .

     (DD statements for user-defined files)
                .
                .
                .
```

The cataloged procedure PROCED is composed of two job steps, PROC1 and PROC2, that compile and linkage edit the source module.

TYPES OF CATALOGED PROCEDURES

The programmer can write his own procedures and catalog them, or he can use the five COBOL cataloged procedures provided by IBM.

PROGRAMMER-WRITTEN CATALOGED PROCEDURES

The programmer can write cataloged procedures, consisting of EXEC and DD statements, which incorporate job control procedures he uses frequently. For example, the programmer may wish to catalog an EXEC statement and the associated DD statements for a job step that specifies execution of a program. In this way, the DD statements need not be specified each time the program is executed.

In writing a procedure for cataloging, the programmer must follow these rules:

- Another cataloged procedure cannot be referred to, i.e., only the PGM=progname form in an EXEC statement can be used.

  Note, however, that a cataloged procedure may contain a DD statement that refers to a cataloged data set.

- SYSABEND or SYSUDUMP DD statements should not be cataloged because they cannot be overridden.

- The following statements cannot be used in a cataloged procedure:

  1. The JOB statement
  2. A DD statement with JOBLIB in the name field
  3. A DD statement with an * in the operand field
  4. A DD statement with DATA in the operand field
  5. The delimiter statement

Testing Programmer-Written Procedures

A procedure can be tested before it is placed in the procedure library by converting it into an in-stream procedure and executing it any number of times during a job. For further information about in-stream procedures, refer to the section "Testing a Procedure as an In-Stream Procedure".

Adding Procedures to the Procedure Library

The IEBUPDTE utility program is used to add procedures to the procedure library. A description of the use of this program is given in the publication OS/VS Utilities.

In Figure 145, two procedures are added to the procedure library (SYS1.PROCLIB). All control statements are in the input stream.

The first procedure is for a COBOL compilation. Mass storage volumes are specified for the four utility data sets, and 100 tracks are allocated for each utility data set. This cataloged procedure is named COBDA.

The second procedure is also for a COBOL compilation. Unlabeled tape volumes are specified for three utility data sets; for the fourth, SYSUT1, a mass storage device must be specified. This cataloged procedure is named COBTP.

Job control statements: the EXEC card specifies that the IEBUPDTE program is to be executed, and PARM=NEW is used because all data is read from one source, i.e., the input stream.

Utility statements: the ADD statement specifies the member name of the procedure, the level modification (00, first run) and the source of the modification (0, user-supplied). The NUMBER statement specifies the sequence numbers for records in the member. The first record of the cataloged procedure is numbered 00000010, and subsequent records are incremented by tens.

```
|Job            //ADPROC   JOB       1234,J.DUBOB
|Control        //STEP1    EXEC      PGM=IEBUPDTE,PARM=NEW
|Language       //SYSPRINT DD        SYSOUT=A
|Statements     //SYSUT2   DD        DSNAME=SYS1.PROCLIB,DISP=OLD
|               //SYSIN    DD        DATA
|
|Utility        ./         ADD       NAME=COBDA,LEVEL=00,SOURCE=0
|Statements     ./         NUMBER    NEW1=00000010,INCR=00000010
|
|               //COB      EXEC      PGM=IKFCBL00
|               //SYSUT1   DD        UNIT=SYSDA,SPACE=(TRK,(100,10))
|               //SYSUT2   DD        UNIT=SYSDA,SPACE=(TRK,(100,10))
|First          //SYSUT3   DD        UNIT=SYSDA,SPACE=(TRK,(100,10))
|Procedure      //SYSUT4   DD        UNIT=SYSDA,SPACE=(TRK,(100,10))
|               //SYSPRINT DD        SYSOUT=A
|               //SYSPUNCH DD        SYSOUT=B
|
|Utility        ./         ADD       NAME=COBTP,LEVEL=00,SOURCE=0
|Statements     ./         NUMBER    NEW1=10,INCR=10
|
|               //COB      EXEC      PGM=IKFCBL00
|               //SYSUT1   DD        UNIT=SYSDA,SPACE=(TRK,(100,10))
|Second         //SYSUT2   DD        UNIT=2400,LABEL=(,NL)
|Procedure      //SYSUT3   DD        UNIT=2400,LABEL=(,NL)
|               //SYSUT4   DD        UNIT=2400,LABEL=(,NL)
|               //SYSPRINT DD        SYSOUT=A
|               //SYSPUNCH DD        SYSOUT=B
|
|Delimiter      ./         ENDUP
|Statements     /*
```

Figure 145. Example of Adding Procedures to the Procedure Library

Note that leading zeros in the NUMBER statement are not necessary, as indicated in the example for the COBTP procedure.

IBM-SUPPLIED CATALOGED PROCEDURES

IBM distributes cataloged procedures with the program product, which can be incorporated when the system is generated.

Five of the procedures are for use with COBOL programs.

1. COBUC provides for compilation.
2. COBUCL provides compilation and linkage editing.
3. COBULG provides linkage editing and execution.
4. COBUCLG provides for compilation, linkage editing, and execution.
5. COBUCG provides for compilation and loading.

These procedures may be used with any of the job schedulers released as part of the IBM Operating System. When parameters required by a particular scheduler are encountered by another scheduler that does not require those parameters, either they are ignored or alternative parameters are substituted automatically.

The five cataloged procedures are shown in Figures 146 through 150. (Space allocations in these procedures are in terms of record lengths on the 2314 disk storage device.) Note that when DSNAME=&& is used in a DD statement the specified data set is given a unique name by the operating system, and it is assumed to be a temporary data set that will be deleted when the job is completed. If the data set is to be kept, the DD statement can be overridden with a permanent data set name, and the appropriate parameters can be specified.

Note: If the compiler options are not explicitly supplied with the procedure, default options established at the installation apply. The programmer can override these default options by using an EXEC statement that includes the desired options (see "Overriding and Adding to

EXECStatements" and "Overriding Cataloged Procedures Using Symbolic Parameters").

Procedure Naming Conventions

Procedure names begin with the abbreviated name of the processor program, which, in the case of the COBOL procedures, is COB.

The processor's abbreviated name is followed by the processor's level indicator (U) and then by C (compile), L (linkage edit), G (go -- i.e., execute), or combinations of them. Hence, procedure COBUC is a single-step procedure that compiles a program using the COBOL processor; COBUCLG is a 3-step procedure wherein the first step compiles a program using COBOL, the second step link-edits the output of the first step, and the third step executes the output of the linkage editor.

Step Names in Procedures

In a cataloged procedure, the step name is the same as the abbreviated processor name (LKED). The step that executes a compiled and link-edited program is named GO.

For example, in the procedure named COBUCLG, the first step is named COB, the second step is named LKED, and the third step is named GO.

Unit Names in Procedures

The two unit names used in IBM-supplied cataloged procedures are as follows:

SYSSQ	any magnetic tape or mass storage device
SYSDA	any mass storage device

A pool of units must be assigned to these unit names during the system generation procedure. For example, only 2314 Disk Storage Drives might be assigned to the SYSSQ name. Then again, both 2400 Magnetic Tape Units and 2314 Disk Storage Drives might be assigned to the SYSSQ name. Once a pool of devices is assigned to these classes, device selection is done by the Job Scheduler.

Data Set Names in Procedures

When DSNAME=&&name is used in a DD statement, the specified data set is given a unique name by the scheduler, and it is assumed to be a temporary data set that will be deleted when the job terminates. If the data set is to be retained, the DD statement must be overridden with a permanent data set name and appropriate DISP parameters.

COBUC Procedure

The COBUC procedure is a single-step procedure to execute the COBOL compiler. It produces a punched object deck. Figure 145 shows the statements that make up the COBUC cataloged procedure.

The following DD statement must be supplied in the input stream:

```
//COB.SYSIN DD *    (or appropriate
                    parameters defining an
                    input data set)
```

Optionally, the delimiter statement (/*) may follow the source module.

COBUCL Procedure

The COBUCL procedure is a two-step procedure to compile and link-edit using the COBOL compiler. Figure 146 shows the statements that make up the cataloged procedure.

The COB job step produces an object module that is input to the linkage editor. Other object modules may be added as illustrated in Example 5 under "Using the DDNAME Parameter."

The following DD statement, indicating the location of the source module, must be supplied in the input stream:

```
//COB.SYSIN DD *    (or appropriate
                    parameters)
```

COBULG Procedure

The COBULG cataloged procedure is a two-step procedure to link-edit and execute the output of a COBOL compilation. Figure 148 shows the statements that make up the procedure.

The following DD statement indicating the location of the object module must be supplied in the input stream:

```
//LKED.SYSIN DD *   (or appropriate
                    parameters)
```

If the COBOL program refers to SYSIN in the execution step, the following DD statement must also be supplied in the input stream.

```
//GO.SYSIN   DD *   (or appropriate
                    parameters)
```

If the COBOL program refers to other data sets in the execution step such as user-defined files, DD statements that define these data sets must also be provided.

```
//COB        EXEC PGM=IKFCBL00,PARM='DECK,NOLOAD,SUPMAP',REGION=128K
//SYSPRINT   DD SYSOUT=A
//SYSPUNCH   DD SYSOUT=B
//SYSUT1     DD UNIT=SYSDA,SPACE=(460,(700,100))
//SYSUT2     DD UNIT=SYSDA,SPACE=(460,(700,100))
//SYSUT3     DD UNIT=SYSDA,SPACE=(460,(700,100))
//SYSUT4     DD UNIT=SYSDA,SPACE=(460,(700,100))
```

Figure 146. Statements in the COBUC Procedure

```
//COB        EXEC PGM=IKFCBL00,REGION=128K
//SYSPRINT   DD SYSOUT=A
//SYSUT1     DD UNIT=SYSDA,SPACE=(460,(700,100))
//SYSUT2     DD UNIT=SYSDA,SPACE=(460,(700,100))
//SYSUT3     DD UNIT=SYSDA,SPACE=(460,(700,100))
//SYSUT4     DD UNIT=SYSDA,SPACE=(460,(700,100))
//SYSLIN     DD DSNAME=&&LOADSET,DISP=(MOD,PASS),UNIT=SYSDA,          X
//              SPACE=(80,(500,100))
//LKED       EXEC PGM=IEWL,PARM='LIST,XREF,LET',COND=(5,LT,COB),      X
//              REGION=96K
//SYSLIN     DD DSNAME=&&LOADSET,DISP=(OLD,DELETE)
//           DD DDNAME=SYSIN
//SYSLMOD    DD DSNAME=&&GOSET,DISP=(NEW,PASS),UNIT=SYSDA,            X
//              SPACE=(1024,(50,20,1))
//SYSLIB     DD DSNAME=SYS1.COBLIB,DISP=SHR
//SYSUT1     DD UNIT=(SYSDA,SEP=(SYSLIN,SYSLMOD)),                    X
//              SPACE=(1024,(50,20))
//SYSPRINT   DD SYSOUT=A
```

Figure 147. Statements in the COBUCL Procedure

```
//LKED       EXEC PGM=IEWL,PARM='LIST,XREF,LET',REGION=128K
//SYSLIN     DD DDNAME=SYSIN
//SYSLMOD    DD DSNAME=&&GOSET(GO),DISP=(NEW,PASS),UNIT=SYSDA,        X
//              SPACE=(1024,(50,20,1))
//SYSLIB     DD DSNAME=SYS1.COBLIB,DISP=SHR
//SYSUT1     DD UNIT=(SYSDA,SEP=(SYSLIN,SYSLMOD)),                    X
//              SPACE=(1024,(50,20))
//SYSPRINT   DD SYSOUT=A
//GO         EXEC PGM=*.LKED.SYSLMOD,COND=(5,LT,LKED)
```

Figure 148. Statements in the COBULG Procedure

```
//COB        EXEC PGM=IKFCBL00,PARM=SUPMAP,REGION=128K
//SYSPRINT   DD SYSOUT=A
//SYSUT1     DD UNIT=SYSDA,SPACE=(460,(700,100))
//SYSUT2     DD UNIT=SYSDA,SPACE=(460,(700,100))
//SYSUT3     DD UNIT=SYSDA,SPACE=(460,(700,100))
//SYSUT4     DD UNIT=SYSDA,SPACE=(460,(700,100))
//SYSLIN     DD DSNAME=&&LOADSET,DISP=(MOD,PASS),UNIT=SYSDA,          X
//               SPACE=(80,(500,100))
//LKED       EXEC PGM=IEWL,PARM='LIST,XREF,LET',COND=(5,LT,COB),      X
//               REGION=96K
//SYSLIN     DD DSNAME=&&LOADSET,DISP=(OLD,DELETE)
//           DD DDNAME=SYSIN
//SYSLMOD    DD DSNAME=&&GOSET(GO),DISP=(NEW,PASS),UNIT=SYSDA,        X
//               SPACE=(1024,(50,20,1))
//SYSLIB     DD DSNAME=SYS1.COBLIB,DISP=SHR
//SYSUT1     DD UNIT=(SYSDA,SEP=(SYSLIN,SYSLMOD)),                    X
//               SPACE=(1024,(50,20))
//SYSPRINT   DD SYSOUT=A
//GO         EXEC PGM=*.LKED.SYSLMOD,COND=((5,LT,COB),(5,LT,LKED))
```

Figure 149. Statements in the COBUCLG Procedure

```
//COB        EXEC PGM=IKFCBL00,PARM='LOAD',REGION=128K
//SYSPRINT   DD SYSOUT=A
//SYSUT1     DD UNIT=SYSDA,SPACE=(460,(700,100))
//SYSUT2     DD UNIT=SYSDA,SPACE=(460,(700,100))
//SYSUT3     DD UNIT=SYSDA,SPACE=(460,(700,100))
//SYSUT4     DD UNIT=SYSDA,SPACE=(460,(700,100))
//SYSLIN     DD DSNAME=&&LOADSET,DISP=(MOD,PASS),                     X
//               UNIT=SYSDA,SPACE=(80,(500,100))
//GO         EXEC PGM=LOADER,PARM='MAP,LET',COND=(5,LT,COB),REGION=106K
//SYSLIN     DD DSNAME=*.COB.SYSLIN,DISP=(OLD,DELETE)
//SYSLOUT    DD SYSOUT=A
//SYSLIB     DD DSNAME=SYS1.COBLIB,DISP=SHR
```

Figure 150. Statements in the COBUCG Procedure

COBUCLG Procedure

The COBUCLG procedure is a three-step procedure to compile, link-edit, and execute using the COBOL compiler. Figure 149 shows the statements that make up the procedure.

The COB job step produces an object module that is input to the linkage editor. Other object modules may be added as illustrated in Example 5 under "Using the DDNAME Parameter."

The following DD statement, indicating the location of the source module, must be supplied in the input stream:

```
//COB.SYSIN DD      * (or appropriate
                        parameters)
```

If the COBOL program refers to SYSIN, the following DD statement indicating the location of the input data set must also be supplied:

```
//GO.SYSIN  DD      * (or appropriate
                        parameters)
```

If the COBOL program refers to other data sets, DD statements that define these data sets must also be supplied.

COBUCG Procedure

The COBUCG procedure is a two-step procedure to compile, load, and execute using the COBOL compiler and OS loader. Figure 150 shows the statements that make up the procedure.

The COB job step produces an object module that is input to the loader.

The following DD statement, indicating the location of the source module, must be supplied in the input stream:

```
//COB.SYSIN DD     * (or appropriate
                     parameters)
```

If the COBOL program refers to SYSIN, the following DD statement indicating the location of the input data set must also be supplied:

```
//GO.SYSIN  DD     * (or appropriate
                     parameters)
```

If the COBOL program refers to other data sets, the DD statements that define these data sets must also be supplied.

MODIFYING EXISTING CATALOGED PROCEDURES

Existing cataloged procedures can be permanently modified by using the IEBUPDTE utility program described in the publication OS/VS Utilities.

OVERRIDING AND ADDING TO CATALOGED PROCEDURES

Any parameter in a cataloged procedure except the PGM=progname parameter in the EXEC statement can be overridden. Parameters or statements not specified in the procedure can also be added. When a cataloged procedure is overridden or added to, the changes apply only during one execution.

OVERRIDING AND ADDING TO EXEC STATEMENTS

An EXEC statement can be overridden or added to in one of two ways:

1. Specify, in the operand field of the EXEC statement calling the procedure, the keyword, the procedure step-name and the subparameters, for example:

   COND.procstep=(subparameters)

   If a multistep procedure is being modified, parameters in the calling EXEC statement must be specified step by step; i.e., the parameters for one step must be specified before those of the next step. If the return code of a cataloged procedure step is to be tested, the name of the step in the procedure (procstep) must be qualified by the name of the step that called for execution of the cataloged procedure (stepname).

2. Specify in the operand field of the EXEC statement calling the procedure only the keyword parameters and subparameters, for example:

   COND=(subparameters)

   If a multistep procedure is being called, the specified parameters (with the exception of PARM) apply to all steps in the procedure. The PARM keyword subparameters override the first EXEC statement and nullify any subsequent PARM keyword subparameters. The COND and ACCT parameters apply to all steps in the procedure. To override PARM parameters in job steps other than the first, the previous method can be used.

Note: A parameter in an EXEC statement cannot be partly overridden; it must be overridden in its entirety. Any parameter not overridden remains as originally defined.

Examples of Overriding and Adding to EXEC Statements

This section contains examples of overriding and adding to the EXEC statement. The procedures overridden or added to are the IBM procedures shown in Figures 146 through 150.

Example 1: The following example shows the overriding of one parameter in the EXEC statement of the one procedure step in the IBM-supplied COBUC procedure. The statements appear in the input stream as follows:

```
//jobname   JOB  1234,J.SMITH
//STEPA     EXEC COBUC,PARM.COB='DECK,     X
//                NOLOAD,BUF=4000,         X
//                SIZE=9600'
//COB.SYSIN DD   *

          (source module)

/*
```

Note: In actual use the PARM.COB parameter cannot be continued in this manner. In the PARM parameter that is overridden, the DECK and NOLOAD options were specified. They are included again since the parameter must be overridden in its entirety. The

information is here enclosed in single quotation marks, since subparameters that contain equal signs must be enclosed in this manner.

Example 2: The following example shows the overriding of two parameters and the adding of another in the EXEC statement of one procedure step of the IBM-supplied COBUCLG procedure. The statements appear in the input stream as shown:

```
//jobname    JOB  1234,J.SMITH
//STEPA      EXEC COBUCLG,PARM.LKED=       X
//                (MAP,LIST),ACCT=(1234),  X
//                COND.LKED=(9,LT,         X
//                STEPA.COB)
//COB.SYSIN  DD   *
          (source module)
/*
```

Note: In actual use the COND.LKED and PARM.LKED parameters cannot be continued in this manner. For the linkage editor job step in the above example, the COND and PARM parameters have been overridden and the ACCT parameter added.

Example 3: The following example shows the overriding of individual parameters in more than one procedure step of the IBM-supplied COBUCLG procedure. The statements appear in the input stream as shown.

```
//jobname    JOB  1234,J.SMITH
//stepname   EXEC COBUCLG,PARM.LKED=OVLY, X
//                COND.GO=((5,EQ,          X
//                stepname.COB),           X
//                (5,EQ,stepname.LKED))
//COB.SYSIN  DD   *
          (source module)
/*
```

Note: In actual use the COND.GO statement cannot be continued in this manner. The PARM option OVLY replaces the PARM subparameters of the link-edit job step. The COND option EQ (equal to) replaces the option LT (less than) in the execution job step.

Note that all overriding parameters for one step of the procedure must be specified before those for the next step.

Example 4: The following example shows the overriding of parameters on all EXEC statements in the IBM-supplied COBUCLG procedure. The statements appear in the input stream as shown:

```
//jobname  JOB    1234,J.SMITH
//stepname EXEC   COBUCLG,                 X
//                PARM=(LOAD,PMAP),        X
//                COND=(3,LT),             X
//                ACCT=(123456,DEPTQ)
//COB.SYSIN  DD    *
          (source module)
/*
```

The PARM options are added to the procedure step COB and nullify the PARM options in the LKED and GO steps. The COND and ACCT parameters apply to all steps in the procedure.

TESTING A PROCEDURE AS AN IN-STREAM PROCEDURE

A procedure can be tested before it is placed in the procedure library by converting it into an in-stream procedure and executing it any number of times during a job. In-stream procedures are described in detail in the publication OS/VS JCL Services.

An in-stream procedure is a series of job control language statements enclosed within a PROC statement and a PEND statement. The following example shows how to convert the COBUC procedure (Figure 146) into an in-stream procedure and execute it twice. (Remember that in actual use the parameters cannot be continued in this manner.)

```
//CONVERT    JOB  1234,YOURNAME
//INSTREAM   PROC
//COB        EXEC PGM=IKFCBL00,PARM='DECK,          X
                NOLOAD,SUPMAP',                     X
                REGION=128K
//SYSPRINT   DD   SYSOUT=A
//SYSPUNCH   DD   SYSOUT=B
//SYSUT1     DD   DSNAME=&&SYSUT1,                  X
//               UNIT=SYSDA,                        X
//               SPACE=(460,6700,100))
//SYSUT2     DD   DSNAME=&&SYSUT2,                  X
//               UNIT=SYSDA,                        X
//               SPACE=(460,6700,100))
//SYSUT3     DD   DSNAME=&&SYSUT3,                  X
//               UNIT=SYSDA,                        X
//               SPACE=(460,6700,100))
//SYSUT4     DD   DSNAME=&&SYSUT4,                  X
//               UNIT=SYSDA,                        X
//               SPACE=(460,6700,100))
//ENDPROC    PEND
//          EXEC INSTREAM
//COB.SYSIN  DD   *

                (input data)

/*
//          EXEC INSTREAM
//COB.SYSIN  DD   *
                (input data)
/*
```

OVERRIDING AND ADDING TO DD STATEMENTS

A DD statement can be overridden or added to by using a DD statement whose name is composed of the procedure step-name that qualifies the ddname of the DD statement being overridden, as follows:

```
//procstep.ddname  DD  (appropriate
                        parameters)
```

Entire DD statements can also be added.

There are rules that must be followed when overriding or adding a DD statement within a step in a procedure.

- Overriding DD statements must be in the same order in the input stream as they are in the cataloged procedure.
- DD statements to be added must follow overriding DD statements.

There are some special cases that should be kept in mind when overriding a DD statement.

- All parameters are overridden in their entirety, except for the DCB and AMP parameters. Within the DCB and AMP parameters, individual subparameters may be overridden.
- To nullify a keyword parameter (except the DCB and AMP parameters), write, in the overriding DD statement, the keyword and an equal sign followed by a comma. For example, to nullify the use of the UNIT parameter, specify UNIT=, in the overriding DD statement.
- A parameter can be nullified by specifying a mutually exclusive parameter. For example, the SPACE parameter can be nullified by specifying the SPLIT parameter in the overriding DD statement.
- The DUMMY parameter can be nullified by omitting it and specifying the DSNAME parameter in the overriding DD statement.
- To override DD statements in a concatenation of data sets, the programmer must provide one DD statement for each data set in the concatenation. Only the first DD statement in the concatenation should be named. However, if a DD statement to be changed follows one (or more) DD statement(s) to be left intact, the first overriding statement(s) should have a blank operand.
- If the DDNAME=ddname parameter is specified in a cataloged procedure, it cannot be overridden; rather it can refer to a DD statement supplied at the time of execution.

Examples of Overriding and Adding to DD Statements

This section contains examples of overriding and adding to parameters in DD statements. The procedures overridden or added to are the IBM procedures shown in Figures 146 through 150.

The DDNAME parameter is not used in these examples, although it can be useful with the cataloged procedures. The use of the DDNAME parameter is described in detail later in this chapter.

Example 1: The following example shows the overriding of DD statements in the IBM-supplied COBUCLG procedure.

```
//jobname      JOB  1234,J.SMITH
//stepname     EXEC COBUCLG
//COB.SYSLIN   DD   DSNAME=GOFILE
//COB.SYSIN    DD   *
```

```
          (source module)
/*
//LKED.SYSLIN  DD   DSNAME=*.COB.SYSLIN,  X
//                  DISP=(OLD,CATLG)
                .
                .
                .
/*

        (other DD statements for
        user-defined files)
                .
                .
                .
/*
```

The name of the data set in SYSLIN in the procedure step COB is changed to GOFILE. The name of the data set of SYSLIN in the procedure step LKED is changed to a reference to the SYSLIN DD statement in the COB procedure step, and the data set name GOFILE is cataloged.

Example 2: The following example shows the adding of DD statements to the IBM-supplied COBUCLG procedure.

```
//jobname      JOB  1234,J.SMITH
//stepname     EXEC COBUCLG,                X
//               PARM.COB= (DECK,LOAD,PMAP)
//COB.SYSPUNCH DD   SYSOUT=B
//COB.SYSIN    DD   *

            (source module)

/*
//GO.TRANSACT  DD   DSNAME=JUNE,DISP=OLD
                .
                .
                .

        (other DD statements for
        user-defined files)
                .
                .
                .
/*
```

Note: In the foregoing example TRANSACT is a cataloged data set. When a data set is cataloged, it is sufficient to refer to it by DSNAME and DISP=OLD.

The PARM.COB option DECK and the SYSPUNCH DD statement are added to obtain a punched object module. The PARM option PMAP is added to obtain a listing of the assembler language expansion of the source module.

Example 3: The following example shows overriding and adding to DD statements at the same time in the IBM-supplied COBUC procedure. Note that overriding statements must be in the same sequence as they appear in the procedure and must precede those statements being added.

```
//jobname      JOB  1234,J.SMITH
//stepname     EXEC COBUC,PARM.COB=(LOAD)
//COB.SYSUT2   DD   SPACE=,UNIT=SYSSQ
//COB.SYSLIN   DD   DSNAME=&&GOFILE,       X
//                    DISP=(MOD,PASS),     X
//                    UNIT=SYSSQ
//COB.SYSIN    DD   *

            (source module)

/*

      (subsequent job steps)
                .
                .
                .
```

The device class on the COB.SYSUT2 DD statement is changed to SYSSQ, and the SPACE parameter is nullified. Therefore, mass storage devices cannot be allocated. Any tape volumes to be assigned must have standard labels. The COB.SYSLIN DD statement is changed so that it passes the object module to subsequent job steps.

Example 4: The following example shows how to concatenate a data set with a data set defined in the COBULG procedure.

```
//jobname      JOB   1234,J.SMITH
//stepname     EXEC  COBULG
                .
                .
                .
//LKED.SYSLIB  DD    [blank operand field]
//             DD    [parameters]
                .
                .
                .
/*
```

Instead of the blank operand field, parameters could have been used to override the SYSLIB statement; the data set defined by the unnamed DD statement would then be concatenated to the data set that was redefined by overriding.

Note that a number of libraries could be concatenated to the SYSLIB data set. For example:

```
//LKED.SYSLIB  DD
//             DD  DSNAME=USERLIB,DISP=OLD
//             DD  DSNAME=MYLIB,DISP=OLD
```

USING THE DDNAME PARAMETER

The DDNAME parameter is used to define a dummy data set that can assume the characteristics of an actual data set, defined by a subsequent DD statement within the step. If a matching DD statement is found, its characteristics, with the exception of its ddname, replace those of the statement using the DDNAME parameter. If a matching DD statement is not found within the step, the data set defined by the DDNAME parameter remains a dummy.

This section contains examples showing the use of the DDNAME parameter with cataloged procedures.

The rules for using the DDNAME parameter are as follows:

- A backward reference (e.g., *.ddname) to a DD statement referred to by a DDNAME parameter cannot be used because the statement that is referred to loses its identity.

- A backward reference to a statement containing a DDNAME parameter can be used, but only after the statement to which the DDNAME parameter refers has been encountered. If a backward reference is used before the dummy data set (defined by DDNAME) has been given real characteristics, these real characteristics will not be transferred to the DD statement that contains the backward reference. For example, if DCB=*.ddname is used (where ddname is the name of a statement containing an unresolved DDNAME parameter), the DCB fields that are transferred are blank.

- Unnamed DD statements can be placed after a statement containing the DDNAME parameter (indicating concatenation), but unnamed DD statements cannot be placed after a statement referred to by a DDNAME parameter.

- The DDNAME parameter can be used a maximum of five times in a step, but each DDNAME parameter must refer to a different statement.

- The DDNAME parameter cannot be used in a JOBLIB statement.

- The DDNAME parameter is not to be used to refer to a DD statement with the DYNAM parameter specified.

When using the DDNAME parameter, the programmer should also keep the following in mind:

- The name of the DD statement referred to does not replace the name of the referencing statement.

- If a statement that contains the DDNAME parameter is overridden, it is nullified.

- If overriding is performed with a statement that contains the DDNAME parameter, all parameters in the overridden statement are nullified.

The following DD statements:

```
//S1        EXEC PGM=progname
//D1        DD   DDNAME=D3
//D2        DD   (parameters X,Y,Z)
//D3        DD   (parameters U,T,V)
```

will result in the same data definition produced by the following statements:

```
//S1        EXEC PGM=progname
//D1        DD   (parameters U,T,V)
//D2        DD   (parameters X,Y,Z)
```

EXAMPLES OF USING THE DDNAME PARAMETER

Example 1: The following example shows how to override the first DD statement in a cataloged procedure with a DD * statement, and allow subsequent statements to be processed. The cataloged procedure (PROC3) is as follows:

```
//STEP1     EXEC PGM=progname
//DD1       DD   (any parameters)
//DD2       DD   (any parameters)
```

The job procedure in which the overriding takes place appears in the input stream as follows:

```
//JOB1      JOB  1234,J.SMITH
//S1        EXEC PROC3
//STEP1.DD1 DD   DDNAME=D1
//D1        DD   *
```

The STEP1.DD1 statement overrides the DD1 statement; the DD2 statement is processed; then the D1 statement is processed.

Example 2: The following example shows how to override the first DD statement in a cataloged procedure with a DD * statement

and how to add a DD statement. The cataloged procedure (PROC3) is as follows:

```
//STEP1     EXEC PGM=progname
//DD1       DD   (any parameters except
                  DATA or *)
//DD2       DD   (any parameters except
                  DATA or *)
```

The job procedure in which the overriding takes place appears in the input stream as follows:

```
//JOB2      JOB  1234,J.SMITH
//S1        EXEC PROC3
//STEP1.DD1 DD   DDNAME=DD4
//STEP1.DD3 DD   (any parameters except
                  DATA or *)
//DD4       DD   *
```

The DD4 statement effectively overrides the DD1 statement, after the DD2 statement has been processed and the DD3 statement has been added.

Example 3: The following example shows how to concatenate a data set in the input stream with a data set defined by a DD statement in a cataloged procedure. The cataloged procedure (PROC3) is as follows:

```
//STEP1     EXEC PGM=progname
//DD1       DD   (any parameters except
                  DATA or *)
//DD2       DD   (any parameters except
                  DATA or *)
```

The job procedure in which the concatenation takes place appears in the input stream as follows:

```
//JOB3      JOB  1234,J.SMITH
//S1        EXEC PROC3
//STEP1.DD1 DD   (blank operand field)
//          DD   DDNAME=DD3
//DD3       DD   *
```

The data set in the input stream is concatenated with the data set defined by the DD1 statement after the DD2 statement has been processed.

Example 4: The following example shows how to concatenate a data set in the input stream with a data set defined by a DD statement in a cataloged procedure and how to add a DD statement. The cataloged procedure (PROC3) is as follows:

```
//STEP1     EXEC PGM=progname
//DD1       DD   (any parameters except
                  DATA or *)
//DD2       DD   (any parameters except
                  DATA or *)
```

The job procedure in which the concatenation takes place appears in the input stream as follows:

```
//JOB4      JOB  1234, J.SMITH
//S1        EXEC PROC3
//STEP1.DD2 DD   (blank operand field)
//          DD   DDNAME=DD4
//STEP1.DD3 DD   (any parameters except
                  DATA or *)
//DD4       DD   *
```

Example 5: The following example shows how the statement DD DDNAME=SYSIN in the IBM-supplied COBUCLG procedure can be used to add more object modules as input to the linkage editor. The statements appear in the input stream as follows:

```
//jobname     JOB    1234,J.SMITH
//stepname    EXEC   COBUCLG
                .
                .
                .
//COB.SYSIN   DD     *

      (source deck)

/*
//LKED.SYSIN  DD     *

      (first object module)
              .
              .
              .

      (last object module)

/*

(//GO. cards)
```

The COBUCLG procedure contains the following two statements in the linkage edit step:

```
//SYSLIN DD DSNAME=&&LOADSET,                X
//          DISP=(OLD,DELETE)
//       DD DDNAME=SYSIN
```

The result of concatenating SYSIN with SYSLIN is that when SYSLIN (input to linkage editor) is read, SYSIN is also read and linked with it. For example, if ILBODSP0 is one of the object modules in the SYSIN stream, it will be linked with SYSLIN. The ILBODSP0 module from SYS1.COBLIB will not be used.

USING THE SORT/MERGE FEATURE

To use the Sort/Merge feature of the OS/VS COBOL Compiler, sort/merge feature statements are written in the COBOL source program. These statements are described in the publication IBM VS COBOL for OS/VS. The Sort/Merge program itself is described in the appropriate Sort/Merge Programmer's Guide.

Use of the full COBOL Sort/Merge feature requires the program product OS/VS Sort/Merge, 5740-SM1. The OS Sort/Merge, 5734-SM1, may only be used if COBOL Sort alone is used--without alternate collating sequence and without merge. (If 5734-SM1 is used, it may issue message IGH067I, "INVALID EXEC OR ATTACH PARAMETER." This will have no effect on operations, and may be safely ignored.)

DD statements must be written in the execution-time job steps of the procedure to describe the data sets used by the sort/merge program. DD statements for data sets used during the sort/merge process are described below.

Note: The Sort/Merge Checkpoint Restart feature is available to the programmer through the use of the RERUN statement.

SORT/MERGE DD STATEMENTS

Three types of data sets can be defined for the sort program in the execution time job step: input, output, and work. In addition, data sets must be defined for the use of the system during the sorting operation.
For MERGE, work DD statements are required.

The maximum number of files that can be merged is 8.

SORT INPUT DD STATEMENTS

The input data set is associated with a ddname that appears as the ddname portion of the system-name in an ASSIGN clause in the COBOL source program. When the USING option is specified, the compiler will generate an input procedure that will open the data set, read the records, release the records and close the data set.

SORT OUTPUT DD STATEMENTS

The output data set is associated with a ddname that appears as the ddname portion of the system-name in an ASSIGN clause in the COBOL source program. When the GIVING option is specified, the compiler generates an output procedure that will open the data set, return the records, write the records, and close the data set.

SORT WORK DD STATEMENTS

The sort program requires at least three work data sets. The ddname for each DD statement is in the form SORTWKnn, where nn is a decimal number. The ddnames for the required data sets must be SORTWK01, SORTWK02, and SORTWK03. Additional work data sets may be defined, but their ddnames must be consecutively numbered, beginning with 04.

SORTWKnn Data Set Considerations

Intermediate data sets (i.e., SORTWKnn data sets) for a sort/merge may be assigned to either magnetic tape or mass storage devices. All of the intermediate storage for one sort/merge must be assigned to the same device type. These may not be on both 7-track and 9-track tape units in the same sort. Any one of the following devices may be used for intermediate storage:

- IBM 2400-series Magnetic Tape Unit (7- or 9-track)
- IBM 3400-series Magnetic Tape Unit (9-track)
- IBM 2314/2319 Direct Access Storage Facility
- IBM 3330 Direct Access Storage Facility
- IBM 3340 Direct Access Storage Facility
- IBM 3350 Direct Access Storage Facility

The Sort/Merge Programmer's Guide contains detailed information about these devices.

Since spanned records can be input to and output from the sorting operation, it is the user's responsibility to assign the sort work files to mass storage devices whose track sizes are larger than the logical record size of the records being sorted. An S-mode file whose logical record length is greater than its track size may be sorted by assigning the work files to a magnetic tape unit.

If data sets not involved in the sorting or merging operation are assigned to tape units, these tape units may be used as sort work files by using the UNIT=AFF parameter. For example, if PAYROLL is specified as the ddname of the ASSIGN clause in a SELECT statement, the tape unit assigned to PAYROLL could be used as a sort work file by using the following DD statement:

```
//PAYROLL DD UNIT=2400,...
//SORTWK02 DD UNIT=AFF=PAYROLL...
```

Input DD Statement

The input data set must reside on a physical device, a magnetic tape unit, a mass storage device, or in the system input stream. The following example shows DD statement parameters that could be used to define a cataloged input data set.

```
//INSORT   DD   DSNAME=INPT,                X
//              DISP=(OLD,DELETE)
```

These parameters cause the system to search the catalog for a data set named INPT (DSNAME parameter). When found, the data set is associated with the ddname INSORT and used by the sort program. The control program obtains the unit assignment and volume serial number from the catalog, and displays a mounting message to the operator. The DISP parameter indicates that the data set has already been created (OLD). It also indicates that the data set should be deleted (DELETE) after the current job step.

Output DD Statment

The output DD statement must define all of the characteristics of the output data set. The following example shows DD statement parameters that could be used to characterize an output data set:

```
//OUTSORT  DD   DSNAME=OUTPT,UNIT=2400,   X
//              DISP=(NEW,CATLG)
```

The DISP parameter indicates that the data set is unknown to the operating system (NEW) and that it should be cataloged (CATLG) under the name OUTPT (DSNAME parameter). The UNIT parameter specifies that the data set is on a 2400-series tape unit.

SORTWKnn DD Statements

SORTWKnn data sets may be contained on tape or mass storage volumes. When mass storage space is assigned, only the primary allocation is used by the sort, and it must be contiguous.

Note that the SORTWKnn data sets:

1. May not be spread over more than one device type.
2. May not be on 7-track tape when the input data set is on 9-track tape.
3. May be on 7-track tape when the output data set is on 9-track tape.
4. Cannot use the data conversion feature if they are on 7-track tape. The TRTCH subparameter must reflect this.
5. May be on 9-track tape when the input data set is on 7-track tape.

SORTWKnn Example A: The following DD statement parameters could be used to define a tape intermediate storage data set:

```
//SORTWK01 DD   UNIT=2400,LABEL=(,NL),    X
//              VOLUME=SER=DUMMY
```

These parameters specify an unlabeled data set on a 2400-series tape unit. Since the DSNAME parameter is omitted, the system assigns a unique name to the data set. The omission of the DISP parameter causes the system to assume that the data set is new and that it should be deleted at the end of the current job step. The 2400-series tape units are explicitly of the 9-track format.

SORTWKnn Example B: The following DD statement parameters could be used to define a mass storage intermediate storage data set:

```
//SORTWK01 DD   UNIT=2314,                 X
//              SPACE=(TRK,(200),,CONTIG)
```

These parameters specify a mass storage data set with a standard label (LABEL parameter default value). The SPACE parameter specifies that the data set is to be allocated 200 contiguous tracks. The system assigns a unique name to the data set and deletes it at the end of the job step.

ADDITIONAL DD STATEMENTS

The sort/merge program requires two additional DD statements:

```
//SYSOUT    DD    SYSOUT=A
```

which defines the system output data set.

```
//SORTLIB   DD    DSNAME=SYS1.SORTLIB,      X
//                DISP=SHR
```

which defines the library containing the SORT/MERGE modules.

Note: At OS/VS Sort/Merge installation time, the programmer can designate that Sort/Merge diagnostic messages be printed on a specified data set. The FLAG option determines whether the messages directed to this data set are either uncorrectable-error messages or both informational and uncorrectablele-error Sort/Merge messages. In either case, uncorrectable-error messages are displayed on the console. SYSOUT is the default data set that is modified by the FLAG option. If there are DISPLAY or EXHIBIT statements in the COBOL program, the Sort/Merge messages cannot be routed to the same data set designated for the output of the DISPLAY and/or EXHIBIT statements. Therefore, one of the following courses of action should be considered:

- The Sort/Merge default data set should be changed at installation time by specifying the PRINT=parameter and selecting a DD name other than SYSOUT. At execution time, the selected DD name must be specified on a DD statement.
- In the COBOL source program, a DD name other than SYSOUT should be placed in the SORT-MESSAGE special register prior to the SORT statement. At execution time, the selected DD name must be specified in a DD statement.
- At compile time, the SYSOUx option can be used to designate a file for COBOL DISPLAY output other than SYSOUT. Therefore, Sort/Merge messages can go to the SYSOUT file.

SHARING DEVICES BETWEEN TAPE DATA SETS

A single tape unit may be assigned to two sort data sets when the data sets are one of the following pairs:

- The input data set and the first intermediate storage data set (SORTWK01).
- The input data set and the output data set.

The AFF subparameter of the UNIT parameter can be used to associate the input data set with either the SORTWK01 data set or the output data set. The subparameter can appear in the DD statement for SORTWK01 or output.

USING MORE THAN ONE SORT/MERGE STATEMENT IN A JOB

More than one SORT/MERGE statement may be used in a single program or in two or more programs that are combined into a single load module.

SORT PROGRAM EXAMPLE

The control cards in Figure 151 could be used with the sample program that illustrates the Sort feature. A description of the Sort Feature can be found in the publication IBM VS COBOL for OS/VS.

```
//SORTEST      JOB   NY838670165,              X
//                   'J.SMITH',                X
//                   MSGLEVEL=1
//SORTJS3      EXEC  COBUCLG
//COB.SYSIN    DD    *
                     .
                     .
                     .
       (COBOL source program)
                     .
                     .
                     .
//GO.SORTWK01  DD    UNIT=2314,                X
//                   SPACE=(TRK,(200),         X
//                   ,CONTIG)
//GO.SORTWK02  DD    UNIT=2314,                X
//                   SPACE=(TRK,(200),         X
//                   ,CONTIG)
//GO.SORTWK03  DD    UNIT=2314,                X
//                   SPACE=(TRK,(200),         X
//                   ,CONTIG)
//GO.OUTSORT   DD    UNIT=183,                 X
//                   LABEL=(,NL),              X
//                   VOLUME=SER=NONE
//GO.SYSOUT    DD    SYSOUT=A
//GO.SORTLIB   DD    DSNAME=SYS1.SORTLIB,      X
//                   DISP=SHR
//GO.INFILE    DD    UNIT=182,                 X
//                   LABEL=(,NL),              X
//                   VOLUME=SER=DUMMY
```

Figure 151. Sort Feature Control Cards

The minimum number of SORTWKnn data sets are used; the sort operation can be optimized by using additional work data sets (see the appropriate Sort/Merge Programmer's Guide).

CATALOGING SORT/MERGE DD STATEMENTS

Since repeated use of the Sort/Merge feature often involves the same execution time DD statements, the user may wish to catalog them (see "Using the Cataloged Procedures").

LINKAGE WITH THE SORT/MERGE PROGRAM

Communication between the Sort/Merge program and the COBOL program is maintained by the COBOL library subroutine ILBOSMG0. This routine links to the Sort/Merge program using the load module name SORT. It is the user's responsibility to set up his LINKLIB/STEPLIB in such a way that an alias or load module name of SORT/MERGE points to the first module of the Sort/Merge program he wishes to use. The programmer must also designate via the appropriate SORTLIB DD statement the library of the Sort/Merge program he wishes to use.

If the INPUT PROCEDURE option of the SORT/MERGE statement is specified, exit E15 of the Sort/Merge program is used. The return code indicating "insert records" is issued when a RELEASE statement is encountered, and the return code indicating "do not return" is issued when the end of the procedure is encountered.

If the OUTPUT PROCEDURE option is specified, exit E35 of the Sort/Merge program is used. The return code indicating "delete records" is issued when a RETURN statement is encountered, and the return code indicating "do not return" is issued when the end of the procedure is encountered. (For additional information, about the Sort/Merge program, see the appropriate Sort/Merge Programmer's Guide.)

Completion Codes

The Sort/Merge program returns a completion code upon termination. This code may be interrogated by the COBOL program. The codes are:

- 0 -- Successful completion of Sort/Merge
- 16 -- Unsuccessful completion of Sort/Merge

SUCCESSFUL COMPLETION: When a Sort/Merge application has been successfully executed, a completion code of zero is returned and the sort terminates.

UNSUCCESSFUL COMPLETION: If the sort/merge, during execution, encounters an error that will not allow it to complete successfully, it returns a completion code of 16 and terminates. (Possible errors include an out-of-sequence condition or an input/output error that cannot be corrected.) The Sort/Merge Programmer's Guide contains a detailed description of the conditions under which this termination will occur.

The returned completion code is stored in a special register called SORT-RETURN by the COBOL library subroutine; an unsuccessful termination of the sort may then be tested for and appropriate action specified. Note that the contents of SORT-RETURN will change with the execution of a SORT statement. The following is an example of the use of SORT-RETURN with the sort feature:

```
SORT SALES-RECORDS ON ASCENDING KEY
CUSTOMER-NUMBER, DESCENDING KEY DAYTE,
USING FN-1, GIVING FN-2.

IF SORT-RETURN NOT EQUAL TO ZERO,
DISPLAY 'SORT UNSUCCESSFUL' UPON
CONSOLE, STOP RUN.
```

If no references to SORT-RETURN are made in a program, an unsuccessful sort will generate the following message:

```
IKF888I- UNSUCCESSFUL SORT FOR SD
         SORT-FILE DDNAME
```

See "Appendix K: Diagnostic Messages" for a description of action to be taken.

A normal Sort/Merge operation will produce the following messages from the OS/VS Sort/Merge program: ICE036I, ICE037I, ICE038I, ICE045I, ICE049I, ICE052I, ICE054I, and ICE055I. Other messages may appear depending on the COBOL options specified.

If the OS Sort/Merge program product is used, the same messages will appear with the same numbers, but the three-character prefix will be different.

TERMINATING THE SORT PROGRAM FROM THE COBOL PROGRAM

By placing the value 16 in the special register SORT-RETURN during the execution of an input or output procedure, the COBOL program can terminate the OS Sort/Merge program product immediately after executing the next RELEASE or RETURN statement. When control returns to the statement following the Sort statement, the special register SORT-RETURN will contain 16 to indicate an unsuccessful completion.

LOCATING SORT/MERGE RECORD FIELDS

Records defined under a COBOL SD are assigned a BLL (Base Locator for Linkage Section), or a BL (Base locator) if SAME RECORD AREA is specified. Location of a given data item in an object-time dump when the record in which it is contained references a BLL can be determined as follows:

1. From the compilation listing, determine:
   a. The displacement of the item (see Data Division Map).
   b. The relative address of the BLL CELLS (see the Memory Map Table).
   c. The BLL number.
2. From the dump, determine the relocation factor (USE/EP).
3. Add the relative address of the BLL CELLS to the relocation factor to obtain the absolute BLL CELLS address in the dump.
4. Each BLL is 4 bytes long; they are located in ascending sequence, beginning in the dump at the address computed in Step 3 BLL=1 is the first 4 bytes, BLL=2 is the second 4 bytes, etc. Find the appropriate 4 bytes.
5. The 4 bytes obtained in Step 4 contain the absolute base address of the desired record. Add the item's displacement to it to obtain the absolute address of the leftmost byte of the field in the dump.

LOCATING LAST RECORD RELEASED TO SORT/MERGE BY AN INPUT PROCEDURE

For debugging purposes, it is sometimes useful to determine the last input record released to the Sort/Merge program. The following procedure should be used:

1. From the Data Division map, determine the BLL number of the SORT/MERGE file being processed at the time of program termination. Assume it is BLLn.
2. From the Task Global Table map, determine the location of the BLL cells in the COBOL object program.
3. The nth BLL in the main storage dump will point to the last record released to SORT/MERGE.

Note: This BLL is initialized when control is first transferred to the input procedure. Thus, if the program terminates before control ever goes to the input procedure, the BLL will not be initialized. Also, with a USING clause in a SORT statement, the BLL will not be initialized.

SORT/MERGE CHECKPOINT/RESTART

The CHECKPOINT/RESTART feature is available to the programmer using the COBOL SORT/MERGE statement. In order to initiate a checkpoint, the programmer uses DD statements and the RERUN clause. The DD statement for use in taking a checkpoint is discussed in "Using the Checkpoint/Restart Feature."

The RERUN clause is used to indicate that checkpoints are written, at logical intervals determined by the sort/merge program, during the execution of all SORT/MERGE statements in the program. This RERUN clause is fully described in the publication IBM OS Full American National Standard COBOL.

EFFICIENT PROGRAM USE

The information you give the Sort/Merge program about the application it is to perform helps the sort and merge phases to produce a fast, efficient sort or merge. When you do not supply information such as data set size and record format, the program must make assumptions, which, if incorrect, lead to ineffiency.

DATA SET SIZE

The most important information one can give is an accurate data set size using the SORT-FILE-SIZE special register. If the exact number of records in the input data set is known, that number should be used as the value. If the exact number is not known, an estimate should be made. When the Sort/Merge program has accurate information about data set size, it can make the most efficient use of both main storage and intermediate storage.

MAIN STORAGE REQUIREMENTS

If the maximum amount of main storage to be used by the Sort/Merge program was not specified at installation time, the program assumes a maximum of 15,500 bytes. The sort program requests 12,000 bytes leaving 3500 bytes for system functions. Performance usually improves as the program is given more main storage. A minimum of 44K bytes of main storage is normally needed for efficient execution of the sort/merge program, and performance may increase as more main storage is made available.

If the amount of main storage was specified at system generation time, it is the programmer's responsibility to ensure that the Sort/Merge program has at least that much main storage available in addition to the space needed for Data Management and the COBOL program. If this amount of main storage is not available, the program will terminate abnormally.

The programmer may alter, dynamically within the COBOL program, the main storage default values for the Sort/Merge program. The SORT-CORE-SIZE special register may be used to communicate changes to the Sort/Merge program. In general, a positive value placed in SORT-CORE-SIZE denotes the amount of storage the programmer is allocating for use by the Sort/Merge program. For example, the statement "MOVE 30000 TO SORT-CORE-SIZE" means that 30000 bytes of storage are available to the Sort/Merge program. Accordingly, if 30000 is moved to SORT-CORE-SIZE, COBOL communicates to Sort/Merge that 30000 bytes of storage are available to it. There are, in addition, two other uses for SORT-CORE-SIZE.

Special considerations apply when a Sort/Merge program product is used. If the program product is installed with the SIZE=MAX option, the program allocates all remaining available main storage in a region for its own use. If an input procedure then attempts to open a file, an 80A abnormal termination may result if buffers and necessary data management modules have not already been loaded, since no more space is available.

If instead, a negative value is placed in the special register prior to execution of the sort, the program uses the default SIZE option specified at installation, but, if SIZE=MAX was specified, sets aside that absolute value before obtaining the remainder. Also, if ALL '9' (or +999999) is moved to SORT-CORE-SIZE prior to a sort operation, the program executes with the SIZE=MAX option, regardless of the installed value, while reserving 6K bytes of main storage for use by the data management routines and buffers.

The Sort/Merge program product may also be installed with a default reserved main storage parameter that will be used if no negative value is passed from COBOL. For additional information about these options see the appropriate Sort/Merge Installation Reference Manual.

Changing the main storage allocation can be useful when a sort/merge application is to be run in a multiprogramming environment. By reducing the amount of main storage allocated to sort/merge, so that other programs can have the storage they need to operate simultaneously, the performance of sort/merge is impaired. However, if this allocation is increased, so that a large sort/merge application runs more efficiently, the performance of other jobs sharing the multiprogramming environment is impaired, if not made altogether impossible.

SORT/MERGE DIAGNOSTIC MESSAGES

The messages generated by the Sort/Merge programs Feature are listed in the appropriate Sort/Merge Installation Reference Manual and Sort/Merge Programmer's Guide.

When the Sort/Merge program is installed, the user can elect to have messages sent to the printer, in which case a DD card with a ddname of SYSOUT must be included in the job step. The programmer can dynamically alter the ddname of the file on which Sort/Merge is to write its messages. If Sort/Merge has been installed with provision for routing its messages to the printer, then the programmer can place in the SORT-MESSAGE special register the ddname that Sort/Merge is to substitute

forSYSOUT, for message routing. For example, when the statement MOVE "SORTDDNM" TO SORT-MESSAGE is executed before sort is initiated, then the sort writes its printer messages to the data set SORTDDNM rather than to SYSOUT. If SORT-MESSAGE is not refered to in the program, the ddname that was specified at Sort/Merge installation time is the default value.

One technique for specifying the sort/merge print file ddname would be to include source language and job control language statements as follows:

- Linkage Section

```
01 SORT-PARAMETERS.
05 PARAMETER-COUNT PIC 9(4) USAGE COMP.
05 SORT-DDNAME PIC X(8).
```

- Immediately preceding the sort operation

```
IF PARAMETER-COUNT IS NOT EQUAL TO 0
   MOVE SORT-DDNAME TO SORT-MESSAGE.
```

- On the EXEC card

```
//GOSTEP EXEC PGM=program-name,
    PARM='SORTDDNM'
```

Note: This technique of assigning a unique value to SORT-MESSAGE without modifying or recompiling the program can also be applied to the special registers SORT-CORE-SIZE, SORT-MODE-SIZE, and SORT-FILE-SIZE.

DEFINING VARIABLE-LENGTH RECORDS

If the input records used are of variable length, the record length that occurs most frequently in the input data set (modal length) should be put into the special register SORT-MODE-SIZE. This value is used to help define a data set based on a particular length. If a value is not specified, the SORT/MERGE program assumes it is equal to the average of the maximum and minimum record lengths in the input data set. If, for example, the data set contains mostly small records and just a few long records, the SORT/MERGE program would assume a high modal length and would allocate a larger record storage area than necessary. Conversely, if the data set contains just a few short records and many long records, the SORT/MERGE program would assume a low modal length and might not allocate a large enough record storage area to sort data. For a complete discussion, see the appropriate Sort/Merge Programmer's Guide.

SORTING VARIABLE-LENGTH RECORDS

Figure 152 illustrates one way to sort variable-length records described by the OCCURS clause with the DEPENDING ON option. If the FD's (file-name description) and the SD's (sort-file-name description) are defined as in this figure, where the record descriptions of the FD's and the SD correspond, possibilities for error arise. It is suggested, therefore, that the user consider the following:

1. Specification of the statement

   ```
   SORT SORT-FILE USING INPUT-FILE...
   ```

   would probably lead to incorrect results. This statement implies a READ ... INTO ... statement; that is, after INPUT-FILE has been read, the record is moved to AAA. However, because the user must set the length of this receiving field prior to moving A to AAA but cannot do so, the compiler may use an incorrect length that results in abnormal termination. Instead, the user should substitute an input procedure for the USING option, as in the section of code labeled PARA2B in the example.

2. Similarly, the statement

   ```
   SORT SORT-FILE...  GIVING OUTPUT-FILE
   ```

   would probably yield incorrect results. Before OUTPUT-FILE is written out, the record is moved to AA. The correct length of this receiving field must be set before the move, but use of the GIVING option precludes this. To avoid error, the user should substitute an output procedure for the GIVING option, as in section PARA3B of the example.

3. If a SORT record contains an item with an OCCURS DEPENDING ON clause and the size of the SORT record description with the minimum number of occurences of the item represent the smallest SORT record, the minimum SORT record length is not reflected in the minimum record length parameter passed to SORT. This may result in inefficient SORT performance. The problem can be avoided by specifying a dummy SORT record of a fixed length (no OCCURS DEPENDING ON) with the size of the smallest SORT record described with OCCURS DEPENDING ON clauses.

SORT/MERGE FOR ASCII FILES

For sorting ASCII files, the normal EBCDIC collating sequence is provided unless the user specifies otherwise.

To specify a sort/merge using the ASCII collating sequence, the programmer may include and identify a program collating sequence of STANDARD-1 (equivalent to ASCII). If LANGLVL(1) is specified, the programmer may alternatively include the "C" organization entry in the ASSIGN clause for the file-name associated with the file to be sorted or merged. No buffer offset may be given with the sort/merge feature.

OTHER COLLATING SEQUENCES

Through use of the COLLATING SEQUENCE clause of the sort/merge feature, the programmer can identify other (non-EBCDIC and non-ASCII) collating sequences to be used. See "Collating Sequences" in the section "Programming Techniques."

Part 1

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  VLSORT.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT ....
    SELECT ....
    SELECT ....
DATA DIVISION.
FILE SECTION.
FD  INPUT-FILE.
    LABEL RECORDS ARE OMITTED
    DATA RECORD IS A.
01  A.
    02  B PIC 99.
    02  C OCCURS 1 TO 10 TIMES
        DEPENDING ON B.
        03  D PIC 99.
        03  E PIC XX.
FD  OUTPUT-FILE
    LABEL RECORDS ARE OMITTED
    DATA RECORD IS AA.
01  AA.
    02  BB PIC 99.
    02  CC OCCURS 1 TO 10 TIMES
        DEPENDING ON BB.
        03  DD PIC 99.
        03  EE PIC XX.
SD  SORT-FILE
    DATA RECORD IS AAA.
01  AAA.
    02  BBB PIC 99.
    02  CCC OCCURS 1 TO 10 TIMES
        DEPENDING ON BBB.
        03  DDD PIC 99.
        03  EEE PIC XX.
```

Part 2

```
PROCEDURE DIVISION.
PAR1 SECTION.
      SORT SORT-FILE ASCENDING KEY BBB
    INPUT PROCEDURE PAR2
    OUTPUT PROCEDURE PAR3.
           STOP RUN.
PAR2 SECTION.
PAR2A.
    OPEN INPUT INPUT-FILE.
PAR2B.
    READ INPUT-FILE AT END GO TO PAR2C.
    MOVE B TO BBB.
    RELEASE AAA FROM A.[1]
    GO TO PAR2B.
PAR2C.
    CLOSE INPUT-FILE.
PAR2-EXIT.
    EXIT.
PAR3 SECTION.
PAR3A.
    OPEN OUTPUT OUTPUT-FILE.
PAR3B.
    RETURN SORT-FILE AT END GO TO PAR3C.[2]
    MOVE BBB TO BB.
    WRITE AA FROM AAA.
    GO TO PAR3B.
PAR3C.
    CLOSE OUTPUT-FILE.
PAR3-EXIT.
    EXIT.
```

[1] When using a sort input procedure, the RELEASE ... FROM clause, which implies a MOVE and then a RELEASE, should always be preceded by a MOVE that sets the length of the receiving field (AAA, in this example).

[2] When using a sort output procedure, the RETURN ... INTO ... clause, which implies the RETURN and then a MOVE, should <u>never</u> be used. There is no way for the user to set the correct length of the receiving field.

Figure 152. Sorting Variable-Length Records Whose File-name Description and Sort-File-name Description Correspond

A DEBUG control statement may be included at execution time to assist in debugging Sort/Merge problems. The ABEND operand of the DEBUG statement can be used to override the NOABEND default of the installation keyword ERRET. The format of the COBOL SORT-DEBUG statement is:

```
bDEBUGb operands
```

The operands and their meanings are as follows:

{ABEND / NOABEND}	Overrides the generated default for action to be taken when the program encounters an uncorrectable error.
CLOCK	Instructs the program to measure elapsed and CPU times for the different PEER phases.
FLAG (x+ [,x'...,x°])	Instructs the program to print PEER information messages (ICE120-124). The values which may be given to x are as follows: @ Messages from all phases (ICE120-124) O Phase 0 messages ICE120) C Phase 1 messages (ICE121) P Phase 2 (partition) messages (ICE124) R Phase 2 (reduction) messages (ICE122) E Phase 3 messages (ICE123)
CTRO=value	The program should keep a count of work I/O operations; when the count reaches "value", it should ABEND.
EM=value	"Value" should be used as the maximum number of strings to be merged in the final merge pass.
RM=value	"Value" should be used as the maximum number of strings to be merged in intermediate merge passes.
CB=value	"Value" is the number of Phase 1 work buffers.
RB=value	"Value" is the number of Phase 2 work buffers.
EB=value	"Value" is the number of Phase 3 work buffers.
BT=value	Instructs the program to calculate the blocking factor for intermediate storage in such a way that "value"is the number of buffers per track.

For more detailed information on the DEBUG feature and the messages generated by the use of the above operands, see the publication OS/VS Sort/Merge Programmer's Guide.

Operands must be separated by commas. Operands may be continued on a second card either by following the last operand on card 1 by a comma and continuing on card 2 in columns 2-16, or by having a nonblank character in column 72 of card 1 and continuing in column 16 of card 2. Comments may be continued on a second card by having a nonblank character in column 72 of card 1 and continuing in column 16 of card 2.

Use of the COBOL SORT-DEBUG statement requires the following DD statement:

```
//SRTCDS DD *
    DEBUG Statement
```

USING THE SEGMENTATION FEATURE

Although Release 2 of the OS/VS COBOL compiler will accept a source program containing segmentation specifications, it will not produce an actual overlay structure. Instead, it combines all segments into one single object program in segment order, and allows the paging of the VS operating system to perform any overlay. The absence of actual COBOL-performed overlay is usually not a problem in the OS/VS environment, since adequate main storage is available for even the largest programs.

The following discussion is provided only for those users who--while recognizing that this compiler will not provide overlay--nevertheless wish to write or maintain programs that include segmentation statements.

Segmentation provides a means of dividing the Procedure Division of a source program into sections. Through the use of a system of priority numbers, certain sections are designated as fixed segments (either fixed permanent or fixed overlayable) and others as independent segments.

Suppose that the program SAVECORE is segmented as shown in Figure 153. Only those segments having priority numbers less than the segment limit of 15 are designated as fixed permanent. Sections in the fixed permanent segment (SECTION-1, SECTION-2, and SECTION-4) are those that must be available for reference at all times, or those to which reference is made frequently. They are distinguished here by the fact that they have been assigned priority numbers less than the program's segment limit.

Fixed overlayable segments are sections that are less frequently used. These sections are sometimes made available in the state in which they were last used (see IBM VS COBOL for OS/VS). They are distinguishable here by the fact that they have been assigned priority numbers greater than the segment limit, but less than 50.

Independent segments are those assigned priority numbers greater than 49 and less than 100 (section-5 and section-7 in this example).

```
IDENTIFICATION DIVISION.

PROGRAM-ID.  SAVECORE.
.
.
.

ENVIRONMENT DIVISION.

OBJECT-COMPUTER.  IBM-370
     SEGMENT-LIMIT IS 15.
.
.
.
DATA DIVISION.
.
.
.
PROCEDURE DIVISION.
SECTION-1  SECTION 8.
.
.
.
SECTION-2  SECTION 8.
.
.
.
SECTION-3  SECTION 16.
.
.
.
SECTION-4  SECTION 8.
.
.
.
SECTION-5  SECTION 50.
.
.
.
SECTION-6  SECTION 16.
.
.
.
SECTION-7  SECTION 50.
.
.
.
```

Figure 153. Segmentation of Program SAVECORE

USING THE PERFORM STATEMENT IN A SEGMENTED PROGRAM

When the PERFORM statement is used in a segmented program, the programmer should be aware of the following:

- A PERFORM statement that appears in a section whose priority-number is less than the segment limit can have within its range only (a) sections with priority-numbers less than 50, and (b) sections wholly contained in a single segment whose priority-number is greater than 49.

  Note: As an extension to American National Standard COBOL, the OS/VS COBOL Compiler allows sections with any priority-number to fall within the range of a PERFORM statement.

- A PERFORM statement that appears in a section whose priority-number is equal to or greater than the segment limit can have within its range only (a)

sections with the same priority-number as the section containing the PERFORM statement, and (b) sections with priority-numbers that are less than the segment limit.

Note: As an extension to American National Standard COBOL, the OS/VS COBOL Compiler allows sections with any priority-number to fall within the range of a PERFORM statement.

- When a procedure-name in a segment with a priority-number less than the segment limit referred to by a PERFORM statement in a segment with a priority-number greater than the segment limit, the independent segment will be reinitialized upon exit from the PERFORM.

OPERATION

Execution of the object program begins in the root segment; i.e., the first segment in the permanent segment. If the program contains no permanent segments, or if the first section to be executed in the program is not part of the root segment, the compiler generates a dummy segment thatwill initiate the execution of the first independent segment. All global tables, literals, and data areas are part of the root segment. Called object-time subroutines are also part of the root segment. Called subprograms are loaded with the fixed portion of the main program and assigned a priority of zero. Otherwise, the program executes just as if it were not segmented.

LANGLVL OPTION AND RE-INITIALIZATION

The LANGLVL compile option chosen by the user affects the degree and manner of re-initialization COBOL will perform on independent segments, since there is a difference between the 1968 and 1974 ANS definitions. For further details, consult the language manual IBM VS COBOL for OS/VS.

COMPILER OUTPUT

The output produced by the compiler is a group of segments organized by priority number. Segments whose priority is greater than the segment limit (or 49, if no SEGMENT-LIMIT clause is specified) consist of executable instructions only. The PMAP output is given in this sequence: the root segment first, followed by all other segments in ascending order by priority number.

Figure 154 shows the output of a sample segmentation program.

```
00001   00006U IDENTIFICATION DIVISION.                                        00154790
00002   00007U PROGRAM-ID.  SEG-SAMPLE.
00003   00008U AUTHOR.  PROGRAMMER-NAME.
00004   00009U REMARKS.                                                        00154820
00005   00010U     SPECIAL OPERATOR INSTRUCTIONS - NONE.                       00154830
00006   00011U     INPUT REQUIRED - NONE.                                      00154840
00007   00012U     PURPOSE                                                     00154850
00008   00013U         TO CREATE A SINGLE FILE ON DISK USING                   00154860
00009   00014U         QSAM/DTFSD, AND READ  IT BACK.                          00154870
00010   00015U         PROGRAM USES SEGMENTATION                               00154880
00011   00016U         WITH FILE PROCESSING SPREAD OVER                        00154890
00012   00017U         THE PERMANENT, OVERLAYABLE FIXED,                       00154900
00013   00018U         AND INDEPENDENT SEGMENTS.                               00154910
00014   00019U     EXPECTED RESULTS                                            00154920
00015   00020U         START TEST SEG-SAMPLE
00016   00021U         (EACH SEGMENT DISPLAYS ITS SEGMENT NUMBER               00154940
00017   00022U         AND FUNCTION)                                           00154950
00018   00023U         END TEST SEG-SAMPLE SUCCESSFUL RUN
00019   00024U         SECTIONS WHILE WRITING APPEAR                           00154970
00020   00025U         IN ORDER 80, 20, 30, 60, 40.                            00154980
00021   00026U         SECTIONS WHILE READING APPEAR                           00154990
00022   00027U         IN ORDER 80, 60, 30, 40, 20.                            00155000
00023   00028U     ERROR INDICATIONS                                           00155010
00024   00029U         **ERROR DISK SEQ I/O**                                  00155020
00025   00030U         **ERROR END OF EXTENT WRITING AFTER (RECORD)**          00155030
00026   00031U         **ERROR UNEXPECTED EOF READING AFTER                    00155040
00027   00032U             RECORD (RECNO)**                                    00155050
00028   00033U         **ERROR EOF NOT FOUND**                                 00155060
00029   00034U         **RECORD IS (RECNO)                                     00155070
00030   00035U           SHOULD BE (RECNO)**                                   00155080
00031   00038U         PROGRAM CONTAINS PERFORMS FROM BASE SECTION             00155110
00032   00039U         TO PERMANENT, OVERLAYABLE FIXED, AND INDEPENDENT        00155120
00033   00040U         SEGMENTS.                                               00155130
00034   00041U         ALSO CONTAINS PERFORMS FROM INDEPENDENT TO PERMANENT    00155140
00035   00042U         AND FROM OVERLAYABLE FIXED TO PERMANENT SEGMENTS.       00155150
00036   00043U         ALSO CONTAINS PERFORMS ENTIRELY WITHIN A SEGMENT IN     00155160
00037   00044U         IN EACH CATEGORY.                                       00155170
00038   00045U ENVIRONMENT DIVISION.                                           00155180
00039   00046U CONFIGURATION SECTION.                                          00155190
00040   00047U SOURCE-COMPUTER. IBM-370.                                       00155200
00041   00048U OBJECT-COMPUTER. IBM-370                                        00155210
00042   00049U     MEMORY SIZE 64000 CHARACTERS                                00155220
00043   00050U     SEGMENT-LIMIT IS 25.                                        00155230
00044   00051U INPUT-OUTPUT SECTION.                                           00155240
00045   00052U FILE-CONTROL.                                                   00155250
00046   00053U     SELECT FILE-1 ASSIGN TO        DA-2314-S-DKSQ01A.           00155260
00047   00054U DATA DIVISION.                                                  00155270
00048   00055U FILE SECTION.                                                   00155280
00049   00056U FD  FILE-1                                                      00155290
00050   00057U     RECORDING MODE IS F                                         00155300
00051   00058U     LABEL RECORDS OMITTED
00052   00059U     DATA RECORD IS RECFD1.                                      00155310
00053   00060U 01  RECFD1 PICTURE X(83).                                       00155320
00054   00061U WORKING-STORAGE SECTION.                                        00155330
00055   00062U 77  ERRORSW PIC A VALUE SPACE.                                 00155340
00056   00063U 77  ERCTFL  PIC S99 VALUE ZERO.                                00155350
00057   00064U 77  MSGHDR  PIC X(22) VALUE '**ERROR DISK SEQ I/O**'.          00155360
```

Figure 154. Sample Segmentation Program (Part 1 of 14)

```
000058  000650 77  MSGEOX PIC X(36)                                          00155370
000059  000660     VALUE '**ERROR END OF EXTENT WRITING AFTER '.              00155380
000060  000670 77  MSGEOF  PIC X(37)                                         00155390
000061  000680     VALUE '**ERROR UNEXPECTED EOF READING AFTER '.             00155400
000062  000690 77  MSGNEF  PIC X(23) VALUE '**ERROR EOF NOT FOUND**'.        00155410
000063  000700 01  REC1.                                                     00155420
000064  000710     02  REC-ID.                                               00155430
000065  000720     03  REC-HD  PIC X(4) VALUE 'RECD'.                        00155440
000066  000730     03  REC-NO  PIC S9(4) VALUE ZERO.                         00155450
000067  000740     02  FILLER  PIC A(75) VALUE SPACES.                       00155460
000068  000750     66  RECID RENAMES REC-ID.                                 00155470
000069  000760 01  VER-REC.                                                  00155480
000070  000770     02  VER-ID.                                               00155490
000071  000780     03  VER-HD  PIC X(4) VALUE 'RECD'.                        00155500
000072  000790     03  VER-NO  PIC S9(4) VALUE ZERO.                         00155510
000073  000800 PROCEDURE DIVISION.                                          00155520
000074  000810 BASE-SECTION SECTION 0.                                      00155530
000075  000820     DISPLAY 'START TEST SEG-SAMPLE'.
000076  000830     OPEN OUTPUT FILE-1.                                      00155550
000077  000840     PERFORM W-80-0 THRU W-80-9.                              00155560
000078  000850     PERFORM W-30-0 THRU W-30-9.                              00155570
000079  000860     PERFORM W-60-0 THRU W-60-9.                              00155580
000080  000870     PERFORM W-40-0 THRU W-40-9.                              00155590
000081  000880 BASE-50.                                                     00155600
000082  000890     CLOSE FILE-1.                                            00155610
000083  000900     OPEN INPUT FILE-1.                                       00155620
000084  000910     PERFORM R-80-0 THRU R-80-9.                              00155630
000085  000920     GO TO R-60-0.                                            00155640
000086  000930 BASE-60.                                                    00155650
000087  000940     PERFORM R-40-0 THRU R-40-9.                              00155660
000088  000950     READ FILE-1 INTO REC1 AT END GO TO BASE-70.              00155670
000089  000960     DISPLAY MSGHDR DISPLAY MSGNEF                            00155680
000090  000970     MOVE 'E' TO ERRORSW.                                     00155690
000091  000980 BASE-70.                                                    00155700
000092  000990     CLOSE FILE-1.                                            00155710
000093  001000 BASE-90.                                                    00155720
000094  001010     IF ERRORSW IS EQUAL TO 'E'                               00155730
000095  001020     DISPLAY 'END TEST SEG-SAMPLE UNSUCCESSFUL RUN' ELSE
000096  001030     DISPLAY 'END TEST SEG-SAMPLE SUCCESSFUL RUN'.
000097  001040     STOP RUN.                                                00155760
000098  001050 SECTION-20 SECTION 20.                                      00155770
000099  001060 W-20-0.                                                     00155780
000100  001070     DISPLAY 'SECTION 20 WRITE'.                              00155790
000101  001080     NOTE ENTERED BY PERFORM FROM W-80-0.                     00155800
000102  001090     PERFORM W-21-0 THRU W-21-9 5 TIMES.                      00155810
000103  001100 W-20-9.                                                     00155820
000104  001110     EXIT.                                                    00155830
000105  001120 W-21-0.                                                     00155840
000106  001130     WRITE RECFD1 FROM REC1 INVALID KEY                       00155850
000107  001140     DISPLAY MSGHDR                                           00155860
000108  001150     DISPLAY MSGEOX RECID                                     00155870
000109  001160     MOVE 'E' TO ERRORSW                                      00155880
000110  001170     GO TO BASE-90.                                           00155890
000111  001180     ADD 0001 TO REC-NO.                                      00155900
000112  001190 W-21-9.                                                     00155910
000113  001200     EXIT.                                                    00155920
000114  001210 R-20-0.                                                     00155930
```

Figure 154. Sample Segmentation Program (Part 2 of 14)

```
00115   00122U      DISPLAY 'SECTION 20 READ'.                                    00155940
00116   00123U      NOTE ENTERED BY PERFORM FROM BASE-40.                         00155950
00117   00124U      PERFORM R-21-0 THRU R-21-9 5 TIMES.                           00155960
00118   00125U R-20-9.                                                            00155970
00119   00126U      EXIT.                                                         00155980
00120   00127U R-21-0.                                                            00155990
00121   00128U      READ FILE-1 INTO REC1 AT END                                  00156000
00122   00129U      DISPLAY MSGHDR DISPLAY MSGEOF                                 00156010
00123   00130U      ADD 4 TO ERCTFL MOVE 'E' TO ERRORSW                           00156020
00124   00131U      GO TO R-21-9.                                                 00156030
00125   00132U      IF REC-ID IS NOT EQUAL TO VER-ID                              00156040
00126   00133U      DISPLAY MSGHDR DISPLAY 'EXPECTED ' VER-ID ' FOUND ' REC-ID    00156050
00127   00134U      ADD 1 TO ERCTFL MOVE 'E' TO ERRORSW                           00156060
00128   00135U      MOVE REC-ID TO VER-ID.                                        00156070
00129   00136U      ADD 1 TO VER-NO.                                              00156080
00130   00137U R-21-9.                                                            00156090
00131   00138U      IF ERCTFL IS GREATER THAN 3                                   00156100
00132   00139U      GO TO BASE-70.                                                00156110
00133   00140U SECTION-30 SECTION 30.                                             00156120
00134   00141U W-30-0.                                                            00156130
00135   00142U      DISPLAY 'SECTION 30 WRITE'.                                   00156140
00136   00143U      NOTE ENTERED BY PERFORM FROM BASE-SECTION.                    00156150
00137   00144U      PERFORM W-31-0 THRU W-31-9 11 TIMES.                          00156160
00138   00145U W-30-9.                                                            00156170
00139   00146U      EXIT.                                                         00156180
00140   00147U W-31-0.                                                            00156190
00141   00148U      WRITE RECFD1 FROM REC1 INVALID KEY                            00156200
00142   00149U      DISPLAY MSGHDR                                                00156210
00143   00150U      DISPLAY MSGEOX RECID                                          00156220
00144   00151U      MOVE 'E' TO ERRORSW                                           00156230
00145   00152U      GO TO BASE-50.                                                00156240
00146   00153U      ADD 0001 TO REC-NO.                                           00156250
00147   00154U W-31-9.                                                            00156260
00148   00155U      EXIT.                                                         00156270
00149   00156U R-30-0.                                                            00156280
00150   00157U      DISPLAY 'SECTION 30 READ'.                                    00156290
00151   00158U      NOTE ENTERED BY GO TO FROM R-60-0.                            00156300
00152   00159U      PERFORM R-31-0 THRU R-31-9 11 TIMES.                          00156310
00153   00160U      GO TO BASE-60.                                                00156320
00154   00161U R-31-0.                                                            00156330
00155   00162U      READ FILE-1 INTO REC1 AT END                                  00156340
00156   00163U      DISPLAY MSGHDR DISPLAY MSGEOF                                 00156350
00157   00164U      ADD 4 TO ERCTFL MOVE 'E' TO ERRORSW                           00156360
00158   00165U      GO TO R-31-9.                                                 00156370
00159   00166U      IF REC-ID IS NOT EQUAL TO VER-ID                              00156380
00160   00167U      DISPLAY MSGHDR DISPLAY 'EXPECTED ' VER-ID ' FOUND ' REC-ID    00156390
00161   00168U      ADD 1 TO ERCTFL MOVE 'E' TO ERRORSW                           00156400
00162   00169U      MOVE REC-ID TO VER-ID.                                        00156410
00163   00170U      ADD 1 TO VER-NO.                                              00156420
00164   00171U R-31-9.                                                            00156430
00165   00172U      IF ERCTFL IS GREATER THAN 3                                   00156440
00166   00173U      GO TO BASE-70.                                                00156450
00167   00174U SECTION-40 SECTION 40.                                             00156460
00168   00175U W-40-0.                                                            00156470
00169   00176U      DISPLAY 'SECTION 40 WRITE'.                                   00156480
00170   00177U      NOTE ENTERED BY PERFORM FROM BASE-SECTION.                    00156490
00171   00178U      PERFORM W-41-0 THRU W-41-9 17 TIMES.                          00156500
```

Figure 154. Sample Segmentation Program (Part 3 of 14)

```
00172   001790 W-40-9.                                                  00156510
00173   001800     EXIT.                                                00156520
00174   001810 W-41-0.                                                  00156530
00175   001820     WRITE RECFD1 FROM REC1 INVALID KEY                   00156540
00176   001830     DISPLAY MSGHDR                                       00156550
00177   001840     DISPLAY MSGEOX RECID                                 00156560
00178   001850     MOVE 'E' TO ERRORSW                                  00156570
00179   001860     GO TO BASE-50.                                       00156580
00180   001870     ADD 0001 TO REC-NO.                                  00156590
00181   001880 W-41-9.                                                  00156600
00182   001890     EXIT.                                                00156610
00183   001900 R-40-0.                                                  00156620
00184   001910     DISPLAY 'SECTION 40 READ'.                           00156630
00185   001920     NOTE ENTERED BY PERFORM FROM BASE-60.                00156640
00186   001930     PERFORM R-41-0 THRU R-41-0 7 TIMES.                  00156650
00187   001940     PERFORM R-20-0 THRU R-20-9.                          00156660
00188   001950 R-40-9.                                                  00156670
00189   001960     EXIT.                                                00156680
00190   001970 R-41-0.                                                  00156690
00191   001980     READ FILE-1 INTO REC1 AT END                         00156700
00192   001990     DISPLAY MSGHDR DISPLAY MSGEOF                        00156710
00193   002000     ADD 4 TO ERCTFL MOVE 'E' TO ERRORSW                  00156720
00194   002010     GO TO R-41-9.                                        00156730
00195   002020     IF REC-ID IS NOT EQUAL TO VER-ID                     00156740
00196   002030     DISPLAY MSGHDR DISPLAY 'EXPECTED ' VER-ID ' FOUND ' REC-ID   00156750
00197   002040     ADD 1 TO ERCTFL MOVE 'E' TO ERRORSW                  00156760
00198   002050     MOVE REC-ID TO VER-ID.                               00156770
00199   002060     ADD 1 TO VER-NO.                                     00156780
00200   002070 R-41-9.                                                  00156790
00201   002080     IF ERCTFL IS GREATER THAN 3                          00156800
00202   002090     GO TO BASE-70.                                       00156810
00203   002100 SECTION-60 SECTION 60.                                   00156820
00204   002110 W-60-0.                                                  00156830
00205   002120     DISPLAY 'SECTION 60 WRITE'.                          00156840
00206   002130     NOTE ENTERED BY PERFORM FROM BASE-SECTION.           00156850
00207   002140     PERFORM W-61-0 THRU W-61-9 13 TIMES.                 00156860
00208   002150 W-60-9.                                                  00156870
00209   002160     EXIT.                                                00156880
00210   002170 W-61-0.                                                  00156890
00211   002180     WRITE RECFD1 FROM REC1 INVALID KEY                   00156900
00212   002190     DISPLAY MSGHDR                                       00156910
00213   002200     DISPLAY MSGEOX RECID                                 00156920
00214   002210     MOVE 'E' TO ERRORSW                                  00156930
00215   002220     GO TO BASE-50.                                       00156940
00216   002230     ADD 0001 TO REC-NO.                                  00156950
00217   002240 W-61-9.                                                  00156960
00218   002250     EXIT.                                                00156970
00219   002260 R-60-0.                                                  00156980
00220   002270     DISPLAY 'SECTION 60 READ'.                           00156990
00221   002280     NOTE ENTERED BY GO TO FROM BASE-50.                  00157000
00222   002290     PERFORM R-61-0 THRU R-61-9 13 TIMES.                 00157010
00223   002300     GO TO R-30-0.                                        00157020
00224   002310 R-61-0.                                                  00157030
00225   002320     READ FILE-1 INTO REC1 AT END                         00157040
00226   002330     DISPLAY MSGHDR DISPLAY MSGEOF                        00157050
00227   002340     ADD 4 TO ERCTFL MOVE 'E' TO ERRORSW                  00157060
00228   002350     GO TO R-61-9.                                        00157070
```

Figure 154. Sample Segmentation Program (Part 4 of 14)

```
00229   00236U      IF REC-ID IS NOT EQUAL TO VER-ID                              00157080
00230   00237U      DISPLAY MSGHDR DISPLAY 'EXPECTED ' VER-ID ' FOUND ' REC-ID    00157090
00231   00238U      ADD 1 TO ERCTFL MOVE 'E' TO ERRORSW                           00157100
00232   00239U      MOVE REC-ID TO VER-ID.                                        00157110
00233   00240U      ADD 1 TO VER-NO.                                              00157120
00234   00241U R-61-9.                                                            00157130
00235   00242U      IF ERCTFL IS GREATER THAN 3                                   00157140
00236   00243U      GO TO BASE-70.                                                00157150
00237   00244U SECTION-80 SECTION 80.                                             00157160
00238   00245U W-80-0.                                                            00157170
00239   00246U      DISPLAY 'SECTION 80 WRITE'.                                   00157180
00240   00247U      NOTE ENTERED BY PERFORM FROM BASE-SECTION.                    00157190
00241   00248U      PERFORM  W-81-0  THRU W-81-9 7 TIMES.                         00157200
00242   00249U      PERFORM W-20-0 THRU W-20-9.                                   00157210
00243   00250U W-80-9.                                                            00157220
00244   00251U      EXIT.                                                         00157230
00245   00252U W-81-0.                                                            00157240
00246   00253U      WRITE RECFD1 FROM REC1 INVALID KEY                            00157250
00247   00254U      DISPLAY MSGHDR                                                00157260
00248   00255U      DISPLAY MSGBOX RECID                                          00157270
00249   00256U      MOVE 'E' TO ERRORSW                                           00157280
00250   00257U      GO TO BASE-50.                                                00157290
00251   00258U      ADD 0001 TO REC-NO.                                           00157300
00252   00259U W-81-9.                                                            00157310
00253   00260U      EXIT.                                                         00157320
00254   00261U R-80-0.                                                            00157330
00255   00262U      DISPLAY 'SECTION 80 READ'.                                    00157340
00256   00263U      NOTE ENTERED BY PERFORM FROM BASE-50.                         00157350
00257   00264U      PERFORM R-81-0 THRU R-81-9 17 TIMES.                          00157360
00258   00265U R-80-9.                                                            00157370
00259   00266U      EXIT.                                                         00157380
00260   00267U R-81-0.                                                            00157390
00261   00268U      READ FILE-1 INTO REC1 AT END                                  00157400
00262   00269U      DISPLAY MSGHDR DISPLAY MSGEOF                                 00157410
00263   00270U      ADD 4 TO ERCTFL MOVE 'E' TO ERRORSW                           00157420
00264   00271U      GO TO R-81-9.                                                 00157430
00265   00272U      IF REC-ID IS NOT EQUAL TO VER-ID                              00157440
00266   00273U      DISPLAY MSGHDR DISPLAY 'EXPECTED ' VER-ID ' FOUND ' REC-ID    00157450
00267   00274U      ADD 1 TO ERCTFL MOVE 'E' TO ERRORSW                           00157460
00268   00275U      MOVE REC-ID TO VER-ID.                                        00157470
00269   00276U      ADD 1 TO VER-NO.                                              00157480
00270   00277U R-81-8.
00271   00278U      IF ERCTFL IS GREATER THAN 3                                   00157500
00272   00279U      GO TO BASE-70.                                                00157510
00273   00280U R-81-9.
00274   00281U      EXIT.
```

INTRNL NAME	LVL	SOURCE NAME	BASE	DISPL	INTRNL NAME	DEFINITION	USAGE	R	O	Q	M
DNM=2-234	FD	FILE-1	DCB=01		DNM=2-234		QSAM				F
DNM=2-254	01	RECFD1	BL=1	000	DNM=2-254	DS 83C	DISP				
DNM=2-273	77	ERRORSW	BL=2	000	DNM=2-273	DS 1C	DISP				
DNM=2-293	77	ERCTFL	BL=2	001	DNM=2-293	DS 2C	DISP-NM				
DNM=2-309	77	MSGHDR	BL=2	003	DNM=2-309	DS 22C	DISP				
DNM=2-325	77	MSGEOX	BL=2	019	DNM=2-325	DS 36C	DISP				
DNM=2-341	77	MSGEOF	BL=2	03D	DNM=2-341	DS 37C	DISP				
DNM=2-357	77	MSGNEF	BL=2	062	DNM=2-357	DS 23C	DISP				
DNM=2-373	01	REC1	BL=2	080	DNM=2-373	DS 0CL83	GROUP				
DNM=2-390	02	REC-ID	BL=2	080	DNM=2-390	DS 0CL8	GROUP				
DNM=2-409	03	REC-HD	BL=2	080	DNM=2-409	DS 4C	DISP				
DNM=2-425	03	REC-NO	BL=2	084	DNM=2-425	DS 4C	DISP-NM				
DNM=2-441	02	FILLER	BL=2	088	DNM=2-441	DS 75C	DISP				
DNM=2-452	66	RECID	BL=2	080	DNM=2-452	DS 0CL8	GROUP				
DNM=2-470	01	VER-REC	BL=2	0D8	DNM=2-470	DS 0CL8	GROUP				
DNM=2-490	02	VER-ID	BL=2	0D8	DNM=2-490	DS 0CL8	GROUP				
DNM=3-000	03	VER-HD	BL=2	0D8	DNM=3-000	DS 4C	DISP				
DNM=3-016	03	VER-NO	BL=2	0DC	DNM=3-016	DS 4C	DISP-NM				

Figure 154. Sample Segmentation Program (Part 5 of 14)

```
          MEMORY MAP

     TGT                        00290

SAVE AREA                       00290
SWITCH                          002D8
TALLY                           002DC
SORT SAVE                       002E0
ENTRY-SAVE                      002E4
SORT CORE SIZE                  002E8
RET CODE                        002EC
SORT RET                        002EE
WORKING CELLS                   002F0
SORT FILE SIZE                  00420
SORT MODE SIZE                  00424
PGT-VN TBL                      00428
TGT-VN TBL                      0042C
RESERVED                        00430
LENGTH OF VN TBL                00434
LABEL RET                       00436
RESERVED                        00437
DBG R14SAVE                     00438
COBOL INDICATOR                 0043C
A(INIT1)                        00440
DEBUG TABLE PTR                 00444
SUBCOM PTR                      00448
SORT-MESSAGE                    0044C
SYSOUT DDNAME                   00454
RESERVED                        00455
COBOL ID                        00456
COMPILED POINTER                00458
COUNT TABLE ADDRESS             0045C
RESERVED                        00460
DBG R11SAVE                     00468
COUNT CHAIN ADDRESS             0046C
PRBL1 CELL PTR                  00470
RESERVED                        00474
TA LENGTH                       00479
RESERVED                        0047C
PCS LIT PTR                     00484
DEBUGGING                       00488
CD FOR INITIAL INPUT            0048C
OVERFLOW CELLS                  00490
BL CELLS                        00490
DECBADR CELLS                   00498
FIB CELLS                       00498
TEMP STORAGE                    004A0
TEMP STORAGE-2                  004A8
TEMP STORAGE-3                  004A8
TEMP STORAGE-4                  004A8
BLL CELLS                       004A8
VLC CELLS                       004B0
SBL CELLS                       004B0
INDEX CELLS                     004B0
SUBADR CELLS                    004B0
ONCTL CELLS                     004B0
PFMCTL CELLS                    004B0
PFMSAV CELLS                    004D8
VN CELLS                        00520
SAVE AREA =2                    005B0
SAVE AREA =3                    005B0
XSASW CELLS                     005B8
XSA CELLS                       005B8
PARAM CELLS                     005B8
RPTSAV AREA                     005B8
CHECKPT CTR                     005B8
```

Figure 154. Sample Segmentation Program (Part 6 of 14)

```
LITERAL POOL (HEX)

006A8 (LIT+0)     1C4C3C10  00000010  00000000  48140000  00044000  001C0005
006C0 (LIT+24)    08000000  00080011  0007000D

     DISPLAY LITERALS (BCD)

006CC (LTL+30)   'START TEST SEG-SAMPLEEND TEST SEG-SAMPLE UNSUCCESSFUL RU'
00704 (LTL+98)   'NEND TEST SEG-SAMPLE SUCCESSFUL RUNSECTION 20 WRITESECTI'
00738 (LTL+148)  'ON 20 READEXPECTED  FOUND SECTION 30 WRITESECTION 30 REA'
00774 (LTL+204)  'DSECTION 40 WRITESECTION 40 READSECTION 60 WRITESECTION '
007AC (LTL+260)  '60 READSECTION 80 WRITESECTION 80 READ'


                    PGT                     005C0

               DEBUG LINKAGE AREA           005C0
               OVERFLOW CELLS               005C0
               VIRTUAL CELLS                005C4
               PROCEDURE NAME CELLS         005E4
               GENERATED NAME CELLS         005E4
               DCB ADDRESS CELLS            00610
               VNI CELLS                    00614
               LITERALS                     006A8
               DISPLAY LITERALS             006CC
               PROCEDURE BLOCK CELLS        007D4


  REGISTER ASSIGNMENT

   REG 6      BL =2
   REG 7      BL =1

WORKING-STORAGE STARTS AT LOCATION 000A0 FOR A LENGTH OF 000E0.

  PROCEDURE BLOCK ASSIGNMENT

   PBL = REG 11

   PBL =1    STARTS AT LOCATION 0007D8  STATEMENT 75      SEGMENT = ROOT
```

Figure 154. Sample Segmentation Program (Part 7 of 14)

```
          ***************ROOT SEGMENT****************

                  0007D8                          PN=01     EQU   *
74      *BASE-SECTION
                  0007D8                          PN=02     EQU   *
75        DISPLAY 0007D8                          START     EQU   *
                  0007D8  58 B0 C 214                       L     11,214(0,12)          PBL=1
                  0007DC  58 F0 C 00C                       L     15,00C(0,12)          V(ILBODSS0)
                  0007E0  05 1F                             BALR  1,15
                  0007E2  0001                              DC    X'0001'
                  0007E4  10                                DC    X'10'
                  0007E5  000015                            DC    X'000015'
                  0007E8  0C000010C                         DC    X'0C000010C'          LIT+36
                  0007EC  0000                              DC    X'0000'
                  0007EE  FFFF                              DC    X'FFFF'
76        OPEN      0007F0  58 10 C 050                     L     1,050(0,12)           DCB=1
                     .
                     .
                     .
                  000D48  58 F0 C 01C                       L     15,01C(0,12)          V(ILBOGD01)
                  000D4C  05 EF                             BALR  14,15
                  000D4E  47 F0 B 57A                       BC    15,57A(0,11)          PN=016
                     .
                     .
                     .
                  000000  90 EC D 00C             INIT1     STM   14,12,00C(13)
                  000004  18 5D                             LR    5,13
                  000006  05 F0                             BALR  15,0
                  000008  45 80 F 010                       BAL   8,010(0,15)
                  00000C  E2C5C7F0E2C1D4D7                  DC    X'E2C5C7F0E2C1D4D7'
                  000014  E5E2D9F1                          DC    X'E5E2D9F1'
                  000018  07 00                             BCR   0,0
                  00001A  98 9F F 024                       LM    9,15,024(15)
                  00001E  07 FF                             BCR   15,15
                  000020  96 02 1 034                       OI    034(1),X'02'
                  000024  07 FE                             BCR   15,14
                  000026  41 F0 0 001                       LA    15,001(0,0)
                  00002A  07 FE                             BCR   15,14
                  00002C  0016EE                            ADCON L4(INIT3)
                  000030  00000000                          ADCON L4(INIT1)
                  000034  00000000                          ADCON L4(INIT1)
                  000038  0005C0                            ADCON L4(PGT)
                  00003C  00000290                          ADCON L4(TGT)
                  000040  0007D8                            ADCON L4(START)
                  000044  0016AE                            ADCON L4(INIT2)
                  000048                                    DS    15F
                  000084  00000000                          DC    X'00000000'
                  000088  F1F74BF4F64BF3F6                  DC    X'F1F74BF4F64BF3F6'
                  000090  C1E4C74040F46B40                  DC    X'C1E4C74040F46B40'
                  000098  F1F9F7F6                          DC    X'F1F9F7F6'
```

Figure 154. Sample Segmentation Program (Part 8 of 14)

```
         ***************SEGMENT OF PTY 30****************

133      *SECTION-30
                    000D52                          PN=016    EQU   *
134      *W-30-0
                    000D52                          PN=017    EQU   *
135       DISPLAY   000D52  58 F0 C 00C                       L     15,00C(0,12)          V(ILBODSS0)
                    000D56  05 1F                             BALR  1,15
                    000D58  0001                              DC    X'0001'
                    000D5A  10                                DC    X'10'
                    000D5B  000010                            DC    X'000010'
                    000D5E  0C000196                          DC    X'0C000196'           LIT+174
                    000D62  0000                              DC    X'0000'
                    000D64  FFFF                              DC    X'FFFF'
137       PERFORM   000D66  D2 03 D 268 D 2B8                 MVC   268(4,13),2B8(13)     PSV=9           VN=06
                      .
                      .
                      .
                    000F90  58 F0 C 01C                       L     15,01C(0,12)          V(ILBOGD01)
                    000F94  05 EF                             BALR  14,15
                    000F96  47 F0 B 7C2                       BC    15,7C2(0,11)          PN=024

         ***************SEGMENT OF PTY 40****************

167      *SECTION-40
                    000F9A                          PN=024    EQU   *
168      *W-40-0
                    000F9A                          PN=025    EQU   *
169       DISPLAY   000F9A  58 F0 C 00C                       L     15,00C(0,12)          V(ILBODSS0)
                    000F9E  05 1F                             BALR  1,15
                    000FA0  0001                              DC    X'0001'
                    000FA2  10                                DC    X'10'
                    000FA3  000010                            DC    X'000010'
                    000FA6  0C000185                          DC    X'0C000185'           LIT+205
                    000FAA  0000                              DC    X'0000'
                    000FAC  FFFF                              DC    X'FFFF'
171       PERFORM   000FAE  D2 03 D 270 D 2D0                 MVC   270(4,13),2D0(13)     PSV=11          VN=09
                      .
                      .
                      .
                    0011F6  58 F0 C 01C                       L     15,01C(0,12)          V(ILBOGD01)
                    0011FA  05 EF                             BALR  14,15
                    0011FC  47 F0 B A28                       BC    15,A28(0,11)          PN=033
```

Figure 154. Sample Segmentation Program (Part 9 of 14)

```
          **************SEGMENT OF PTY 60***************

203     *SECTION-60
                  001200                          PN=033    EQU   *
204     *W-60-0
                  001200                          PN=034    EQU   *
205       DISPLAY 001200  58 F0 C 00C                       L     15,00C(0,12)          V(ILBODSS0)
                  001204  05 1F                             BALR  1,15
                  001206  0001                              DC    X'0001'
                  001208  10                                DC    X'10'
                  001209  000010                            DC    X'000010'
                  00120C  0C0001D4                          DC    X'0C0001D4'           LIT+236
                  001210  0000                              DC    X'0000'
                  001212  FFFF                              DC    X'FFFF'
207       PERFORM 001214  D2 03 D 27C D 2F0                 MVC   27C(4,13),2F0(13)     PSV=14          VN=013
                          :
                          :
                  00144A  58 F0 C 01C                       L     15,01C(0,12)          V(ILBOGD01)
                  00144E  05 EF                             BALR  14,15
                  001450  47 F0 B C7C                       BC    15,C7C(0,11)          PN=041


          **************SEGMENT OF PTY 80***************

237     *SECTION-80
                  001454                          PN=041    EQU   *
238     *W-80-0
                  001454                          PN=042    EQU   *
239       DISPLAY 001454  58 F0 C 00C                       L     15,00C(0,12)          V(ILBODSS0)
                  001458  05 1F                             BALR  1,15
                  00145A  0001                              DC    X'0001'
                  00145C  10                                DC    X'10'
                  00145D  000010                            DC    X'000010'
                  001460  0C0001F3                          DC    X'0C0001F3'           LIT+267
                  001464  0000                              DC    X'0000'
                  001466  FFFF                              DC    X'FFFF'
241       PERFORM 001468  D2 03 D 284 D 308                 MVC   284(4,13),308(13)     PSV=16          VN=016
                          :
                          :
                  001766  05 10                             BALR  1,0
                  001768  58 00 8 000                       L     0,000(0,8)
                  00176C  12 00                             LTR   0,0
                  00176E  47 80 1 010                       BC    8,010(0,1)
                  001772  1E 0B                             ALR   0,11
                  001774  50 00 8 000                       ST    0,000(0,8)
                  001778  87 86 1 000                       BXLE  8,6,000(1)
                  00177C  58 60 D 204                       L     6,204(0,13)           BL =2
                  001780  58 70 D 200                       L     7,200(0,13)           BL =1
                  001784  D2 8F D 290 C 054                 MVC   290(144,13),054(12)   VN=01           VNI=1
                  00178A  58 E0 D 054                       L     14,054(0,13)
                  00178E  07 FE                             BCR   15,14
```

Figure 154. Sample Segmentation Program (Part 10 of 14)

```
*STATISTICS*      SOURCE RECORDS =   274     DATA DIVISION STATEMENTS =    18     PROCEDURE DIVISION STATEMENTS =   165
*OPTIONS IN EFFECT*     SIZE =  131072  BUF =  12288  LINECNT = 57  SPACE1, FLAGW,   SEQ,   SOURCE
*OPTIONS IN EFFECT*       DMAP,   PMAP, NOCLIST, NOSUPMAP, NOXREF,   SXREF,   LOAD, NODECK, APOST, NOTRUNC, NOFLOW
*OPTIONS IN EFFECT*     NOTERM, NONUM, NOBATCH, NONAME, COMPILE=01, NOSTATE, NORESIDENT, NODYNAM, NOLIB, NOSYNTAX
*OPTIONS IN EFFECT*       OPTIMIZE, NOSYMDMP, NOTEST,   VERB,   ZWB, SYST, NOENDJOB, NOLVL
*OPTIONS IN EFFECT*     NOLST , NOFDECK,NOCDECK, LCOL2,  L120,   DUMP , NOADV , NOPRINT,
*OPTIONS IN EFFECT*     NOCOUNT, NOVBSUM, NOVBREF, LANGLVL(2)
```

CROSS-REFERENCE DICTIONARY

DATA NAMES	DEFN	REFERENCE									
ERCTFL	000056	000123	000127	000131	000157	000161	000165	000193	000197	000201	000227
		000231	000235	000263	000267	000271					
ERROKSW	000055	000090	000094	000109	000123	000127	000144	000157	000161	000178	000193
		000197	000214	000227	000231	000249	000263	000267			
FILE-1	000046	000076	000082	000083	000088	000092	000106	000121	000141	000155	000175
		000191	000211	000225	000246	000261					
MSGEOF	000060	000122	000156	000192	000226	000262					
MSGEOX	000058	000108	000143	000177	000213	000248					
MSGHDR	000057	000089	000107	000122	000126	000142	000156	000160	000176	000192	000196
		000212	000226	000230	000247	000262	000266				
MSGNEF	000062	000089									
REC-HD	000065										
REC-ID	000064	000125	000126	000128	000159	000160	000162	000195	000196	000198	000229
		000230	000232	000265	000266	000268					
REC-NO	000066	000111	000146	000180	000216	000251					
RECFD1	000053	000088	000106	000121	000141	000155	000175	000191	000211	000225	000245
		000261									
RECID	000068	000108	000143	000177	000213	000248					
REC1	000063	000088	000106	000121	000141	000155	000175	000191	000211	000225	000246
		000261									
VER-HD	000071										
VER-ID	000070	000125	000126	000128	000159	000160	000162	000195	000196	000198	000229
		000230	000232	000265	000266	000268					
VER-NO	000072	000129	000163	000199	000233	000269					
VER-REC	000069										

Figure 154. Sample Segmentation Program (Part 11 of 14)

```
PROCEDURE NAMES                 DEFN    REFERENCE

BASE-SECTION                    000074
BASE-50                         000081  000110  000145  000179  000215  000250
BASE-60                         000086  000153
BASE-70                         000091  000088  000132  000166  000202  000236  000272
BASE-90                         000093
R-20-0                          000114  000187
R-20-9                          000118  000187
R-21-0                          000120  000117
R-21-9                          000130  000117  000124
R-30-0                          000149  000223
R-31-0                          000154  000152
R-31-9                          000164  000152  000158
R-40-0                          000183  000087
R-40-9                          000188  000087
R-41-0                          000190  000186
R-41-9                          000200  000194
R-60-0                          000219  000085
R-61-0                          000224  000222
R-61-9                          000234  000222  000228
R-80-0                          000254  000084
R-80-9                          000258  000084
R-81-0                          000260  000257
R-81-8                          000270
R-81-9                          000273  000257  000264
SECTION-20                      000098
SECTION-30                      000133  000132
SECTION-40                      000167  000166
SECTION-60                      000203  000202
SECTION-80                      000237  000236
W-20-0                          000099  000242
W-20-9                          000103  000242
W-21-0                          000105  000102
W-21-9                          000112  000102
W-30-0                          000134  000078
W-30-9                          000138  000078
W-31-0                          000140  000137
W-31-9                          000147  000137
W-40-0                          000168  000080
W-40-9                          000172  000080
W-41-0                          000174  000171
W-41-9                          000181  000171
W-60-0                          000204  000079
W-60-9                          000208  000079
W-61-0                          000210  000207
W-61-9                          000217  000207
W-80-0                          000238  000077
W-80-9                          000243  000077
W-81-0                          000245  000241
W-81-9                          000252  000241
```

Figure 154. Sample Segmentation Program (Part 12 of 14)

```
F64-LEVEL LINKAGE EDITOR OPTIONS SPECIFIED LIST,XREF
        DEFAULT OPTION(S) USED -  SIZE=(196608,65536)
IEW0000     ENTRY SEGOSAMP
IEW0201
```

CROSS REFERENCE TABLE

CONTROL SECTION NAME	ORIGIN	LENGTH	SEG. NO.
SEGOSAMP	00	1790	1
ILBOCOM0*	1790	169	1
ILBODSS *	1900	40A	1
ILBOEXT *	1D10	50	1
ILBOGDO *	1D60	114	1
ILBOQIO *	1E78	56E	1
ILBOSRV *	23E8	48E	1
ILBOSYN *	2878	440	1
ILBOBEG *	2CB8	128	1
ILBOCHN *	2DE0	1B0	1
ILBOCMM *	2F90	388	1
ILBOMSG *	3320	F2	1

ENTRY

NAME	LOCATION	NAME	LOCATION	NAME	LOCATION	NAME	LOCATION
ILBOCOM	1790						
ILBODSS0	1902						
ILBOEXT0	1D12	ILBOEXT1	1D16				
ILBOGDO0	1D62	ILBOGDO1	1D66	ILBOGDO2	1D6A		
ILBOQIO0	1E7A						
ILBOSRV0	23F2	ILBOSR5	23F2	ILBOSR3	23F2	ILBOSR	23F2
ILBOSRV1	23F6	ILBOSTP1	23F6	ILBOST	23FA	ILBOSTP0	23FA
ILBOSYN0	287A	ILBOSYN1	287E	ILBOSYN2	2882	ILBOSYN3	2886
ILBOSYN4	288A	ILBOSYN5	288E				
ILBOBEG0	2CBA						
ILBOCHN0	2DE2						
ILBOCMM0	2F92	ILBOCMM1	2F96				
ILBOMSG0	3322						

LOCATION	REFERS TO SYMBOL	IN CONTROL SECTION	SEG. NO.	LOCATION	REFERS TO SYMBOL	IN CONTROL SECTION	SEG. NO.
5C4	ILBOSRV0	ILBOSRV	1	5C8	ILBOSR5	ILBOSRV	1
5CC	ILBODSS0	ILBODSS	1	5D0	ILBOSYN1	ILBOSYN	1
5D4	ILBOEXT0	ILBOEXT	1	5D8	ILBOQIO0	ILBOQIO	1
5DC	ILBOGDO1	ILBOGDO	1	5E0	ILBOSRV1	ILBOSRV	1
448	ILBOCOM0	ILBOCOM0	1	2738	ILBOCOM	ILBOCOM0	1
273C	ILBOCMM0	ILBOCMM	1	2740	ILBOBEG0	ILBOBEG	1
2744	ILBOMSG0	ILBOMSG	1	2748	ILBOSND2	$UNRESOLVED(W)	
2C40	ILBOCHN0	ILBOCHN	1				

```
ENTRY ADDRESS       00

 TOTAL LENGTH      3418
****GO        DOES NOT EXIST BUT HAS BEEN ADDED TO DATA SET
AUTHORIZATION CODE IS        0.
```

Figure 154. Sample Segmentation Program (Part 13 of 14)

```
START TEST SEG-SAMPLE
SECTION 80 WRITE
SECTION 20 WRITE
SECTION 30 WRITE
SECTION 60 WRITE
SECTION 40 WRITE
SECTION 80 READ
SECTION 60 READ
SECTION 30 READ
SECTION 40 READ
SECTION 20 READ
END TEST SEG-SAMPLE SUCCESSFUL RUN
```

Figure 154. Sample Segmentation Program (Part 14 of 14)

USING THE CHECKPOINT/RESTART FEATURE

The IBM Operating System Checkpoint/Restart feature is designed to be used with programs running for an extended period of time when interruptions may halt processing before the end of the job. The feature may be used when the programmer anticipates any type of interruption, i.e., interruptions caused by machine malfunctions, input/output errors, or intentional operator intervention, etc. It allows the interrupted program to be restarted at the job step or at a point other than at the beginning of the job step. The feature consists of two routines: Checkpoint and Restart.

The Checkpoint routine is invoked from the COBOL load module containing the user's program. It moves information stored in registers and in main storage into a checkpoint record at user-designated points during execution of the program. The programmer specifies these points using the COBOL RERUN clause in the Environment Division.

The Restart routine restarts an interrupted program. Restart can occur at the beginning of a job step, or at a checkpoint if a checkpoint record has been written. The checkpoint record will contain all information necessary to restart the program. Restart can be initiated at any time after the program was interrupted; that is, it may be run immediately after the interrupt has occurred, as an automatic restart, or at a later time convenient to the programmer, as a deferred restart.

The COBOL RERUN clause provides linkage to the system checkpoint routine. Hence, any cautions and restrictions on the use of the system Checkpoint/Restart feature also apply to the use of the RERUN clause.

The Checkpoint/Restart feature is fully described in the publication OS/VS Checkpoint/Restart.

TAKING A CHECKPOINT

In order to initiate a checkpoint, the programmer uses job control statements and the COBOL RERUN clause. The programmer associates each RERUN clause with a particular COBOL file. The RERUN clause indicates that a checkpoint record is to be written onto a checkpoint data set whenever a specified number of records on that file are processed or when end of volume is reached while processing a file. The programmer decides when he wants the checkpoints taken as he codes the RERUN clause. The checkpoint records are written on the checkpoint data set defined by the DD statement and are referenced by system-name in the RERUN clause. The DD statement describes both a checkpoint data set and a checkpoint method.

Checkpoint records on ASCII-collated sorts can be taken, but the system-name indicating the checkpoint data set must not specify an ASCII file.

Note: If checkpoints are to be taken during a sorting operation, a DD statement called SORTCKPT must be added when the program is executed.

Checkpoint Methods

The programmer may elect to store single or multiple checkpoints.

Single: Only one checkpoint record exists at any given time. After the first checkpoint record is written, any succeeding checkpoint record overlays the previous one. This method is acceptable for most programs. It offers the advantage of saving space on the checkpoint data set and allows the programmer to restart his program at the latest checkpoint.

Multiple (multiple contiguous): Checkpoints are recorded and numbered sequentially. Each checkpoint is saved. This method is used when the programmer may wish to restart a program at a checkpoint other than the latest one taken.

DD STATEMENT FORMATS

The programmer records checkpoints on tape or direct access devices. Following are the DD formats to define checkpoint data sets.

For Tape:

```
//ddname  DD DSNAME=data-set-name,       X
//           VOLUME=SER=volser,          X
//           UNIT=deviceno,              X
//           DISP=( {NEW}                X
//                  {MOD} ,PASS),        X
//           DCB=(TRTCH=C),LABEL=(,NL)
```

Note: The DCB parameter is necessary only for 7-track tape conversion; for 9-track tape it is not used.

For Mass Storage:

```
//ddname  DD DSNAME=data-set-name,       X
//           VOLUME=(PRIVATE,RETAIN,     X
//             SER=volser),              X
//           UNIT=deviceno,              X
//           SPACE=(subparms),           X
//           DISP=( {NEW}                X
//                  {MOD} ,PASS,KEEP)    X
```

where:

ddname
: is the same as the ddname portion of the system-name used in the COBOL RERUN clause to provide a link to the DD statement.

data-set-name
: is the name given to each particular data set used to write checkpoint records. This name identifies the checkpoint data set to the Restart procedure (see "Restarting a Program").

volser
: identifies the volume by serial number.

deviceno
: identifies the device. For tape it indicates the device number for 7-track or 9-track tape. For mass storage, it indicates the device number for disk or drum.

subparms
: specifies the amount of track space needed for the data set.

MOD
: is specified for the multiple contiguous checkpoint method.

NEW
: is specified for the single checkpoint method.

PASS
: is specified in order to prevent deletion of the data set at the successful completion of the job step, unless it is the last step in the job. If it is the last step, the data set will be deleted with PASS.

KEEP
: is specified in order to keep the data set if the job step abnormally terminated and may be restarted.

The following listings are examples that define checkpoint data sets.

- To write single checkpoint records using tape:

```
//CHECKPT  DD DSNAME=CHECK1,              X
//            VOLUME=SER=ND003,           X
//            UNIT=2400,DISP=(NEW,KEEP),  X
//            LABEL=(,NL)
                 .
                 .
                 .
       ENVIRONMENT DIVISION.
                 .
                 .
                 .
           RERUN ON UT-2400-S-CHECKPT EVERY
           5000 RECORDS OF ACCT-FILE.
```

- To write single checkpoint records using disk (note that more than one data set may share the same external-name):

```
//CHEK  DD    DSNAME=CHECK2,              X
//            VOLUME=(PRIVATE,RETAIN,     X
//              SER=DB030,                X
//            UNIT=2314,DISP=(NEW,KEEP),  X
//            SPACE=(TRK,300)
                 .
                 .
                 .
       ENVIRONMENT DIVISION.
                 .
                 .
                 .
           RERUN ON UT-2314-S-CHEK EVERY
           20000 RECORDS OF PAYCODE.
           RERUN ON UT-2314-S-CHEK EVERY
           30000 RECORD OF IN-FILE.
```

- To write multiple contiguous checkpoint records (on tape):

```
//CHEKPT   DD  DSNAME=CHECK3,                X
//             VOLUME=SER=111111,            X
//             UNIT=2400,DISP=(MOD,PASS),    X
//             LABEL=(,NL)
                .
                .
                .
       ENVIRONMENT DIVISION.
                .
                .
                .
           RERUN ON UT-2400-S-CHEKPT EVERY
           10000 RECORDS OF PAY-FILE.
```

Note: A checkpoint data set must have sequential or partitioned organization.

DESIGNING A CHECKPOINT

The programmer should design his checkpoints at critical points in his program so that data may be easily reconstructed. For example, in a program using mass storage files, changes to records in these files will replace previous information; thus the programmer should be sure he can identify previously processed records. Assume that a mass storage file contains loan records that periodically are updated for interest due. If a checkpoint is taken, records are updated, and then the program is interrupted, the records updated after the last checkpoint will be updated a second time in error unless the programmer controls this condition. (He may set up a date field for each record and update the date each time the record is processed. Then, after the restart, by investigating the date field he can determine whether or not the record was previously processed.) For efficient repositioning of a print file, the programmer should take checkpoints on that file only after printing the last line of a page. At system generation time, those ABEND codes for which the checkpoints are desired (DEFAULT) must be specified.

MESSAGES GENERATED DURING CHECKPOINT

The system checkpoint routine advises the operator of the status of the checkpoints taken by displaying informative messages on the console.

When a checkpoint has been successfully completed, the following message will be displayed:

```
[IHJ004I jobname (ddname,unit,volser)
      CHKPT checkid]
```

where checkid is the identification name of the checkpoint taken. Checkid is assigned by the control program as an 8-digit number. The first digit is the letter C, followed by a decimal number indicating the checkpoint. For example, checkid C0000004 indicates the fourth checkpoint taken in the job step.

RESTARTING A PROGRAM

The system Restart routine retrieves the information recorded in a checkpoint record, restores the contents of main storage and all registers.

The Restart routine can be initiated in one of two ways:

- Automatically at the time an interruption stopped the program
- At a later time as a deferred restart

The type of restart is determined by the RD parameter of the job control language.

RD Parameter

The RD parameter may appear on either the JOB or the EXEC statement. If coded on the JOB statement, the parameter overrides any RD parameters on the EXEC statement. If the programmer wishes to have his program restart automatically, he codes RD=R or RD=RNC. RD=R indicates that restart is to occur at the latest checkpoint. The programmer should specify the RERUN clause for at least one data set in his program in order to record checkpoints. If no checkpoint is taken prior to interruption, restart occurs at the beginning of the job step. RD=RNC indicates that no checkpoint is to be written and any restart will occur at the beginning of the job step. In this case, RERUN clauses are unnecessary; if any are present, they are ignored. If the RD parameter is omitted, the CHKPT macro instruction remains activated, and checkpoints may be taken during processing. If an interrupt occurs after the first checkpoint, automatic restart will occur. Thus, if the user does not want automatic restart, he should always include the RD parameter with a code of either RD=NR or RD=NC, both of which suppress the automatic restart procedure.

If the programmer wishes his program to be restarted on a deferred basis, he should code the RD parameter as RD=NR. This form of the parameter suppresses automatic restart but allows a checkpoint record to be written provided a RERUN clause has been specified. At restart time, the programmer may choose to restart his program at a checkpoint other than at the beginning of the job step.

The programmer may also elect to suppress both restart and writing checkpoints. By coding RD=NC, the programmer, in effect, is ignoring the features of the Checkpoint/Restart facility.

Automatic Restart

Automatic Restart occurs only at the latest checkpoint taken. (If no checkpoint was taken before interruption, Automatic Restart occurs at the beginning of the job step).

In order to restart automatically, a program must satisfy the following conditions.

- A program must request restart by using the RD parameter or by taking a checkpoint.
- An ABEND that terminated the job must return a code eligible to cause restart. (For further discussion on this requirement, see the publication OS/VS Checkpoint/Restart.)
- The operator authorizes the restart, with the following procedure:

  The system displays the following message to request authorization of the restart:

  ```
  xxIEF225D SHOULD
       jobname.stepname.procstep
       RESTART [checkid]
  ```

  The operator must reply in the following form:

  ```
       REPLY xx, '{YES|NO|HOLD}'
  ```

  where YES authorizes restart, NO prevents restart, and HOLD defers restart until the operator issues a RELEASE command, at which time restart will occur.

Whenever automatic restart is to occur, the system will reposition all devices except unit-record machines.

Deferred Restart

Deferred restart may occur at any checkpoint, not necessarily the latest one taken.

The programmer requests a deferred restart by means of the RESTART parameter on the JOB card and a SYSCHK DD statement to identify the checkpoint data set. The formats for these statements are as follows:

```
//jobname JOB ,MSGLEVEL=1,                    X
//            RESTART=(request,[checkid])
//SYSCHK  DD  DSNAME=data-set-name,           X
//            DISP=OLD,UNIT-deviceno,         X
//            VOLUME=SER=volser
```

where:

MSGLEVEL=1 (or MSGLEVEL=(1,y) where y is either 0 or 1)
is required.

RESTART=(request,[checkid])
identifies the particular checkpoint at which restart is to occur. Request may take one of the following forms:

* to indicate restart at the beginning of the job

stepname to indicate restart at the beginning of a job step

stepname.procstep to indicate restart at a procedure step within the jobstep

checkid
identifies the checkpoint where restart is to occur.

SYSCHK
is the DDNAME used to identify a checkpoint data set to the control program. The SYSCHK DD statement must immediately precede the first EXEC statement of the resubmitted job, and must follow any JOBLIB statement.

data-set-name
must be the same name that was used when the checkpoint was taken. It identifies the checkpoint data set

deviceno and volser
identify the device number and the volume serial number containing the checkpoint data set.

As an example illustrating the use of these job control statements, a restart of the GO step of a COBUCLG procedure, at checkpoint identifier (CHECKID) C0000003, might appear as follows:

```
//jobname JOB ,MSGLEVEL=1,                    X
//            RESTART=                        X
//              (stepname.GO,C0000003)
//SYSCHK   DD  DSNAME=CHEKPT,                  X
//            DISP=OLD,UNIT=2400,              X
//            VOLUME=SER=111111
         .
         .
         .
```

{DD statements similar to original deck}

The Restart routine uses information from DD statements in the resubmitted job to reset files for use after restart; therefore, care should be taken with any DD statements that may affect the execution of the restarted job step. Attention should be paid to the following:

- During the original execution, a data set meant to be deleted at the end of a job step should conditionally be defined as PASS rather than DELETE in order to be available if an interruption forces a restart. If the restart is at the beginning of a step, a data set created in the original execution (defined as NEW on a DD statement) must be scratched prior to the restart. If the data set is not deleted, the DD statement must be changed to define it as OLD.

- At restart time, input data sets on cards should be positioned as they were at the time of the checkpoint. Input data sets on tape or direct access devices will be automatically repositioned by the system.

- At restart time, the EXEC statement parameters PGM and COND, and the DD statement parameters SUBALLOC and VOLUME=REF must not be used in steps following the restart step if they contain the form stepname or stepname.procstep referring to a step preceding the restart step. However, if these parameters are used, the preceding step referred to must be specified in the resubmitted deck.

When a deferred restart has been successfully completed, the system will display the following message on the console:

```
IHJ008I   jobname   RESTARTED
```

Control is then given to the user's program that executes in a normal manner.

CHECKPOINT/RESTART DATA SETS

If the RERUN clause was executed during the original execution of the processing program, checkpoint entries were written on a checkpoint data set. To resubmit a job for restart when execution is to be resumed at a particular checkpoint, an additional DD statement must be included. This DD statement describes the data set on which the checkpoint entry was written and it must have the ddname SYSCHK. The SYSCHK DD statement must immediately precede the first EXEC statement of the resubmitted job and must follow the DD statement named JOBLIB, if one is present.

For both deferred and automatic checkpoint/restart, if Direct SYSOUT Writer for the restarted job was active at the time the checkpoint was was taken, it must be available for the job to restart. For further information, see the publication OS/VS Checkpoint/Restart.

If the checkpoint data set is multivolume, the sequence number of the volume on which the checkpoint entry was written must be included in the VOLUME parameter. If the checkpoint data set is on a 7-track magnetic tape with nonstandard labels or no labels, the SYSCHK DD statement must contain DCB=(TRTCH=C,...).

Figure 155 illustrates a sequence of control statements for restarting a job.

```
//PAYROLL     JOB    MSGLEVEL=1,REGION=80K,RESTART=(STEP1,CHECKPT4)
//JOBLIB      DD     DSNAME=PRIV.LIB3,DISP=OLD
//SYSCHK      DD     DSNAME=CHKPTLIB,UNIT=2314,VOL=SER=456789,        X
//                   DISP=(OLD,KEEP)
//STEP1       EXEC   PGM=PROG4,TIME=5
```

Figure 155. Restarting a Job at a Specific Checkpoint Step

If a SYSCHK DD statement is present in a job and the JOB statement does not contain the RESTART parameter, the SYSCHK DD statement is ignored. If a RESTART parameter without the CHECKID subparameter (as in Figure 157) is included in a job, a SYSCHK DD statement must not appear before the first EXEC statement for a job.

Figure 156 illustrates the use of the RD parameter. Here, the RD parameter requests step restart for any abnormally terminated job step. The DD statement DDCKPNT defines a checkpoint data set. For this step, once a RERUN clause is executed, only automatic checkpoint restart can occur, unless a CHKPT cancel is issued.

Figure 157 illustrates those modifications that might be made to control statements before resubmitting the job for step restart. The job name has been changed to distinguish the original job from the restarted job. The RESTART parameter has been added to the JOB statement and indicates that restart is to begin with the first job step. The DD statement WORK originally assigned a conditional disposition of KEEP for this data set. If this step did not abnormally terminate during the original execution, the data set was deleted and no modifications need be made to this statement. If the step did abnormally terminate, the data set was kept. In this case, define a new data set as shown in Figure 157, or change the data set's status to OLD before resubmitting the job. A new data set has also been defined as the checkpoint data set.

Figure 158 illustrates those modifications that might be made to control statements before resubmitting the job for checkpoint restart.

```
//J1234      JOB   386,SMITH,MSGLEVEL=1,RD=R
//S1         EXEC  MYPROG
//INDATA     DD    DSNAME=INVENT,UNIT=2400,DISP=OLD,VOLUME=SER=91468,  X
//                 LABEL=RETPD=14
//REPORT     DD    SYSOUT=A
//WORK       DD    DSNAME=T91468,DISP=(,,KEEP),UNIT=SYSDA,             X
//                 SPACE=(3000,(5000,500)),VOLUME=(PRIVATE,RETAIN,,6)
//DDCKPNT    DD    UNIT=2400,DISP=(MOD,PASS,CATLG),DSNAME=C91468,LABEL=(,NL)
```

Figure 156. Using the RD Parameter

```
//J3412      JOB   386,SMITH,MSGLEVEL=1,RD=R,RESTART=*
//S1         EXEC  MYPROG
//INDATA     DD    DSNAME=INVENT,UNIT=2400,DISP=OLD,VOLUME=SER=91468,  X
//                 LABEL=RETPD=14
//REPORT     DD    SYSOUT=A
//WORK       DD    DSNAME=S91468,DISP=(,,KEEP),UNIT=SYSDA,             X
//                 SPACE=(3000,(5000,500)),VOLUME=(PRIVATE,RETAIN,,6)
//DDCHKPNT   DD    UNIT=2400,DISP=(MOD,PASS,CATLG),DSNAME=R91468,LABEL=(,NL)
```

Figure 157. Modifying Control Statements Before Resubmitting for Step Restart

```
//J3412     JOB   386,SMITH,MSGLEVEL=1,RD=R,RESTART=(*,C0000002)
//SYSCHK    DD    DSNAME=C91468,DISP=OLD
//S1        EXEC  MYPROG
//INDATA    DD    DSNAME=INVENT,UNIT=2400,DISP=OLD,                    X
//                VOLUME=SER=91468,LABEL=RETPD=14
//REPORT    DD    SYSOUT=A
//WORK      DD    DSNAME=T91468,DISP=(,,KEEP),UNIT=SYSDA,              X
//                SPACE=(3000,(5000,500)),VOLUME=(PRIVATE,RETAIN,,6)
//DDCKPNT   DD    UNIT=2400,DISP=(MOD,KEEP,CATLG),DSNAME=C91468,       X
//                LABEL=(,NL)
```

Figure 158. Modifying Control Statements Before Resubmitting for Checkpoint Restart

The job name has been changed to distinguish the original job from the restarted job. The RESTART parameter has been added to the JOB statement and indicates that restart is to begin with the first step at the checkpoint entry named C0000002. The DD statement DDCKPNT originally assigned a conditional disposition of CATLG for the checkpoint data set. If this step did not abnormally terminate during the original execution, the data set was kept. In this case, the SYSCHK DD statement must contain all of the information necessary to retrieve the checkpoint data set. If the job did abnormally terminate, the data set was cataloged. In this case, the only parameters required on the SYSCHK DD statement, as shown in Figure 158, are the DSNAME and DISP parameters.

Note: If a checkpoint is taken in a job that is running when V=R is specified, the job cannot be restarted until adequate nonpageable dynamic storage becomes available.

A communication environment consists of a central computer, remote or local[1] stations, and communication lines between such stations and the central computer. Use of the Communication Feature enables the COBOL programmer to create device-independent programs for communication applications.

Communication applications require a special, user-written assembler-language program that controls the flow of data between the central computer and the remote stations. This message control program (MCP) also performs such additional tasks required only in a communication environment as dial-up, polling, (or contacting each remote station), and synchronization, as well as such device-dependent tasks as character translation and insertion of control characters.

The MCP consists of routines that identify the communication network to the operating system, establish line control between the computer and the various kinds of stations, and process messages in a way tailored to meet the needs of the user. A "message" is the data flowing either from a remote station to the central computer or from the central computer to a remote station. An MCP is required in a communication system operating under TCAM.

Depending on the needs of the installation, one or more COBOL programs may be required to process the contents of the messages. An example of a job needing no application program is message switching, an operation consisting only of forwarding messages unaltered (except for such processing as the MCP may perform) to one or more other stations.

The MCP itself can perform limited processing (for example, examination of the first portion of a message to determine certain routine information and message code translation). Further, the MCP can obtain the time of day a message is received from a station and transmit this information to a COBOL program. It can also check the input messages to determine whether an error message should be sent to the designated station.

This section describes the flow of a single-segment message through a system operating under TCAM, from the time it is entered at the remote station to its transmission to a destination station. Figure 159 outlines the flow of a message segment through a TCAM system. The encircled numerals in the flow diagram correspond to the steps listed in the description that follows.

Because of the possible variety of both message types and destinations, it is often helpful for the user to precede the message "text" with a message "header" so that the user can transmit to the MCP information essential to handling the text. It is the user who determines which part of the message is the header and which part is the text.

Steps 1 and 2: The input message is prepared at the remote station and entered on the line. The message may be keyed in, or it may be entered from a card or tape reader. The originating station enters the message via a communication line, the transmission control unit, and the multiplexor channel.

Step 3: The message enters the central computer and is stored, together with the internally generated buffer prefix, in a main storage buffer. As message data fills the buffer, TCAM inserts the necessary control information in the prefix. Before the message characters are placed in the first buffer, TCAM may reserve space in the buffer for later insertion of the time, date, and sequence number for the message, and for control characters, if appropriate. Once a buffer is filled with the first segment of the message, the MCP controls the flow of the buffer through the communication network. The heart of the MCP consists of the message handlers (MH) constructed by the user to process messages from the various lines or line groups.

---

[1] A station whose control unit is connected directly to a computer data channel by a local cable.

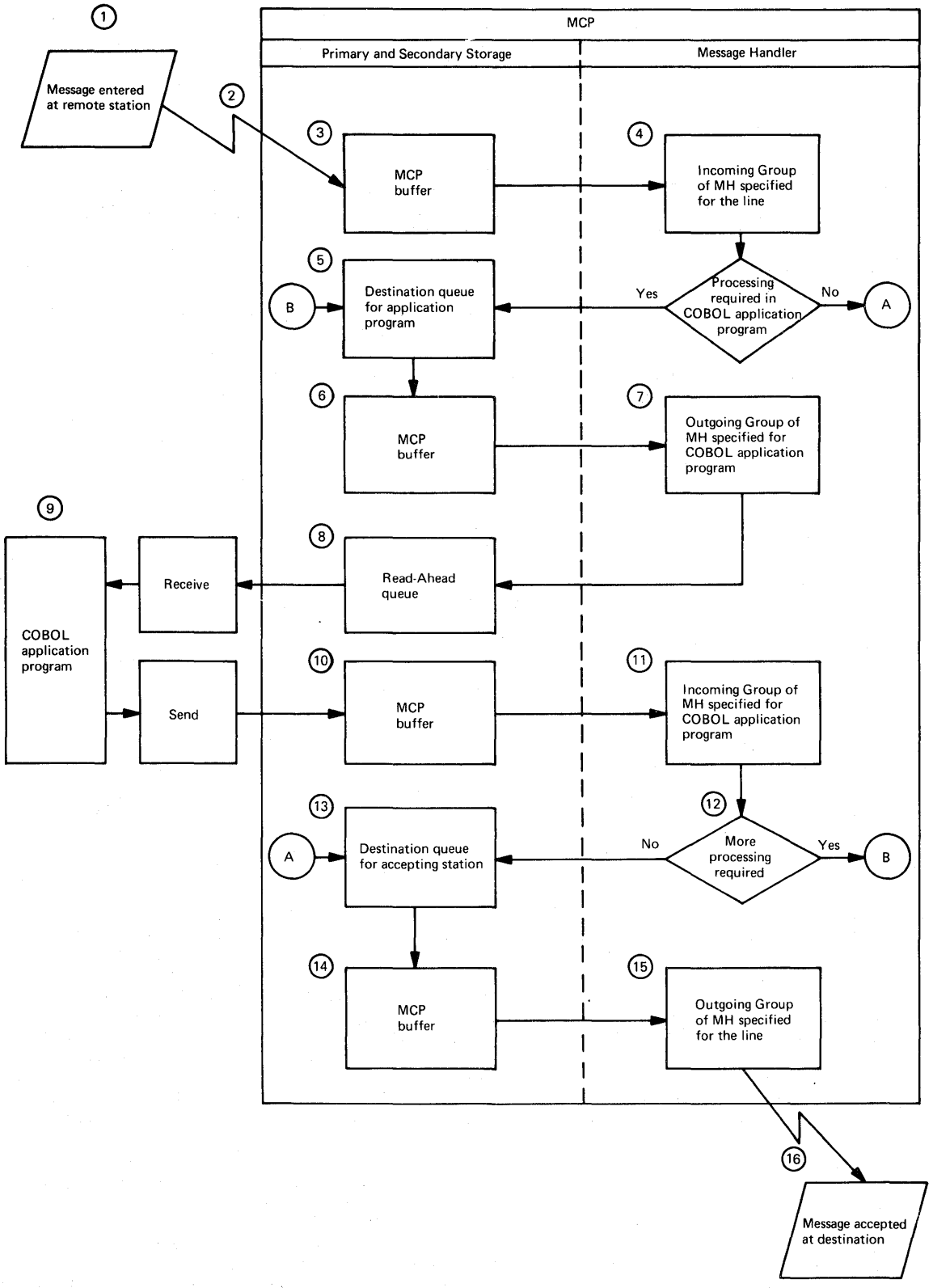


Figure 159. Message Flow between Remote Stations and a COBOL Program

Step 4: The incoming message is routed to the incoming group of the MH specified for the line (by the MH= operand of the DCB macro for the line group in which the line is included). The message is passed, a buffer at a time, through the incoming group, which performs such user-selected functions on the message header as origin checking, and input sequence-number checking. Similarly, such functions may be performed for the message segment as translating the segment from line code to EBCDIC and causing an error message to be sent to the originating station when the incoming group detects any user-specified error in the segment. In performing its functions, the incoming group of the MH scans and processes header fields based on the relative order of the individual MH macro instructions. The incoming group then routes the message to the destination queue.

Step 5: After processing by the incoming group, the message is placed on a destination queue for either the COBOL program, for processing, or an accepting station. (If no message processing is necessary, the next action performed is that described in Step 13.) All messages requiring text processing are routed to the destination queue for the COBOL program that processes that type of message. The user controls this routing via the message header by placing the name of the destination queue for the COBOL program in a destination field of the message header or by MH macro instructions such as MSGTYPE that may be used to direct messages of a particular type to a particular queue.

Steps 6, 7, and 8: The message from a destination queue for a COBOL program is placed in a main-storage buffer; the outgoing group of an MH (the MH is created especially for the application program and is assigned to it by the MH= operand of the PCB macro in the MCP) places it on the read-ahead queue, a special queue that allows overlap of MCP and application program processing of messages queued for a particular destination.

Step 9: Each time the COBOL program issues a RECEIVE statement, TCAM passes message data from the read-ahead queue to a user-specified work area in the COBOL program. As the message data is moved to the work area, TCAM removes the header or text prefix from the buffer. After receiving the message data, the COBOL program processes it as required and then generates a response message, if any is to be returned to a station. The destination queues act as buffers between the COBOL TP program and the remote stations. Thus, the COBOL communication program can accept messages from MCP destination queues and place these messages in MCP destination queues as if the queues were sequential files within a conventional COBOL program. (The sample COBOL program TESTTP1, shown in Figure 165, reads a sequential file and then sends each record to a destination queue, creating a TCAM data set for the COBOL communication program TESTTP2, shown in Figure 169, making it possible to test a COBOL TP program without terminals.)

Steps 10 and 11: When the COBOL program issues a SEND statement, TCAM moves the data from the work area into an MCP buffer before it is handled by the incoming group of the MH designed for the COBOL program. A header or text buffer prefix is created when data is moved to the buffer, as for other incoming messages. As the message data fills the buffer, TCAM inserts control information in the prefix field. The response message generated by an application program can be any user-selected length. After the buffer is filled, the message is handled by the incoming group of the MH assigned to the application program by the MH= operand of the PCB macro instruction that provides an interface between the MCP and the COBOL program.

Step 12: If further processing of the message is required in another application program, the message is queued for that destination (and Steps 5 through 11 are repeated). If however, no other application program processing is needed, the processed message is placed on the destination queue for an accepting station. The destination is that specified by the COBOL programmer in the file referenced by the SYMBOLIC DESTINATION clause of the output CD. It may be for an application program or a station.

Step 13: The destination queue for an accepting station, like the destination queue for an application program, is a part of the message queues data set. TCAM obtains message segments from the destination queue on a first-ended first-out (FEFO) basis within priority groups.

Steps 14 and 15: The message segment is placed in a buffer, and the outgoing group of the MH specified for the line processes the message. The MH performs such user-selected functions as converting the code of the message to the transmission code for the station (if necessary), inserting the time and data in the header, logging messages, and updating message counts. These operations are performed in the buffers that receive the message segments from the destination queue.

Step 16: TCAM transmits the message, minus the header and text prefixes, to the appropriate station.

WRITING A MESSAGE CONTROL PROGRAM

The COBOL programmer can write a message control program (MCP) designed specifically for his communication needs using telecommunications access methods (TCAM) macro instructions. Using a group of TCAM macro instructions, the user follows in general the coding requirements and restrictions of any other assembler-language macro instruction. Guidelines for writing an MCP are contained in the publication OS/VS TCAM Programmer's Guide. The user must tailor these general statements to meet the needs of the installation.

The sample message control program that appears in Figure 160 in this chapter is a hypothetical program designed for specific COBOL applications. The needs of the user will undoubtedly vary from installation to installation. Nevertheless, the sample MCP together with the sample COBOL programs TESTTP1 and TESTTP2 (shown in Figures 168 and 169) can serve as an excellent example of COBOL programs and an MCP written for teleprocessing applications. Note that references to LOG are shown as comments, which would have to be changed to valid statements if that feature was required.

If the MCP to be written must conform to the 1974 ANS standard, then Figure 160 must be modified (as explained later in this section).

FUNCTIONS OF THE MESSAGE CONTROL PROGRAM

Depending on the requirements of the installation, the user can create an MCP to perform any of the following functions:

- Enable and disable communication lines
- Invite terminals to transmit messages
- Receive messages from terminals
- Dynamically assign buffers to incoming messages
- Handle messages on the basis of user-specified priorities
- Perform message-editing functions for incoming messages
- Determine the appropriate destination queue for a message and route the message to that queue
- Queue the message in the appropriate destination queue
- Place response messages generated by application programs on queues for subsequent transmission
- Retrieve messages from destination queues and prepare them for transmission to remote stations
- Perform message-editing functions for outgoing messages
- Take periodic checkpoints of the system
- Provide operator-to-system communications through system control terminals
- Initiate corrective action when an error or unusual condition is detected
- Cancel incoming messages containing errors
- Reroute messages with erroneous control information to a special queue
- Transmit error messages

However, not all of these functions are required of an MCP. Many of the optional TCAM macros allow the user to write an MCP that includes functions that would otherwise have to be executed by the COBOL program. There are, nevertheless, some functions the MCP must always provide and in so doing follow certain conventions. These requirements are discussed under "User Tasks."

USER TASKS

Guidelines for writing an MCP are contained in the publication OS/VS TCAM Programmer's Guide: The user must tailor these general statements to meet the specific needs of his installation. For example, a message can be transmitted from one terminal to another, from a terminal to an application program, or from one application program to another. Moreover, the message may contain any one of several types of data.

Regardless of the specific requirements of the user, the MCP writer must always be concerned with four major tasks, as follows:

- Defining the main storage buffers used by the MCP for handling, queueing, and transferring message data between communication lines and queueing devices.
- Defining the data sets referred to by the MCP, and providing for their activation and deactivation.
- Defining the various terminal and line control areas used by the MCP (that is, the operating procedures and signals by which a teleprocessing system is controlled).
- Defining the message handlers (the sets of routines that examine and process control information in message headers, prepare message segments for forwarding to their destination, and route messages to their proper destination).

In carrying out each of these tasks, the user codes a variety of assembler-language macros in a specified order. Some of these macros must be included in every MCP; others the user specifies according to the needs of his installation. Required as well as optional macros are illustrated in the sample MCP given in Figure 160. The encircled numerals in the discussion that follows refer to sections of code that are similarly labeled in the figure.

```
 LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                ASM 0102 19.28 06/07/74
                                    1 ***
                                    2 *
                                    3 * MESSAGE CONTROL PROGRAM
                                    4 *
000000                              5 MCP       CSECT
                                    6           PRINT NOGEN
                                    7 *
                                    8 * IN THE FOLLOWING MACRO--
                                    9 *         PROGID MAY BE OMITTED--IF USED, IT IS PLACED AT THE
                                   10 *           BEGINNING OF THE EXECUTABLE CODE IN THE MCP
                                   11 *         DISK=YES IS THE ASSUMED OPERAND--IF NO MESSAGE QUEUES DATA
                                   12 *           SETS ARE ON DISK,CODE DISK=NO
                                   13 *         CPB= USED IN READING FROM AND WRITING TO DISK--NEEDED IF
                                   14 *           DISK=YES--NO. DEPENDS ON NO. OF LINES, AMOUNT OF MESSAGE
                                   15 *           TRAFFIC AND SIZE OF BUFFER UNITS
                                   16 *         CIB=NO. OF COMMAND INPUT BLOCKS--BUFFER-LIKE AREAS USED TO
                                   17 *           CONTAIN OPERATOR CONTROL MESSAGES FROM SYSTEM CONSOLE--
                                   18 *           FREED ONCE A MESSAGE PROCESSED--2 ASSUMED AND MAX. IS 255
                                   19 *         PRIMARY=SYSCON--THIS IS ASSUMED AND SPECIFIES THE SYSTEM
                                   20 *           CONSOLE AS THE PRIMARY OPERATOR CONTROL TERMINAL FOR
                                   21 *           ENTERING AND ACCEPTING OPERATOR CONTROL MESSAGES--IF A
                                   22 *           TERMINAL IS SPECIFIED, IT MUST BE ON A NON-SWITCHED LINE
                                   23 *           AND BE ABLE TO ACCEPT AND ENTER MESSAGES
                                   24 *         CONTROL=--USED TO IDENTIFY OPERATOR CONTROL MESSAGES TO SYSTEM
                                   25 *           WHEN RECEIVED FROM OTHER THAN SYSTEM CONSOLE--0 IS DEFAULT
                                   26 *           AND IS VALID ONLY IF ALL OPERATOR COMMANDS ARE TO BE
                                   27 *           ENTERED FROM SYSTEM CONSOLE
                                   28 *         KEYLEN=--SIZE OF BUFFER UNIT--BETWEEN 33 AND 255--
                                   29 *           CAN ALSO SPECIFY BY UNITSZ= RATHER THAN KEYLEN=
                                   30 *         LNUNITS=--NO. OF BUFFER UNITS TO BE USED IN BUILDING BUFFERS
                                   31 *           FOR INCOMING AND OUTGOING MESSAGE SEGMENTS--IF TOO FEW ARE
                                   32 *           SPECIFIED, INCOMING MESSAGE DATA MAY BE LOST--TOO MANY
                                   33 *           WASTES STORAGE SPACE
                                   34 *         MSUNITS=--NEEDED IF HAVE MAIN STORAGE MESSAGE QUEUES DATA SET
                                   35 *           --NO. OF BUFFER UNITS ASSIGNED TO THIS DATA SET--IF NO DISK
                                   36 *           BACK-UP IS SPECIFIED, MESSAGE SEGMENTS MAY BE LOST IF NOT
                                   37 *           ENOUGH UNITS
                                   38 *         MSMAX=--PERCENTAGE OF UNITS IN MAIN STORAGE MESSAGE QUEUES
                                   39 *           DATA SET WANT USED BEFORE BIT IN ERROR RECORD SET--
                                   40 *           70 ASSUMED
                                   41 *         MSMIN=--PERCENTAGE OF UNITS IN MAIN STORAGE MESSAGE QUEUES
                                   42 *           DATA SET WANT UNUSED BEFORE BIT SET NOTIFYING NO LONGER
                                   43 *           CROWDED--MUST BE LESS THAN MSMAX--
                                   44 *           50 ASSUMED
                                   45 *           (NOTE--THIS BIT ALWAYS SET IF SPECIFIED PERCENTAGE OF UNITS
                                   46 *           UNUSED)
                                   47 *         DLQ=--OPTIONAL--USED TO SPECIFY A TERMINAL TO RECEIVE MESSAGES
                                   48 *           HAVING INVALID DESTINATIONS AS DETERMINED BY FORWARD MACRO
                                   49 *         INTVAL=--AN OPERATOR CONTROL MESSAGE TELLS TCAM TO ENTER THIS
                                   50 *           DELAY TO MINIMIZE UNPRODUCTIVE POLLING--WHEN ALL MULTIPOINT
                                   51 *           LINES ARE INACTIVE, THE INTERVAL COMMENCES--LINES TO
                                   52 *           SWITCHED STATIONS AND NONSWITCHED CONTENTION LINES LEFT
                                   53 *           ACTIVE--THE OPERATOR COMMAND IS A MODIFY COMMAND REFERRED
                                   54 *           TO AS 'INTERVAL'--THE NO. SPECIFIES THE NO. OF SECONDS
                                   55 *         STARTUP=-IF THIS OPERAND IS OMITTED, THE USER WILL BE GIVEN
```

Figure 160. A Message Control Program for Communication Application (Part 1 of 20)

```
LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                ASM 0102 19.28 06/07/74
                                   56 *          THE OPPORTUNITY TO SPECIFY IT AT INITIALIZATION TIME AND
                                   57 *          HE MAY ALSO CHANGE OTHER INTRO OPERANDS--CY MEANS ALWAYS
                                   58 *          A COLD START--W SPECIFIES A WARM START AFTER A QUICK OR
                                   59 *          FLUSH CLOSEDOWN AND A CONTINUATION AFTER A SYSTEM FAILURE
                                   60 *          --W INDICATES THE CONTINUATION RESTART WILL INCLUDE FULL
                                   61 *          SCANNING OF THE QUEUES--WY IS THE SAME AS W EXCEPT NO
                                   62 *          SCANNING OF THE QUEUES FOR ALREADY SENT MESSAGES IS DONE--
                                   63 *          --A CHECKPOINT DATA SET IS NEEDED FOR ANYTHING BUT A COLD
                                   64 *          START--ALSO, IF DD CARD FOR CHECKPOINT DATA SET SPECIFIES
                                   65 *          DISP=NEW, WILL GET A COLD START REGARDLESS
                                   66 *        OLTEST=IF DO NOT WISH ON-LINE TEST FACILITY--CODE 0
                                   67 *        FEATURE= THE DEFAULTS ARE DIAL, 2741, AND TIMER--SINCE WE DO
                                   68 *          NOT HAVE A 2741 TERMINAL, WE ARE CODING TO INDICATE THIS
                                   69 *        LINETYP= STSP SPECIFIES START-STOP LINES ONLY, BISC SPECIFIES
                                   70 *          BSC LINES ONLY, MINI SPECIFIES ALL TERMINALS ARE IBM 1050
                                   71 *          ON LEASED LINES, BOTH IS DEFAULT AND INDICATES ALL TYPES
                                   72 *          OF LINES ARE SUPPORTED--IF THE LINES IN THE SYSTEM DO NOT
                                   73 *          FALL UNDER THE 'BOTH' CATEGORY, SPACE IS SAVED BY CODING
                                   74 *          THIS OPERAND
                                   75 *        DTRACE -- PUT IN FOR TESTING ONLY                       TEST *
                                   76 *
                              (1)  77          INTRO PROGID=MCP,DISK=YES,CPB=10,CIB=2,PRIMARY=SYSCON,        X
                                                     CONTROL=TCAM,KEYLEN=100,LNUNITS=20,MSUNITS=50,MSMAX=75, X
                                                     MSMIN=50,DLQ=T1,INTVAL=1200,STARTUP=W,OLTEST=0,          X
                                                     FEATURE=(DIAL,NO2741,TIMER),LINETYP=BOTH,                X
                                                     DTRACE=700
                                  313 *
                                  314 * TEST IF INTRO MACRO WORKED SUCCESSFULLY
000512 12FF                       315          LTR   15,15
000514 4780 D520    00528         316          BZ    OPENFILE          YES
                                  317 ABEND  ABEND 123,DUMP            INTRO OR AN OPEN FAILED
                                  325 *
                                  326 * THE MESSAGE QUEUES DATA SET MUST BE OPENED FIRST IF IT RESIDES ON
                                  327 * DISK--A MAIN STORAGE MESSAGE QUEUES DATA SET IS NOT OPENED
                              (2) 328 OPENFILE OPEN  (MSGQ,(INOUT)) (a)
000532 9110 D718    00720         334          TM    MSGQ+48,X'10'     CHECK IF OPEN SUCCESSFUL
000536 47E0 D510    00518         335          BNO   ABEND             BRANCH IF NOT
                                  336 *
                                  337 * IF THE CHECKPOINT DATA SET IS USED, IT MUST BE OPENED NEXT
                                  338          OPEN  (CHKPT,(INOUT)) (b)
000546 9110 D744    0074C         344          TM    CHKPT+48,X'10'    CHECK IF OPEN SUCCESSFUL
00054A 47E0 D510    00518         345          BNO   ABEND             BRANCH IF NOT
                                  346 *
                                  347 * OPEN LINE GROUP DATA SETS--LINES WILL BE ACTIVATED SINCE IDLE NOT
                                  348 * SPECIFIED
                                  349 * NOTE--WE ARE NOT CHECKING FOR OPEN ERRORS FOR THE LINES--SINCE THERE
                                  350 * IS PROBABLY NO NEED TO STOP THE SYSTEM IF SOME OF THE LINES ARE NOT
                                  351 * WORKING--MESSAGES WILL BE PRINTED ON THE SYSTEM CONSOLE FOR LINES
                                  352 * THAT ARE NOT WORKING--
                                  353 * IF A LINE BECOMES OPERATIONAL DURING A RUN, IT CAN THEN BE STARTED
                                  354 * BY THE VARY COMMAND USED TO START A LINE WHICH IS OPENED AS IDLE
                                  355          OPEN  (LN1050,(INOUT),LNTWX,(INOUT)) (c)
                                  363 *
                                  364 * OPEN LOG DATA SET (d)
                                  365 *                                                NEXT IWD DISABLED--*
```

Figure 160. A Message Control Program for Communication Applications (Part 2 of 20)

```
 LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                    ASM 0102 19.28 06/07/74
                                   366 *        OPEN  (MSGLOG,(OUTPUT))
                                   367 *        TM    MSGLOG+48,X'10'      CHECK IF OPEN SUCCESSFUL
                                   368 *        BNO   ABEND                BRANCH IF NOT
                                   369 *                                                                  TEST***
                                   370          OPEN  (DUMP,(OUTPUT)) (e)  FOR SNAPS
00056A 9110 D834       0083C       376          TM    DUMP+48,X'10'        CHECK IF OPEN SUCCESSFUL
00056E 47E0 D510       00518       377          BNO   ABEND                BRANCH IF NOT
                                   378 *
                                   379 * ISSUE THE FOLLOWING BETWEEN THE OPENING AND CLOSING OF THE DATA SETS
                          (3)      380          READY
                                   397 *                                                                  TEST***
                                   398          SNAP  DCB=DUMP,PDATA=ALL                                    *
                                   410 *
                          (4)      411 * CLOSE DATA SETS
                                   412          CLOSE (LN1050,,LNTWX) (a)  LINE GROUP DATA SETS
                                   420 *                                                                  TEST***
                                   421          CLOSE (DUMP,DISP) (b)      SNAP DATA SET                    *
                                   427 *                                                NEXT IWD DISABLED--*
                                   428 *        CLOSE (MSGLOG,DISP) (c)    LOG DATA SET
                                   429 *
                                   430 * ALWAYS CLOSE CHECKPOINT DATA SET NEXT TO LAST
                                   431          CLOSE (CHKPT,DISP) (d)
                                   437 *
                                   438 * THE MESSAGE QUEUES DATA SET MUST ALWAYS BE CLOSED LAST
                                   439          CLOSE (MSGQ,DISP) (e)
                                   445 *
                                   446 * RETURN TO OS SUPERVISOR
0005EE 58DD 0004       00004       447          L     13,4(13)             PICK UP ADDRESS OF SYSTEM SAVE AREA SAVED
                                   448 *                                   IN IEDSAVE1--ADDRESS OF IEDSAVE1 WAS PUT
                                   449 *                                   IN REG. 13 WHICH WAS MADE BASE REGISTER
                          (5)      450          RETURN  (14,12),RC=0
                                   454 *
```

Figure 160. A Message Control Program for Communication Applications (Part 3 of 20)

```
LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                ASM 0102 19.28 06/07/74
                                  456 ****
                                  457 *
                                  458 * DATA DEFINITIONS--PROCESS CONTROL BLOCKS AND DATA CONTROL BLOCKS
                                  459 *
                                  460 ****
                                  461 *
                                  462 * PCB--PROCESS CONTROL BLOCK--USED TO COMMUNICATE BETWEEN THE MCP
                                  463 * AND AN APPLICATION PROGRAM--
                                  464 * ONE PCB IS NEEDED FOR EACH ACTIVE APPLICATION PROGRAM
                                  465 *
                                  466 * IN THE FOLLOWING MACRO--
                                  467 *   MH= GIVES THE SYMBOLIC ADDRESS OF THE MESSAGE HANDLER FOR THIS
                                  468 *     APPLICATION PROGRAM
                                  469 *   BUFSIZE= SPECIFIES SIZE OF BUFFERS TO HANDLE MESSAGES FOR
                                  470 *     APPLICATION PROGRAM
                                  471 *   BUFIN= INITIAL NO. OF BUFFERS INTO WHICH USERS WRITE WORK AREA
                                  472 *     EMPTIED--OPTIMUM NO. IS ENOUGH FOR ALL OF WORK AREA--BETWEEN
                                  473 *     2 AND 15--2 ASSUMED
                                  474 *   BUFOUT= INITIAL NO. OF BUFFERS THAT MAY BE FILLED IN ANTICIPATION
                                  475 *     OF A READ--BETWEEN 2 AND 15--2 ASSUMED
                                  476 *   RESERVE=NO. OF BYTES TO RESERVE FOR INSERTION OF CHARS. BY DATETIME
                                  477 *     AND SEQUENCE MACROS FOR MESSAGES COMING FROM APPLICATION PROGRAMS
                                  478 *   DATE=YES--THIS IS NEEDED FOR ALL PCB ENTRIES FOR A COBOL PROGRAM.
                                  479 *     THIS WILL MAKE THE DATE AND TIME AVAILABLE SO IT MAY BE PLACED
                                  480 *     IN THE COBOL PROGRAM INPUT CD--(IT IS ALSO NEEDED ON AN INPUT
                                  481 *     TPROCESS ENTRY)
                                  482 *
                                  483 * PROCESS CONTROL BLOCK FOR COBOL PROGRAM RUNNING WITH TERMINALS
                                  484 *
(6)                               485 PCBLK    PCB(a)  MH=MHTRMAPP,BUFSIZE=100,BUFIN=2,BUFOUT=5,RESERVE=21,     X
                                                       DATE=YES
                                  518 *
                                  519 * PROCESS CONTROL BLOCK FOR COBOL PROGRAMS THAT SIMULATE TERMINAL
                                  520 * INPUT DATA--USED FOR TESTING WITHOUT TERMINALS
                                  521 *
                                  522 PCBLK1   PCB(b)  MH=MHAPPAPP,BUFSIZE=100,BUFIN=2,BUFOUT=5,DATE=YES
                                  555 *
                                  556 * PROCESS CONTROL BLOCK FOR COBOL PROGRAMS TESTING MESSAGES SENT TO
                                  557 * DESTINATIONS DEFINED BY A QUEUE STRUCTURE
                                  558 *
                                  559 * IT USES THE SAME MH THAT PCKBLK1 USES
                                  560 *
                                  561 PCBLK2   PCB(c)  MH=MHAPPAPP,BUFSIZE=100,BUFIN=2,BUFOUT=5,DATE=YES
                                  594 *
                                  595 * DCBS
                                  596 *
                                  597 * DCB FOR MESSAGE QUEUES DATA SET
                                  598 *  IN THE FOLLOWING MACRO--
                                  599 *        OPTCD=R SPECIFIES REUSABLE DISK--IF NON-REUSABLE,SPECIFY L
                                  600 *        THRESH= SHOULD PROBABLY BE USED IF NON-REUSABLE DISK--
                                  601 *          SPECIFIES PERCENTAGE OF RECORDS TO BE USED BEFORE A FLUSH
                                  602 *          CLOSEDOWN INITIATED--A CERTAIN PERCENTAGE ASSUMED
(7)                               603 MSGQ     DCB(a)  DSORG=TQ,MACRF=(G,P),DDNAME=QFILE,OPTCD=R
                                  637 *
                                  638 * DCB FOR THE CHECKPOINT DATA SET
```

Figure 160. A Message Control Program for Communication Applications (Part 4 of 20)

```
LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                    ASM 0102 19.28 06/07/74
                                  639 CHKPT    DCB(b) DSORG=TQ,MACRF=(G,P),DDNAME=CFILE,OPTCD=C
                                  673 *
                                  674 * DCB FOR THE 1050 LINE GROUP
                                  675 *   IN THE FOLLOWING MACRO--
                                  676 *        CPRI=R INDICATES THAT RECEIVE HAS PRIORITY OVER SENDING--
                                  677 *             S INDICATES THAT SENDING HAS PRIORITY OVER RECEIVING--
                                  678 *             E INDICATES EQUAL PRIORITY--
                                  679 *          FOR SWITCHED LINES, S MUST BE SPECIFIED
                                  680 *        BUFIN=NO. OF BUFFERS TO ASSIGN INITIALLY FOR RECEIVING FOR
                                  681 *          EACH LINE--1 ASSUMED--15 MAXIMUM
                                  682 *        BUFOUT=NO. OF BUFFERS TO ASSIGN INITIALLY FOR SENDING FOR
                                  683 *          EACH LINE--2 ASSUMED--15 MAXIMUM
                                  684 *        BUFMAX=MAX. NO. OF BUFFERS TO BE USED FOR DATA TRANSFER FOR
                                  685 *          EACH LINE IN LINE GROUP--NO LESS THAN LARGER OF BUFIN AND
                                  686 *          BUFOUT--15 MAXIMUM
                                  687 *        BUFSIZE=BUFFER SIZE IN BYTES USED FOR ALL LINES IN THIS LINE
                                  688 *          GROUP--SIZE SHOULD BE A MULTIPLE OF THE BUFFER UNIT SIZE
                                  689 *          SPECIFIED IN KEYLEN= OPERAND OF INTRO MACRO-- (MAY BE
                                  690 *          OVERRIDDEN ON A STATION BASIS BY BUFSIZE= OPERAND OF THE
                                  691 *          TERMINAL MACRO)
                                  692 *        INVLIST=NAMES OF INVITATION LISTS FOR LINES OF LINE GROUP
                                  693 *          --INVITATION LIST NAMES ARE SPECIFIED ACCORDING TO THE
                                  694 *          ASCENDING RELATIVE LINE NOS. OF THE LINES IN THE GROUP
                                  695 *        MH=ADDRESS OF MESSAGE HANDLER
                                  696 *        PCI=SPECIFIES IF AND HOW A PROGRAM-CONTROLLED INTERRUPTION
                                  697 *          TO BE USED FOR BUFFER ALLOCATION AND DEALLOCATION--1ST
                                  698 *          SUBOPERAND REFERS TO RECEIVING AND 2ND TO SENDING--
                                  699 *          N SPECIFIES NO PCIS--R SPECIFIES AFTER 1ST BUFFER, COMPLETED
                                  700 *          BUFFER DEALLOCATED--A IS ASSUMED AND  SPECIFIES AFTER 1ST
                                  701 *          BUFFER, COMPLETED BUFFER DEALLOCATED AND ANOTHER BUFFER IS
                                  702 *          ALLOCATED
                                  703 *        RESERVE=NO. OF BYTES TO RESERVE FOR INSERTION OF CHARS. BY
                                  704 *          DATETIME AND SEQUENCE MACROS
                                  705 *        TRANS=TRANSLATION TABLE
                                  706 *        SCT=SPECIAL CHARACTERS TABLE
                                  707 *        (IF CPRI=R AND NON-SWITCHED LINE, NEED INTVL= OR NO MESSAGES
                                  708 *        ARE SENT--INTVL=NO. OF SECONDS TO DELAY AFTER PASS THRU
                                  709 *        INVITATION LIST--NO LARGER THAN 255--TOO SHORT A DELAY CAUSES
                                  710 *        MESSAGES TO ACCUMULATE)
                                  711 *
                                  712 LN1050   DCB(c) DSORG=TX,MACRF=(G,P),CPRI=S,DDNAME=LN1,BUFIN=2,          X
                                                      BUFOUT=4,BUFMAX=4,BUFSIZE=100,INVLIST=(LIST1050),       X
                                                      MH=MH1050,PCI=(A,A),RESERVE=21,TRANS=105F,SCT=105F
                                  749 *
                                  750 * DCB FOR THE TWX LINE--SEE DESCRIPTION OF OPERANDS BEFORE DCB FOR
                                  751 * 1050--LN1050
                                  752 *
                                  753 LNTWX    DCB(d) DSORG=TX,MACRF=(G,P),CPRI=S,DDNAME=LN2,BUFIN=2,          X
                                                      BUFOUT=4,BUFMAX=4,BUFSIZE=100,INVLIST=(LISTTWX),        X
                                                      MH=MHTWX,PCI=(A,A),RESERVE=21,TRANS=TTYC,SCT=TTYC
                                  790 *
                                  791 * DCB FOR LOG DATA SET
                                  792 *   IN THE FOLLOWING MACRO--                                             *
                                  793 *        BLKSIZE=--THE VALUE SHOULD BE THE SAME AS IN KEYLEN OPERAND OF
                                  794 *          INTRO MACRO                                                    *
```



```
LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                    ASM 0102 19.28 06/07/74
                                  795 *        NCP=--MAX. NO. OF BUFFER UNITS THAT MAY APPEAR IN A BUFFER     *
                                  796 *
                                  797 MSGLOG   DCB(e) DSORG=PS,MACRF=(W),DDNAME=LOGFILE,BLKSIZE=100,RECFM=F,  *X
                                                      NCP=2                                                    *
                                  848 *          (f)                                                  TEST***
                                  849 * DCB FOR SNAPS                                                        *
                                  850 DUMP     DCB    DSORG=PS,RECFM=VBA,MACRF=(W),LRECL=125,DDNAME=LRDUMP,  *X
                                                      BLKSIZE=882                                              *
```

Figure 160. A Message Control Program for Communication Applications (Part 5 of 20)

```
LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                ASM 0102 19.28 06/07/74
                                  902 ****
                                  903 *
                                  904 * TERMINAL AND LINE CONTROL--DEFINES  TERMINAL TABLE ENTRIES AND THE
                                  905 * INVITATION LISTS FOR EACH LINE
                                  906 *
                                  907 ****
                                  908 *
                                  909 * DEFINE THE TERMINAL TABLE
                                  910 *        LAST= NAME OF LAST ENTRY IN TABLE
                                  911 *        MAXLEN= NUMBER OF CHARACTERS IN LONGEST NAME
                                  912 *
                              (8) 913          TTABLE  LAST=D1,MAXLEN=5
                                  942 *                                                                    NOTE*
                                  943 * IF ANY OPTION MACROS ARE NEEDED, THEY GO HERE--DATA GOES IN ENTRIES *
                                  944 * USING THE OPDATA= OPERAND OF THE TERMINAL OR TPROCESS ENTRIES
                                  945 *
                                  946 * ENTRY FOR 1050 TERMINAL
                                  947 *   IN THE FOLLOWING MACRO--
                                  948 *        QBY= T SPECIFIES THAT OUTGOING MESSAGES ARE TO BE QUEUED BY
                                  949 *          TERMINAL--USE L IF BY LINE
                                  950 *          --MUST QUEUE BY TERMINAL IF A SWITCHED STATION OR A
                                  951 *          BUFFERED TERMINAL
                                  952 *        DCB= DCBNAME FOR LINE
                                  953 *        RLN=RELATIVE LINE NO. WITHIN THE LINE GROUP OF THIS LINE
                                  954 *        TERM=SPECIFIES TYPE OF TERMINAL
                                  955 *        QUEUES=MR SPECIFIES MESSAGE QUEUES KEPT IN MAIN STORAGE WITH
                                  956 *          BACKUP ON REUSABLE DISK
                                  957 *        ADDR=6213 IS A9 IN 1050 CODE--USED WHEN COMPUTER HAS MESSAGE
                                  958 *          TO SEND--9 IS CODE FOR ANY OUTPUT DEVICE
                                  959 *        ALTDEST=IS NEEDED BECAUSE THIS IS REUSABLE DISK--NEEDED SO
                                  960 *          MESSAGE IS NOT DISCARDED AT ZONE CHANGEOVER
                                  961 *        NTBLKSZ= THE NO. OF CHARS. BETWEEN INSERTION OF EOB CHARS.
                                  962 *          IN OUTPUT MSG. WHEN MSGFORM CODED IN OUTHDR
                                  963 *
                              (9) 964 T1       TERMINAL (a) QBY=T,DCB=LN1050,RLN=1,TERM=1050,QUEUES=MR,          X
                                                    ADDR=6213,ALTDEST=T1,NTBLKSZ=(120)
                                  994 *
                                  995 * DEFINE ENTRY FOR THE SWITCHED TWX LINE WHICH CAN BE USED BEFORE AN
                                  996 * ORIGIN MACRO IS ISSUED TO IDENTIFY THE STATION
                                  997 *        UTERM=YES IDENTIFIES THIS AS SUCH AN ENTRY
                                  998 * THIS MACRO MUST PRECEDE ALL TERMINAL MACROS FOR STATIONS ON LINE
                                  999 * IN THE FOLLOWING MACRO--
                                 1000 *        ALWAYS SPECIFY DCB NAME,RELATIVE LINE NO., TERMINAL TYPE,
                                 1001 *          AND QUEUES
                                 1002 *        --ADDR= MIGHT BE CODED IF STATION HAD ADDRESSING CHARS.--IF
                                 1003 *        USED, ALL STATIONS ON LINE MUST HAVE IDENTICAL ADDRESSING
                                 1004 *        CHARACTERS
                                 1005 *
                                 1006 T2A      TERMINAL (b) UTERM=YES,DCB=LNTWX,RLN=1,TERM=3335,QUEUES=MR
                                 1030 *
                                 1031 * TERMINAL ENTRY FOR TWX TERMINAL--SEE DESCRIPTION OF MOST OF OPERANDS
                                 1032 * PRECEDING TERMINAL MACRO FOR 1050
                                 1033 * IN ADDITION--
                                 1034 *        DIALNO= SPECIFIES TELEPHONE NO. OF STATION AND MUST BE
                                 1035 *          SPECIFIED FOR SWITCHED STATIONS--CODE 'NONE' IF NO AUTO
```

Figure 160. A Message Control Program for Communication Applications (Part 6 of 20)

```
LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                ASM 0102 19.28 06/07/74
                                 1036 *          CALL FEATURE
                                 1037 *        ADDR= IS NOT GIVEN SINCE THIS STATION IS ON A SWITCHED LINE
                                 1038 *        NTBLKSZ IS NOT USED FOR TWX TERMINALS
                                 1039 *        CINTVL= NO. OF SECONDS BEFORE COMPUTER SHOULD CALL STATION
                                 1040 *          --NOT NEEDED IF NO AUTO CALL FEATURE
                                 1041 *                  (c)
                                 1042 T2       TERMINAL QBY=T,DCB=LNTWX,RLN=1,TERM=3335,QUEUES=MR,          X
                                                     DIALNO=NONE,ALTDEST=T2
                                 1063 *
                                 1064 * TPROCESS ENTRIES
                                 1065 *
                                 1066 * IN THE FOLLOWING MACROS--
                                 1067 *        PCB= NAME OF PROCESS CONTROL BLOCK--ALL TPROCESS
                                 1068 *          ENTRIES FOR THE SAME APPLICATION PROGRAM MUST HAVE THE SAME
                                 1069 *          PCB
                                 1070 *        QUEUES= IS THE SAME AS FOR A TERMINAL MACRO--HOWEVER, BY
                                 1071 *          OMITTING, USER SPECIFIES THAT THIS ENTRY IS USED FOR PUTS &
                                 1072 *          WRITES FROM APPLICATION PROGRAM
                                 1073 *        ALTDEST= FOR OUTPUT,GIVES WHERE REPLIES TO OPERATOR MSGS. SENT
                                 1074 *          IF WERE ENTERED FROM AN APPLICATION PROGRAM--NOT APPLICABLE
                                 1075 *          TO COBOL--
                                 1076 *          ONLY NEEDED FOR INPUT QUEUES IF REUSABLE DISK QUEUEING
                                 1077 *        RECDEL= SPECIFIES CHARACTER USED TO DENOTE END OF RECORD
                                 1078 *        DATE=YES--THIS IS NEEDED FOR ALL INPUT TPROCESS ENTRIES
                                 1079 *          FOR A COBOL PROGRAM.  THIS WILL MAKE THE DATE AND TIME
                                 1080 *          AVAILABLE SO IT MAY BE PLACED IN THE COBOL PROGRAM
                                 1081 *          INPUT CD.
                                 1082 *
                                 1083 * INPUT TPROCESS ENTRY FOR COBOL PROGRAM RUNNING WITH TERMINALS
                          (10)   1084 *
                                 1085 PIN      TPROCESS  PCB=PCBLK,QUEUES=MR,ALTDEST=PIN,RECDEL=FF,DATE=YES (a)
                                 1114 *
                                 1115 * OUTPUT TPROCESS ENTRY FOR COBOL PROGRAM RUNNING WITH TERMINALS
                                 1116 *
                                 1117 POUT     TPROCESS  PCB=PCBLK,RECDEL=FF (b)
                                 1141 *
                                 1142 * THE FOLLOWING TWO INPUT TPROCESS ENTRIES ARE FOR COBOL PROGRAMS
                                 1143 * THAT SIMULATE TERMINAL INPUT DATA--USED FOR TESTING WITHOUT
                                 1144 * TERMINALS
                                 1145 *
                                 1146 P1       TPROCESS  PCB=PCBLK1,QUEUES=MR,ALTDEST=P1,RECDEL=FF,DATE=YES (c)
                                 1170 *
                                 1171 P2       TPROCESS  PCB=PCBLK1,QUEUES=MR,ALTDEST=P2,RECDEL=FF,DATE=YES (d)
                                 1195 *
                                 1196 * OUTPUT TPROCESS ENTRY FOR THESE COBOL PROGRAMS
                                 1197 *
                                 1198 POUT1    TPROCESS  PCB=PCBLK1,RECDEL=FF (e)
                                 1222 *
                                 1223 * THE FOLLOWING SIX INPUT TPROCESS ENTRIES ARE FOR COBOL QUEUE
                                 1224 * STRUCTURE TEST PROGRAMS
                                 1225 *
                                 1226 PQ1      TPROCESS  PCB=PCBLK2,QUEUES=MR,ALTDEST=PQ1,RECDEL=FF,DATE=YES
                                 1250 *
                                 1251 PQ2      TPROCESS  PCB=PCBLK2,QUEUES=MR,ALTDEST=PQ2,RECDEL=FF,DATE=YES
                                 1275 *
```

Figure 160. A Message Control Program for Communication Applications (Part 7 of 20)

```
LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                    ASM 0102 19.28 06/07/74

                                 1276 PQ3      TPROCESS  PCB=PCBLK2,QUEUES=MR,ALTDEST=PQ3,RECDEL=FF,DATE=YES
                                 1300 *
                                 1301 PQ4      TPROCESS  PCB=PCBLK2,QUEUES=MR,ALTDEST=PQ4,RECDEL=FF,DATE=YES
                                 1325 *
                                 1326 PQ5      TPROCESS  PCB=PCBLK2,QUEUES=MR,ALTDEST=PQ5,RECDEL=FF,DATE=YES
                                 1350 *
                                 1351 PQ6      TPROCESS  PCB=PCBLK2,QUEUES=MR,ALTDEST=PQ6,RECDEL=FF,DATE=YES
                                 1375 *
                                 1376 * OUTPUT TPROCESS ENTRY FOR COBOL QUEUE STRUCTURE TEST PROGRAMS
                                 1377 *
                                 1378 PQOUT    TPROCESS  PCB=PCBLK2,RECDEL=FF
                                 1402 *
                                 1403 *
                                 1404 * DISTRIBUTION LIST ENTRY --
                                 1405 *
                                 1406 * IN THE FOLLOWING MACRO --
                                 1407 *        LIST = NAMES OF TERMINAL OR TPROCESS ENTRIES IN THE
                                 1408 *          TERMINAL TABLE
                                 1409 *
                                 1410 *        THE LIST SHOULD NOT INCLUDE A TPROCESS ENTRY FOR A
                                 1411 *        COBOL APPLICATION PROGRAM
                                 1412 *
                                 1413 *        TYPE= D SPECIFIES THIS IS A DISTRIBUTION LIST ENTRY
                                 1414 *              C WOULD SPECIFY A CASCADE LIST ENTRY
                                 1415 *              DISTRIBUTION LISTS INDICATE A MESSAGE FORWARDED TO THEM
                                 1416 *              WILL BE SENT TO ALL NAMES IN THE LIST
                                 1417 *
                                 1418 *              WITH CASCADE LISTS, MESSAGES WILL BE SENT TO THE QUEUE
                                 1419 *              SPECIFIED IN THE LIST WITH THE FEWEST NO. OF MESSAGES
                                 1420 *
                                 1421 * 1050 AND TWX--USED BY MESSAGE PROCESSING PROGRAM
                                 1422 *
(11)                             1423 D1       TLIST  LIST=(T1,T2),TYPE=D  (a)
                                 1451 *
                                 1452 *
                                 1453 * INVITATION LISTS
                                 1454 * SHOULD ALWAYS BE SPECIFIED FOLLOWING THE MACROS DEFINING THE TERMINAL
                                 1455 * TABLE
                                 1456 *
                                 1457 * LIST FOR 1050 LINE--
                                 1458 *        ORDER= ENTRIES FOR STATIONS ON LINE IN THE ORDER TO BE POLLED
                                 1459 *          T1 SPECIFIES A STATION ON THE LINE DEFINED BY A TERMINAL
                                 1460 *            MACRO
                                 1461 *          + SPECIFIES THE TERMINAL IS INITIALLY ACTIVE, - WOULD
                                 1462 *            SPECIFY IT WAS INITIALLY INACTIVE
                                 1463 *        6215=A0 IN 1050 CODE--A IS THE STATION ADDRESS--0 ASKS FOR
                                 1464 *            INPUT FROM ANY INPUT COMPONENT
                                 1465 *
(12)                             1466 LIST1050 INVLIST  ORDER=(T1+6215) (a)
                                 1477 *
                                 1478 * LIST FOR TWX LINE--
                                 1479 *   SINCE A TERMINAL MACRO WITH UTERM=YES WAS DEFINED FOR THIS LINE,
                                 1480 *   THIS MACRO NAME IS USED RATHER THAN THE ONE FOR THE TWX STATION
                                 1481 *
                                 1482 * THIS IS A SWITCHED LINE WHICH DOES NOT HAVE THE AUTO-CALL FEATURE--
```



```
LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                    ASM 0102 19.28 06/07/74

                                 1483 * THE COMPUTER NEVER ASKS FOR THE ID SEQUENCE FROM THE TWX TERMINAL
                                 1484 * UNLESS THE AUTO-CALL FEATURE IS PRESENT
                                 1485 *
                                 1486 * IF AUTO-CALL FEATURE IS NOT PRESENT OR TWX TERMINAL DOES NOT HAVE
                                 1487 * AN ID SEQUENCE FOR AN ANSWER-BACK, OMIT THE ID SEQUENCE CHARS. IN
                                 1488 * THE INVLIST MACRO
                                 1489 *
                                 1490 * IF AN ID SEQUENCE IS USED FOR THE TWX--IT IS SUGGESTED THE
                                 1491 * FOLLOWING CHARACTERS BE USED -- CR LF IDCHARS CR LF XON--IN LINE
                                 1492 * CODE
                                 1493 *
                                 1494 * THE CPUID OPERAND IS NEEDED FOR TWX TERMINALS--IT WILL PRINT AT
                                 1495 * TERMINAL WHEN CONNECTION IS MADE
                                 1496 *
                                 1497 LISTTWX  INVLIST  ORDER=(T2A+),CPUID=TWXSEQ (b)
                                 1508 *
                                 1509 * REFERENCED BY LISTTWX AS CPUID OPERAND
                                 1510 * -- SUGGESTED USE NULL CR LF RUBOUT IDCHARS CR LF XON
                                 1511 * CPUID IS -- COBOL
000864 0C                        1512 TWXSEQ   DC    X'0C'          12 CHARACTERS
000865 01B151FFC3F343F3          1513          DC    X'01B151FFC3F343F333B15189'
                                 1514 *
```

Figure 160. A Message Control Program for Communication Applications (Part 8 of 20)

```
LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                    ASM 0102 19.28 06/07/74
                                 1516 ****
                                 1517 *
                                 1518 * MESSAGE HANDLERS--MH'S
                                 1519 *
                                 1520 * THE HEADER RECEIVED FROM THE TERMINAL IS--
                                 1521 *        POSSIBLE LINE FORMAT CHARS.--CR,LF,NL
                                 1522 *        $
                                 1523 *        BLANK
                                 1524 *        MSGTYPE--1 CHAR.
                                 1525 *        BLANK
                                 1526 *        SOURCE--2 CHARS.
                                 1527 *        BLANK
                                 1528 *        EOF FIELD--F IF END OF A GROUP OF MESSAGES
                                 1529 *                  --ANY OTHER CHAR. (EXCEPT BLANK) IF NOT
                                 1530 *        BLANK
                                 1531 *        ACTION CODE FOR APPLICATION PROGRAM--2 CHARS.
                                 1532 *        BLANK
                                 1533 *        PUNCTUATION MARK--PERIOD
                                 1534 *
                                 1535 ****
                                 1536 ***
                                 1537 * MESSAGE HANDLER FOR INPUT FROM AND OUTPUT TO 1050 TERMINAL
                                 1538 *
                                 1539 * THE FOLLOWING MACRO IS REQUIRED AND MUST BE FIRST
                                 1540 *        LC= IS THE ONLY REQUIRED OPERAND--
                                 1541 *          OUT SAYS TO REMOVE LINE CONTROL CHARS.
                                 1542 *          IN SAYS NOT TO REMOVE LINE CONTROL CHARS.
                                 1543 *        STOP= SAYS WHEN EOB ERROR FOUND AND RETRY COUNT EXHAUSTED,
                                 1544 *          ONLY THAT PORTION OF MESSAGE RECEIVED OR SENT CONTINUES
                                 1545 *          THRU MH--USER MAY CHECK ERROR RECORD BITS IN INMSG OR OUTMSG
                                 1546 *        CONT= SAYS THAT AFTER RETRY, SET BIT IN ERROR RECORD--BUT
                                 1547 *          CONTINUE TRANSMISSION
                                 1548 * IF NEITHER STOP NOR CONT SPECIFIED,NO EOB CHECKING PERFORMED
                                 1549 *
(13)                             1550 MH1050   STARTMH  LC=OUT,CONT=YES
                                 1572 *
                                 1573 * THE FOLLOWING MACRO IS REQUIRED AS THE FIRST MACRO IN ANY INCOMING
                                 1574 * GROUP
(14)                             1575          INHDR
                                 1589 *
                                 1590 * THE FOLLOWING MACRO TRANSLATES FROM LINE CODE TO EBCDIC--MACROS
                                 1591 * FOLLOWING THIS WILL ACT UPON CHARACTERS IN EBCDIC--IT WILL CAUSE
                                 1592 * ENTIRE MESSAGE TO BE TRANSLATED EVEN THOUGH IN INHDR GROUP
(15)                             1593          CODE
                                 1617 *
                                 1618 * LOG INCOMING HEADERS--USE DCBNAME AS OPERAND
                                 1619 *                                                    NEXT IWD DISABLED--*
(16)                             1620 *        LOG   MSGLOG
                                 1621 *
                                 1622 * SET SCAN POINTER TO $
(17)                             1623          SETSCAN C'$'
                                 1639 *
                                 1640 * PROCESS THE REMAINDER OF THE HEADER ACCORDING TO THE MSGTYPE FIELD
                                 1641 * SPECIFIED NEXT IN THE HEADER--IF THE NEXT FIELD MATCHES THE CHARACTER
                                 1642 * SPECIFIED IN THE OPERAND, THE MACROS SPECIFIED BETWEEN IT AND THE
```

Figure 160. A Message Control Program for Communication Applications (Part 9 of 20)

```
LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                ASM 0102 19.28 06/07/74
                                 1643 * NEXT MSGTYPE MACRO ARE EXECUTED AND CONTROL IS THEN PASSED TO THE
                                 1644 * NEXT DELIMITER--IN THIS CASE INBUF -IF THEY DO NOT MATCH, CONTROL
                                 1645 * PASSES TO THE NEXT MSGTYPE MACRO WHERE THE TEST IS AGAIN MADE
                                 1646 *
                                 1647 * IF MSGTYPE IS 1, THIS MESSAGE SHOULD BE FORWARDED TO THE 1050
                                 1648         MSGTYPE  C'1'
                                 1664 *
                                 1665 * SCAN POINTER IS AT SOURCE FIELD--SINCE THIS IS A NON-SWITCHED STATION
                                 1666 * --ORIGIN VERIFIES THAT THE SOURCE FIELD CONTAINS THE SYMBOLIC NAME
                                 1667 * OF THE STATION THAT WAS INVITED TO SEND THE MESSAGE--IF NOT, ERROR
                                 1668 * BIT IN ERROR RECORD FOR MESSAGE IS SET TO 1
                                 1669         ORIGIN
                                 1682         FORWARD  DEST=C'T1'
                                 1700 *
                                 1701 * IF MSGTYPE IS 2, THIS MESSAGE SHOULD BE FORWARDED TO TWX TERMINAL--
                                 1702 * SEE COMMENTS UNDER MSGTYPE 1 FOR OTHER MACROS
                                 1703         MSGTYPE  C'2'
                                 1721         ORIGIN
                                 1731         FORWARD  DEST=C'T2'
                                 1746 *
                                 1747 * IF MSGTYPE IS 5, THIS MESSAGE SHOULD BE FORWARDED TO THE COBOL
                                 1748 * APPLICATION PROGRAM--
                                 1749 * SEE COMMENTS UNDER MSGTYPE 1 FOR OTHER MACROS
                                 1750         MSGTYPE  C'5'
                                 1768         ORIGIN
                                 1778         FORWARD  DEST=C'PIN'
                                 1793 *
                                 1794 * IF MSGTYPE IS 6, THE SOURCE FIELD HAS BEEN OMITTED--UNNECESSARY TO
                                 1795 * ISSUE AN ORIGIN FOR A NON-SWITCHED LINE--SEND MESSAGE TO THE COBOL
                                 1796 * APPLICATION PROGRAM
                                 1797         MSGTYPE  C'6'
                                 1815         FORWARD  DEST=C'PIN'
                                 1830 *
                                 1831 * IF THE MSGTYPE IS ANYTHING ELSE, IT IS INVALID--SET THE USER ERROR
                                 1832 * BIT WITH THE TERRSET MACRO--IN THE INMSG GROUP, WE WILL CANCEL MSG.--
                                 1833 * ISSUE FORWARD MACRO ANYWAY SINCE REQUIRED
                                 1834         MSGTYPE
                                 1839         FORWARD  DEST=C'T1'
                                 1854         TERRSET
                                 1861 *
                                 1862 * THE MACROS IN THE FOLLOWING SUBGROUP ARE EXECUTED FOR EVERY BUFFER
                                 1863 * OF THE MESSAGE
                                 1864         INBUF
                                 1869 *
                                 1870 * SPECIFY THE MAXIMUM NO. OF CHARACTERS ALLOWED IN AN INCOMING MESSAGE
                                 1871 * --THIS MACRO ALSO CHECKS IF THE INPUT BUFFER IS FILLED WITH IDENTICAL
                                 1872 * CHARACTERS, USUALLY AN INDICATION OF STATION MALFUNCTION--SETS A
                                 1873 * BIT IN ERROR RECORD FOR EITHER CONDITION
                                 1874         CUTOFF  900
                                 1885 *
                                 1886 * INSERT X'FF' FOR EVERY NL AND LF CHARACTER--X'FF' IS THE RECDEL CHAR.
                                 1887 * SPECIFIED IN THE TPROCESS MACROS--IF A MESSAGE WERE ALWAYS BEING
                                 1888 * FORWARDED TO AN APPLICATION PROGRAM, WE COULD USE DELIMIT INSTEAD
                                 1889 * OF XL1'FF'
                                 1890         'MSGEDIT ((RA,XL1'FF',XL1'15'),(RA,XL1'FF',XL1'25'))
```

Figure 160. A Message Control Program for Communication Applications (Part 10 of 20)

```
LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                ASM 0102 19.28 06/07/74

                                 1919 *
                                 1920 * THE INMSG SUBGROUP IS SPECIFIED AFTER OTHER SUBGROUPS--IT IS EXECUTED
                                 1921 * AFTER AN ENTIRE MESSAGE OR BLOCK HAS BEEN PROCESSED--NO EXECUTABLE
                                 1922 * USER-WRITTEN CODE SHOULD BE INCLUDED IN THIS SUBGROUP
                                 1923          INMSG
                                 1931 *
                                 1932 * CANCELMG CAUSES IMMEDIATE CANCELLATION OF MESSAGE IF ANY ERRORS
                                 1933 * SPECIFIED BY ITS MASK OCCUR--IF USED, IT MUST BE 1ST MACRO UNDER
                                 1934 * INMSG--AN ERRORMSG MACRO MAY THEN NOTIFY OF THE ERROR--
                                 1935 * CANCELMG IF THE USER ERROR BIT IS SET INDICATING THE MSGTYPE FIELD
                                 1936 * WAS INVALID--BIT20
                                 1937          CANCELMG  X'0000080000'
                                 1945 *
                                 1946 * IN THE FOLLOWING ERROR MESSAGES, THE 1ST FIELD IS THE MASK CORRE-
                                 1947 * SPONDING TO THE BITS IN THE ERROR RECORD,DEST= IS ALWAYS T1 FOR THE
                                 1948 * 1050 TERMINAL AND THE DATA= IS THE ERROR MESSAGE THAT IS SENT--
                                 1949 * THE MESSAGE INCLUDES THE HEADER OF THE MESSAGE IN ERROR AND THE
                                 1950 * ERROR MESSAGE
                                 1951 *
                                 1952 * THE LAST CHARACTER OF THE MESSAGE IS NL--SO THE CARRIAGE WILL BE
                                 1953 * RETURNED WITH A LINE FEED AT THE END OF THE PRINTING OF THE MESSAGE
                                 1954          ERRORMSG  X'8000000000',DEST=C'T1',                              X
                                                     DATA=C'E ERROR IN PROCESSING HEADER '
                                 1970          ERRORMSG  X'4000000000',DEST=C'T1',                              X
                                                     DATA=C'E INVALID ORIGIN IN HEADER '
                                 1982          ERRORMSG  X'0200000000',DEST=C'T1',                              X
                                                     DATA=C'E INSUFFICIENT BUFFERS FOR INCOMING MESSAGE '
                                 1994          ERRORMSG  X'0100000000',DEST=C'T1',                              X
                                                     DATA=C'E MESSAGE TOO LONG '
                                 2006 *
                                 2007 * THE FOLLOWING ERROR MESSAGE SHOULD ONLY OCCUR WITH MAIN STORAGE
                                 2008 * QUEUEING WITH OR WITHOUT DISK BACKUP
                                 2009          ERRORMSG  X'0040000000',DEST=C'T1',                              X
                                                     DATA=C'E PERCENTAGE OF BUFFER UNITS IN BUFMAX ARE USED-SX
                                                     LOW DOWN '
                                 2021 *
                                 2022          ERRORMSG  X'0002000000',DEST=C'T1',                              X
                                                     DATA=C'E FORWARDED TO INVALID DESTINATION '
                                 2034          ERRORMSG  X'0000400000',DEST=C'T1',                              X
                                                     DATA=C'E INVALID STATION ID AT CONNECT TIME '
                                 2046          ERRORMSG  X'0000200000',DEST=C'T1',                              X
                                                     DATA=C'E TERMINAL IS IN HOLD STATUS '
                                 2058          ERRORMSG  X'0000080000',DEST=C'T1',                              X
                                                     DATA=C'E MSGTYPE CODE IN HEADER INVALID '
                                 2070          ERRORMSG  X'000000E000',DEST=C'T1',                              X
                                                     DATA=C'E A HARDWARE ERROR HAS OCCURRED '
                                 2082 *
                                 2083 * INEND IS REQUIRED AS LAST DELIMITER MACRO OF INCOMING GROUP
                                 2084          INEND
                                 2088 *
                                 2089 ***
                                 2090 *
                                 2091 * OUTGOING GROUP OF MESSAGE HANDLER FOR 1050 TERMINAL
                                 2092          OUTHDR
                                 2098 *
```

Figure 160. A Message Control Program for Communication Applications (Part 11 of 20)

```
LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                ASM 0102 19.28 06/07/74

                                 2099 * THE FOLLOWING MACRO CAUSES EOT LINE CONTROL CHARACTERS TO BE INSERTED
                                 2100 * IN EACH OUTGOING MESSAGE--SINCE NTBLKSZ=(BLKSIZE) CODED IN THE
                                 2101 * TERMINAL MACRO--IT ALSO INSERTS EOB CHARS.--THIS PARAMETER COULD
                                 2102 * ALSO BE PLACED AS AN OPERAND OF THIS MACRO TO OVERRIDE THE NO.
                                 2103 * SPECIFIED IN THE TERMINAL MACRO
(30)                             2104         MSGFORM
                                 2115 *
                                 2116 * SINCE ERROR MESSAGES ARE SENT TO THIS TERMINAL--AND THESE COULD
                                 2117 * INCLUDE THOSE FOR THE APPLICATION TO APPLICATION PROGRAM WHICH
                                 2118 * WILL NOT HAVE A HEADER AND CANNOT BE PROCESSED AS A NORMAL OUTPUT
                                 2119 * MESSAGE TO THIS TERMINAL--CHECK 1ST CHARACTER FOR AN E--THE 1ST
                                 2120 * CHAR. OF EVERY ERRORMSG--IF NOT E WILL SKIP TO NEXT MSGTYPE MACRO--
                                 2121 * IF E, WILL PROCESS TO NEXT MSGTYPE MACRO AND THEN SKIP TO NEXT
                                 2122 * DELIMITER--OUTBUF
                                 2123         MSGTYPE   C'E'
                                 2139 *
                                 2140 * SET SCAN POINTER BACK TO BEGINNING OF BUFFER AND INSERT NL CHARACTER
                                 2141 * AT BEGINNING OF MESSAGE--IDLES WILL BE INSERTED AFTER NL IN OUTBUF
                                 2142         SETSCAN  1,POINT=BACK
                                 2153         MSGEDIT  ((I,XL1'15',SCAN))
                                 2168 *
                                 2169 * USE MSGTYPE WITH BLANK OPERAND TO PROCESS OTHER MESSAGES
                                 2170         MSGTYPE
                                 2175 *
                                 2176 * INSERT NL CHARACTER AT BEGINNING OF MESSAGE--IDLES WILL BE INSERTED
                                 2177 * AFTER NL IN OUTBUF
                                 2178         MSGEDIT  ((I,XL1'15',SCAN))
                                 2190 *
                                 2191 * SET THE SCAN POINTER TO THE PERIOD IN THE HEADER AND INSERT DATE,
                                 2192 * TIME, AND SEQUENCE NO.--INSERTED IN EBCDIC SO DO BEFORE CODE
                                 2193         SETSCAN C'.'
                                 2206 *
                                 2207 * IF NO OPERAND--BOTH DATE AND TIME ARE INSERTED--SPACE MUST BE
                                 2208 * RESERVED BY MEANS OF THE RESERVE= OPERAND OF DCB FOR LINE--THE DATE
                                 2209 * IS IN FORM--(BLANK)YY.DDD--7 CHARS.--TIME IN FORM--
                                 2210 * (BLANK) HH.MM.SS--9 CHARACTERS
(31)                             2211         DATETIME
                                 2227 *
                                 2228 * SEQUENCE IN AN OUTHDR SUBGROUP INSERTS SEQUENCE NO. IN FORM--
                                 2229 * (BLANK)NNNN--5 CHARS.--SPACE MUST BE RESERVED BY MEANS OF RESERVE=
                                 2230 * OPERAND OF DCB FOR LINE
(32)                             2231         SEQUENCE
                                 2241 *
                                 2242 * LOG OUTGOING HEADERS--USE DCBNAME AS OPERAND--PUT MACRO AFTER
                                 2243 * INSERTION OF DATE, TIME, AND SEQUENCE NOS. SO THESE WILL APPEAR
                                 2244 * IN LOGGED HEADER
                                 2245 *                                                NEXT IWD DISABLED--*
                                 2246 *       LOG   MSGLOC
                                 2247 *
                                 2248 * THE MACROS IN THE FOLLOWING SUBGROUP ARE EXECUTED FOR EVERY BUFFER
                                 2249 * OF THE MESSAGE
(33)                             2250         OUTBUF
                                 2255 *
                                 2256 * INSERT NL CHAR. FOR EVERY X'FF' CHAR. IN MESSAGE--X'FF' IS THE
                                 2257 * RECDEL CHAR. SPECIFIED IN THE TPROCESS MACROS
```

Figure 160. A Message Control Program for Communication Applications (Part 12 of 20)

```
LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                ASM 0102 19.28 06/07/74
                                 2258          MSGEDIT  ((RA,XL1'15',XL1'FF'))
                                 2276 *
                                 2277 * INSERT 13 IDLE CHARS. AFTER EVERY NL CHARACTER PLACED IN MESSAGE
                                 2278          MSGEDIT  ((I,(X'17',13),XL1'15'))
                                 2295 *
                                 2296 * TRANSLATE THE MESSAGE FROM EBCDIC TO LINE CODE--IF ISSUED IN A
                                 2297 * SUBGROUP AND ANY SEGMENTS OF A MESSAGE PROCESSED BY THAT SUBGROUP,
                                 2298 * THE ENTIRE MESSAGE IS TRANSLATED
                                 2299          CODE
                                 2308 *
                                 2309 * THE OUTMSG SUBGROUP IS SPECIFIED AFTER OTHER SUBGROUPS IN OUTGOING
                                 2310 * GROUP--IT IS EXECUTED ONLY AFTER AN ENTIRE BLOCK OR MESSAGE HAS BEEN
                                 2311 * SENT
                              (34) 2312          OUTMSG
                                 2321 *
                                 2322 * THE HOLD MACRO SUSPENDS TRANSMISSION TO A STATION EITHER FOR A TIME
                                 2323 * INTERVAL (IF SPECIFIED) OR UNTIL RELEASED BY AN OPERATOR CONTROL
                                 2324 * MESSAGE--IF NOT USED, MESSAGES THAT CANNOT BE TRANSMITTED ARE
                                 2325 * TREATED AS THOUGH THEY HAVE BEEN TRANSMITTED--ALSO, A HOLD OPERATOR
                                 2326 * CONTROL MESSAGE HAS NO EFFECT IF THERE IS NO HOLD MACRO--
                                 2327 * BITS BEING TESTED BY MASK ARE FOR HARDWARE ERRORS
                              (35) 2328          HOLD  X'000000E000'
                                 2340 *
                                 2341 * IN THE FOLLOWING ERROR MESSAGES, THE 1ST FIELD IS THE MASK CORRE-
                                 2342 * SPONDING TO THE BITS IN THE ERROR RECORD,DEST= IS ALWAYS T1 FOR THE
                                 2343 * 1050 TERMINAL AND THE DATA= IS THE ERROR MESSAGE THAT IS SENT--
                                 2344 * THE MESSAGE INCLUDES THE HEADER OF THE MESSAGE IN ERROR AND THE
                                 2345 * ERROR MESSAGE
                                 2346 *
                                 2347 * THE LAST CHARACTER OF THE MESSAGE IS NL--SO THE CARRIAGE WILL BE
                                 2348 * RETURNED WITH A LINE FEED AT THE END OF THE PRINTING OF THE MESSAGE
                                 2349          ERRORMSG  X'8000000000',DEST=C'T1',                              X
                                                     DATA=C'E ERROR IN PROCESSING HEADER '
                                 2361 *
                                 2362 * THE FOLLOWING ERROR MESSAGE SHOULD ONLY OCCUR WITH MAIN STORAGE
                                 2363 * QUEUEING WITH OR WITHOUT DISK BACKUP
                                 2364          ERRORMSG  X'0040000000',DEST=C'T1',                              X
                                                     DATA=C'E PERCENTAGE OF BUFFER UNITS IN BUFMAX ARE USED-SX
                                                     LOW DOWN '
                                 2376 *
                                 2377          ERRORMSG  X'0000400000',DEST=C'T1',                              X
                                                     DATA=C'E INVALID STATION ID AT CONNECT TIME '
                                 2389          ERRORMSG  X'0000200000',DEST=C'T1',                              X
                                                     DATA=C'E TERMINAL IS IN HOLD STATUS '
                                 2401          ERRORMSG  X'000000E000',DEST=C'T1',                              X
                                                     DATA=C'E A HARDWARE ERROR HAS OCCURRED '
                                 2413 *
                                 2414 * OUTEND REQUIRED AS LAST DELIMITER MACRO OF OUTGOING GROUP
                              (36) 2415          OUTEND
                                 2419 *
                                 2420 * A LTORG SHOULD BE CODED AFTER LAST DELIMITER OF EACH MH IF MCP HAS
                                 2421 * MORE THAN 1 MH
000CD0                           2422          LTORG
                                 2423 *
```

Figure 160. A Message Control Program for Communication Applications (Part 13 of 20)

```
LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                    ASM 0102 19.28 06/07/74
                                 2425 ***
                                 2426 *
                                 2427 * MESSAGE HANDLER FOR INPUT FROM AND OUTPUT TO TWX TERMINAL
                                 2428 *
                                 2429 * THE FOLLOWING MACRO IS REQUIRED AND MUST BE FIRST
                                 2430 *         LC= IS THE ONLY REQUIRED OPERAND--
                                 2431 *           OUT SAYS TO REMOVE LINE CONTROL CHARS.
                                 2432 *           IN SAYS NOT TO REMOVE LINE CONTROL CHARS.
                                 2433 *
                                 2434 MHTWX     STARTMH  LC=OUT
                                 2450 *
                                 2451 * THE FOLLOWING MACRO IS REQUIRED AS THE FIRST MACRO IN ANY INCOMING
                                 2452 * GROUP
                                 2453          INHDR
                                 2464 *
                                 2465 * THE FOLLOWING MACRO TRANSLATES FROM LINE CODE TO EBCDIC--MACROS
                                 2466 * FOLLOWING THIS WILL ACT UPON CHARACTERS IN EBCDIC--IT WILL CAUSE
                                 2467 * ENTIRE MESSAGE TO BE TRANSLATED EVEN THOUGH IN INHDR GROUP
                                 2468          CODE
                                 2488 *
                                 2489 * LOG INCOMING HEADERS--USE DCBNAME AS OPERAND
                                 2490 *                                                    NEXT IWD DISABLED--*
                                 2491 *         LOG    MSGLOG
                                 2492 *
                                 2493 * SET SCAN POINTER TO $
                                 2494          SETSCAN C'$'
                                 2507 *
                                 2508 * PROCESS THE REMAINDER OF THE HEADER ACCORDING TO THE MSGTYPE FIELD
                                 2509 * SPECIFIED NEXT IN THE HEADER--IF THE NEXT FIELD MATCHES THE CHARACTER
                                 2510 * SPECIFIED IN THE OPERAND, THE MACROS SPECIFIED BETWEEN IT AND THE
                                 2511 * NEXT MSGTYPE MACRO ARE EXECUTED AND CONTROL IS THEN PASSED TO THE
                                 2512 * NEXT DELIMITER--IN THIS CASE INBUF -IF THEY DO NOT MATCH, CONTROL
                                 2513 * PASSES TO THE NEXT MSGTYPE MACRO WHERE THE TEST IS AGAIN MADE
                                 2514 *
                                 2515 * IF MSGTYPE IS 1, THIS MESSAGE SHOULD BE FORWARDED TO THE 1050
                                 2516          MSGTYPE  C'1'
                                 2532 *
                                 2533 * SCAN POINTER IS AT SOURCE--ISSUE ORIGIN--SINCE THIS IS A SWITCHED
                                 2534 * LINE,ORIGIN WILL CHECK VALIDITY OF FIELD AND IDENTIFY THE CALLING
                                 2535 * STATION TO TCAM
                                 2536          ORIGIN
                                 2546          FORWARD  DEST=C'T1'
                                 2561 *
                                 2562 * IF MSGTYPE IS 2, THIS MESSAGE SHOULD BE FORWARDED TO TWX TERMINAL--
                                 2563 * SEE COMMENTS UNDER MSGTYPE 1 FOR OTHER MACROS
                                 2564          MSGTYPE  C'2'
                                 2582          ORIGIN
                                 2592          FORWARD  DEST=C'T2'
                                 2607 *
                                 2608 * IF MSGTYPE IS 5, THIS MESSAGE SHOULD BE FORWARDED TO THE COBOL
                                 2609 * APPLICATION PROGRAM--
                                 2610 * SEE COMMENTS UNDER MSGTYPE 1 FOR OTHER MACROS
                                 2611          MSGTYPE  C'5'
                                 2629          ORIGIN
                                 2639          FORWARD  DEST=C'PIN'
```

Figure 160. A Message Control Program for Communication Applications (Part 14 of 20)

```
LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                    ASM 0102 19.28 06/07/74

                                 2654 *
                                 2655 * IF MSGTYPE IS 6, THE SOURCE FIELD HAS BEEN OMITTED (IN ORDER FOR
                                 2656 * THE COBOL PROGRAM TO CHECK THAT THE LINE NAME--T2A--RATHER THAN THE
                                 2657 * STATION NAME--T2--IS GIVEN AS SOURCE)--THE MESSAGE IS TO BE SENT TO
                                 2658 * THE COBOL APPLICATION PROGRAM
                                 2659         MSGTYPE  C'6'
                                 2677         FORWARD  DEST=C'PIN'
                                 2692 *
                                 2693 * IF THE MSGTYPE IS ANYTHING ELSE, IT IS INVALID--SET THE USER ERROR
                                 2694 * BIT WITH THE TERRSET MACRO--IN THE INMSG GROUP, WE WILL CANCEL MSG.--
                                 2695 * ISSUE FORWARD MACRO ANYWAY SINCE REQUIRED
                                 2696         MSGTYPE
                                 2701         FORWARD  DEST=C'T1'
                                 2716         TERRSET
                                 2723 *
                                 2724 * THE MACROS IN THE FOLLOWING SUBGROUP ARE EXECUTED FOR EVERY BUFFER
                                 2725 * OF THE MESSAGE
                                 2726         INBUF
                                 2731 *
                                 2732 * SPECIFY THE MAXIMUM NO. OF CHARACTERS ALLOWED IN AN INCOMING MESSAGE
                                 2733 * --THIS MACRO ALSO CHECKS IF THE INPUT BUFFER IS FILLED WITH IDENTICAL
                                 2734 * CHARACTERS, USUALLY AN INDICATION OF STATION MALFUNCTION--SETS A
                                 2735 * BIT IN ERROR RECORD FOR EITHER CONDITION
                                 2736         CUTOFF  900
                                 2744 *
                                 2745 * DELETE EVERY CR CHAR. AND INSERT X'FF' FOR EVERY LF CHAR.--X'FF'
                                 2746 * IS THE RECDEL CHARACTER SPECIFIED IN THE TPROCESS MACROS (IF MESSAGES
                                 2747 * WERE ALWAYS GOING TO AN APPLICATION PROGRAM, WE COULD USE DELIMIT
                                 2748 * INSTEAD OF XL1'FF')
                                 2749         MSGEDIT  ((RA,CONTRACT,XL1'26'),(RA,XL1'FF',XL1'15'))
                                 2772 *
                                 2773 * THE INMSG SUBGROUP IS SPECIFIED AFTER OTHER SUBGROUPS--IT IS EXECUTED
                                 2774 * AFTER AN ENTIRE MESSAGE OR BLOCK HAS BEEN PROCESSED--NO EXECUTABLE
                                 2775 * USER-WRITTEN CODE SHOULD BE INCLUDED IN THIS SUBGROUP
                                 2776         INMSG
                                 2784 * CANCELMG CAUSES IMMEDIATE CANCELLATION OF MESSAGE IF ANY ERRORS
                                 2785 *
                                 2786 * SPECIFIED BY ITS MASK OCCUR--IF USED, IT MUST BE 1ST MACRO UNDER
                                 2787 * INMSG--AN ERRORMSG MACRO MAY THEN NOTIFY OF THE ERROR--
                                 2788 * CANCELMG IF THE USER ERROR BIT IS SET INDICATING THE MSGTYPE FIELD
                                 2789 * WAS INVALID--BIT20
                                 2790         CANCELMG  X'0000080000'
                                 2795 *
                                 2796 * IN THE FOLLOWING ERROR MESSAGES, THE 1ST FIELD IS THE MASK CORRE-
                                 2797 * SPONDING TO THE BITS IN THE ERROR RECORD,DEST= IS ALWAYS T1 FOR THE
                                 2798 * 1050 TERMINAL AND THE DATA= IS THE ERROR MESSAGE THAT IS SENT--
                                 2799 * THE MESSAGE INCLUDES THE HEADER OF THE MESSAGE IN ERROR AND THE
                                 2800 * ERROR MESSAGE
                                 2801 *
                                 2802 * THE LAST CHARACTER OF THE MESSAGE IS NL--SO THE CARRIAGE WILL BE
                                 2803 * RETURNED WITH A LINE FEED AT THE END OF THE PRINTING OF THE MESSAGE
                                 2804         ERRORMSG  X'8000000000',DEST=C'T1',                                  X
                                                    DATA=C'E ERROR IN PROCESSING HEADER '
                                 2816         ERRORMSG  X'4000000000',DEST=C'T1',                                  X
                                                    DATA=C'E INVALID ORIGIN IN HEADER '
```

Figure 160. A message Control Program for Communication Applications (Part 15 of 20)

```
LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                    ASM 0102 19.28 06/07/74

                                 2828          ERRORMSG  X'0200000000',DEST=C'T1',                                X
                                                     DATA=C'E INSUFFICIENT BUFFERS FOR INCOMING MESSAGE '
                                 2840          ERRORMSG  X'0100000000',DEST=C'T1',                                X
                                                     DATA=C'E MESSAGE TOO LONG '
                                 2852 *
                                 2853 * THE FOLLOWING ERROR MESSAGE SHOULD ONLY OCCUR WITH MAIN STORAGE
                                 2854 * QUEUEING WITH OR WITHOUT DISK BACKUP
                                 2855          ERRORMSG  X'0040000000',DEST=C'T1',                                X
                                                     DATA=C'E PERCENTAGE OF BUFFER UNITS IN BUFMAX ARE USED-SX
                                                     LOW DOWN '
                                 2867 *
                                 2868          ERRORMSG  X'0002000000',DEST=C'T1',                                X
                                                     DATA=C'E FORWARDED TO INVALID DESTINATION '
                                 2880          ERRORMSG  X'0000400000',DEST=C'T1',                                X
                                                     DATA=C'E INVALID STATION ID AT CONNECT TIME '
                                 2892          ERRORMSG  X'0000200000',DEST=C'T1',                                X
                                                     DATA=C'E TERMINAL IS IN HOLD STATUS '
                                 2904          ERRORMSG  X'0000080000',DEST=C'T1',                                X
                                                     DATA=C'E MSGTYPE CODE IN HEADER INVALID '
                                 2916          ERRORMSG  X'000000E000',DEST=C'T1',                                X
                                                     DATA=C'E A HARDWARE ERROR HAS OCCURRED '
                                 2928 *
                                 2929 * INEND IS REQUIRED AS LAST DELIMITER MACRO OF INCOMING GROUP
                                 2930          INEND
                                 2934 *
                                 2935 ***
                                 2936 *
                                 2937 * OUTGOING GROUP OF MESSAGE HANDLER FOR TWX TERMINAL
                                 2938          OUTHDR
                                 2944 *
                                 2945 * INSERT CR LF RUBOUT AT BEGINNING OF MESSAGE
                                 2946          MSGEDIT  ((I,XL3'261507',SCAN))
                                 2958 *
                                 2959 * THE FOLLOWING MACRO CAUSES EOT LINE CONTROL CHARACTERS TO BE INSERTED
                                 2960 * IN EACH OUTGOING MESSAGE
                                 2961          MSGFORM
                                 2968 *
                                 2969 * SET THE SCAN POINTER TO THE PERIOD IN THE HEADER AND INSERT DATE,
                                 2970 * TIME, AND SEQUENCE NO.--INSERTED IN EBCDIC SO DO BEFORE CODE
                                 2971          SETSCAN C'.'
                                 2984 *
                                 2985 * IF NO OPERAND--BOTH DATE AND TIME ARE INSERTED--SPACE MUST BE
                                 2986 * RESERVED BY MEANS OF THE RESERVE= OPERAND OF DCB FOR LINE--THE DATE
                                 2987 * IS IN FORM--(BLANK)YY.DDD--7 CHARS.--TIME IN FORM--
                                 2988 * (BLANK) HH.MM.SS--9 CHARACTERS
                                 2989          DATETIME
                                 3002 *
                                 3003 * SEQUENCE IN AN OUTHDR SUBGROUP INSERTS SEQUENCE NO. IN FORM--
                                 3004 * (BLANK)NNNN--5 CHARS.--SPACE MUST BE RESERVED BY MEANS OF RESERVE=
                                 3005 * OPERAND OF DCB FOR LINE
                                 3006          SEQUENCE
                                 3013 *
                                 3014 * LOG OUTGOING HEADERS--USE DCBNAME AS OPERAND--PUT MACRO AFTER
                                 3015 * INSERTION OF DATE, TIME, AND SEQUENCE NOS. SO THESE WILL APPEAR
                                 3016 * IN LOGGED HEADER
```

Figure 160. A Message Control Program for Communication Applications (Part 16 of 20)

```
 LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                ASM 0102 19.28 06/07/74
                                  3017 *                                                NEXT IWD DISABLED--*
                                  3018 *        LOG    MSGLOG
                                  3019 *
                                  3020 * THE MACROS IN THE FOLLOWING SUBGROUP ARE EXECUTED FOR EVERY BUFFER
                                  3021 * OF THE MESSAGE
                                  3022          OUTBUF
                                  3026 *
                                  3027 * INSERT CR LF RUBOUT FOR EVERY X'FF' CHAR. IN MESSAGE--X'FF' IS
                                  3028 * THE RECDEL CHAR. SPECIFIED IN THE TPROCESS MACROS
                                  3029          MSGEDIT  ((RA,XL3'261507',XL1'FF'))
                                  3047 *
                                  3048 * TRANSLATE THE MESSAGE FROM EBCDIC TO LINE CODE--IF ISSUED IN A
                                  3049 * SUBGROUP AND ANY SEGMENTS OF A MESSAGE PROCESSED BY THAT SUBGROUP,
                                  3050 * THE ENTIRE MESSAGE IS TRANSLATED
                                  3051          CODE
                                  3060 *
                                  3061 * THE OUTMSG SUBGROUP IS SPECIFIED AFTER OTHER SUBGROUPS IN OUTGOING
                                  3062 * GROUP--IT IS EXECUTED ONLY AFTER AN ENTIRE BLOCK OR MESSAGE HAS BEEN
                                  3063 * SENT
                                  3064          OUTMSG
                                  3073 *
                                  3074 * THE HOLD MACRO SUSPENDS TRANSMISSION TO A STATION EITHER FOR A TIME
                                  3075 * INTERVAL (IF SPECIFIED) OR UNTIL RELEASED BY AN OPERATOR CONTROL
                                  3076 * MESSAGE--IF NOT USED, MESSAGES THAT CANNOT BE TRANSMITTED ARE
                                  3077 * TREATED AS THOUGH THEY HAVE BEEN TRANSMITTED--ALSO, A HOLD OPERATOR
                                  3078 * CONTROL MESSAGE HAS NO EFFECT IF THERE IS NO HOLD MACRO--
                                  3079 * BITS BEING TESTED BY MASK ARE FOR HARDWARE ERRORS
                                  3080          HOLD  X'000000E000'
                                  3086 *
                                  3087 * IN THE FOLLOWING ERROR MESSAGES, THE 1ST FIELD IS THE MASK CORRE-
                                  3088 * SPONDING TO THE BITS IN THE ERROR RECORD,DEST= IS ALWAYS T1 FOR THE
                                  3089 * 1050 TERMINAL AND THE DATA= IS THE ERROR MESSAGE THAT IS SENT--
                                  3090 * THE MESSAGE INCLUDES THE HEADER OF THE MESSAGE IN ERROR AND THE
                                  3091 * ERROR MESSAGE
                                  3092 *
                                  3093 * THE LAST CHARACTER OF THE MESSAGE IS NL--SO THE CARRIAGE WILL BE
                                  3094 * RETURNED WITH A LINE FEED AT THE END OF THE PRINTING OF THE MESSAGE
                                  3095          ERRORMSG  X'8000000000',DEST=C'T1',                                  X
                                                      DATA=C'E ERROR IN PROCESSING HEADER '
                                  3107 *
                                  3108 * THE FOLLOWING ERROR MESSAGE SHOULD ONLY OCCUR WITH MAIN STORAGE
                                  3109 * QUEUEING WITH OR WITHOUT DISK BACKUP
                                  3110          ERRORMSG  X'0040000000',DEST=C'T1',                                  X
                                                      DATA=C'E PERCENTAGE OF BUFFER UNITS IN BUFMAX ARE USED-SX
                                                      LOW DOWN '
                                  3122 *
                                  3123          ERRORMSG  X'0000400000',DEST=C'T1',                                  X
                                                      DATA=C'E INVALID STATION ID AT CONNECT TIME '
                                  3135          ERRORMSG  X'0000200000',DEST=C'T1',                                  X
                                                      DATA=C'E TERMINAL IS IN HOLD STATUS '
                                  3147          ERRORMSG  X'000000E000',DEST=C'T1',                                  X
                                                      DATA=C'E A HARDWARE ERROR HAS OCCURRED '
                                  3159 *
                                  3160 * OUTEND REQUIRED AS LAST DELIMITER MACRO OF OUTGOING GROUP
                                  3161          OUTEND
```



```
 LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                ASM 0102 19.28 06/07/74
                                  3165 *
                                  3166 * A LTORG SHOULD BE CODED AFTER LAST DELIMITER OF EACH MH IF MCP HAS
                                  3167 * MORE THAN 1 MH
0010C0                            3168          LTORG
                                  3169 *
```

Figure 160. A Message Control Program for Communication Applications (Part 17 of 20)

```
LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                ASM 0102 19.28 06/07/74
                                 3171 ***
                                 3172 *
                                 3173 * MESSAGE HANDLER FOR INPUT FROM AND OUTPUT TO APPLICATION PROGRAM
                                 3174 * RUNNING WITH TERMINALS
                                 3175 *
                                 3176 * THE FOLLOWING MACRO IS REQUIRED AND MUST BE FIRST
                                 3177 *         LC= IS A REQUIRED OPERAND--PUT'IN'SINCE NO LINE CONTROL
                                 3178 *           CHARACTERS TO REMOVE
                                 3179 MHTRMAPP STARTMH  LC=IN
                                 3195 *
                                 3196 * THE INCOMING GROUP HANDLES MESSAGES COMING FROM AN APPLICATION
                                 3197 * PROGRAM--THE MESSAGES WILL SUBSEQUENTLY BE PROCESSED BY THE OUTGOING
                                 3198 * GROUP FOR THE DESTINATION TERMINAL
                                 3199 *
                                 3200 * THE INHDR DELIMITER IS REQUIRED AND IS ALWAYS 1ST MACRO
                              (38) 3201         INHDR
                                 3212 *
                                 3213 * LOG INCOMING HEADERS--USE DCBNAME AS OPERAND
                                 3214 *                                                NEXT IWD DISABLED--*
                                 3215 *         LOG   MSGLOG
                                 3216 *
                                 3217 * THE FORWARD MACRO IS REQUIRED IN EACH INHDR SUBGROUP--
                                 3218 * THE OPERAND DEST=PUT SAYS TO FORWARD TO THE DESTINATION SPECIFIED
                                 3219 * IN THE PREFIX TO THE APPLICATION PROGRAM WORK AREA
                                 3220         FORWARD  DEST=PUT
                                 3228 *
                                 3229 * THE INMSG SUBGROUP IS SPECIFIED AFTER OTHER SUBGROUPS IN AN INCOMING
                                 3230 * GROUP--IT IS EXECUTED AFTER AN ENTIRE MESSAGE OR BLOCK HAS BEEN
                                 3231 * PROCESSED
                                 3232         INMSG
                                 3240 *
                                 3241 * IN THE FOLLOWING ERROR MESSAGES, THE 1ST FIELD IS THE MASK CORRE-
                                 3242 * SPONDING TO THE BITS IN THE ERROR RECORD, DEST= IS ALWAYS T1 FOR THE
                                 3243 * 1050 TERMINAL AND THE DATA= IS THE ERROR MESSAGE THAT IS SENT--
                                 3244 *
                                 3245 * THE LAST CHARACTER OF THE MESSAGE IS NL--SO THE CARRIAGE WILL BE
                                 3246 * RETURNED WITH A LINE FEED AT THE END OF THE PRINTING OF THE MESSAGE
                                 3247         ERRORMSG  X'0200000000',DEST=C'T1',                                X
                                                DATA=C'E INSUFFICIENT BUFFERS FOR INCOMING MESSAGE '
                                 3259 *
                                 3260 * THE FOLLOWING ERROR MESSAGE SHOULD ONLY OCCUR WITH MAIN STORAGE
                                 3261 * QUEUEING WITH OR WITHOUT DISK BACKUP
                                 3262         ERRORMSG  X'0040000000',DEST=C'T1',                                X
                                                DATA=C'E PERCENTAGE OF BUFFER UNITS IN BUFMAX ARE USED-SX
                                                LOW DOWN '
                                 3274         ERRORMSG  X'0002000000',DEST=C'T1',                                X
                                                DATA=C'E FORWARDED TO INVALID DESTINATION '
                                 3286 *
                                 3287 * INEND IS REQUIRED AS LAST DELIMITER OF INCOMING GROUP
                                 3288         INEND
                                 3292 *
                                 3293 ***
                                 3294 *
                                 3295 * OUTGOING GROUP HANDLES MESSAGES BEING SENT TO APPLICATION PROGRAM
                              (37) 3296         OUTHDR
```

Figure 160. A Message Control Program for Communication Applications (Part 18 of 20)

```
LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                ASM 0102 19.28 06/07/74
                                 3302 *
                                 3303 * DELETE ANY CHARS. SUCH AS CR,LF WHICH APPEAR BEFORE $ IN HEADER
                                 3304         MSGEDIT  ((R,CONTRACT,SCAN,C'$'))
                                 3319 *
                                 3320 * SET SCAN POINTER OVER 2 NON-BLANK CHARS.--$ AND MSGTYPE FIELD--SO
                                 3321 * IT POINTS TO BEFORE SOURCE FIELD
                                 3322         SETSCAN  2
                                 3330 *
                                 3331 * INSERT SEQUENCE NO.--FORMAT IS (BLANK)NNNN--5 CHARS--SPACE MUST BE
                                 3332 * RESERVED BY MEANS OF RESERVE= OPERAND OF DCB FOR LINE
                                 3333         SEQUENCE
                                 3340 *
                                 3341 * LOG OUTGOING HEADERS--USE DCBNAME AS OPERAND--PUT MACRO AFTER
                                 3342 * INSERTION OF SEQUENCE NO. SO THIS WILL APPEAR IN LOGGED HEADER
                                 3343 *                                                NEXT IWD DISABLED--*
                                 3344 *         LOG   MSGLOG
                                 3345 *
                                 3346 * SET SCAN POINTER OVER 2 NON-BLANK CHARS. (SOURCE FIELD) SO IT POINTS
                                 3347 * TO EOF FIELD
                                 3348         SETSCAN  2
                                 3356 *
                                 3357 * SETEOF IS USED TO IDENTIFY THE LAST MESSAGE OF A GROUP OF MESSAGES
                                 3358 * TO THE APPLICATION PROGRAM--IT CAUSES THE NEXT READ/CHECK AFTER
                                 3359 * THIS COMPLETE MESSAGE HAS BEEN RECEIVED TO PASS TO AN APPLICATION
                                 3360 * PROGRAM EODAD ROUTINE--(THE COBOL PROGRAM WOULD RECEIVE AN ETI
                                 3361 * INDICATION)
                                 3362         SETEOF  C'F'
                                 3379 *
                                 3380 * NO OUTMSG SUBGROUP WILL BE EXECUTED FOR A MESSAGE BEING TRANSFERRED
                                 3381 * FROM A TPROCESS QUEUE TO AN APPLICATION PROGRAM--
                                 3382 * SO OMIT OUTMSG IN THIS MESSAGE HANDLER
                                 3383 *
                                 3384 *
                                 3385 * OUTEND IS REQUIRED AS LAST DELIMITER OF OUTGOING GROUP
                                 3386         OUTEND
                                 3397 *
                                 3398 * A LTORG SHOULD BE CODED AFTER LAST DELIMITER OF EACH MH IF MCP HAS
                                 3399 * MORE THAN 1 MH
0011C8                           3400         LTORG
                                 3401 *
```

Figure 160. A Message Control Program for Communication Applications (Part 19 of 20)

```
LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                ASM 0102 19.28 06/07/74

                                 3403 ***
                                 3404 *
                                 3405 * MESSAGE HANDLER FOR COBOL PROGRAMS THAT SIMULATE TERMINAL INPUT DATA
                                 3406 * --USED FOR TESTING WITHOUT TERMINALS
                                 3407 *
                                 3408 * THE FOLLOWING MACRO IS REQUIRED AND MUST BE FIRST
                                 3409 *        LC= IS A REQUIRED OPERAND--PUT'IN'SINCE NO LINE CONTROL
                                 3410 *          CHARACTERS TO REMOVE
                                 3411 MHAPPAPP STARTMH  LC=IN
                                 3427 * THE INCOMING GROUP HANDLES MESSAGES COMING FROM AN APPLICATION
                                 3428 * PROGRAM--THE MESSAGES WILL SUBSEQUENTLY BE PROCESSED BY THE OUTGOING
                                 3429 * GROUP WHEN THE APPLICATION PROGRAM READS THEM BACK
                                 3430 *
                                 3431 * THE INHDR DELIMITER IS REQUIRED AND IS ALWAYS 1ST MACRO
                                (40) 3432         INHDR
                                 3443 *
                                 3444 * THE FORWARD MACRO IS REQUIRED IN EACH INHDR SUBGROUP--
                                 3445 * THE OPERAND DEST=PUT SAYS TO FORWARD TO THE DESTINATION SPECIFIED
                                 3446 * IN THE PREFIX TO THE APPLICATION PROGRAM WORK AREA
                                 3447         FORWARD  DEST=PUT
                                 3455 *
                                 3456 * THE INMSG SUBGROUP IS SPECIFIED AFTER OTHER SUBGROUPS IN AN INCOMING
                                 3457 * GROUP--IT IS EXECUTED AFTER AN ENTIRE MESSAGE OR BLOCK HAS BEEN
                                 3458 * PROCESSED
                                 3459         INMSG
                                 3467 *
                                 3468 * IN THE FOLLOWING ERROR MESSAGES, THE 1ST FIELD IS THE MASK CORRE-
                                 3469 * SPONDING TO THE BITS IN THE ERROR RECORD, DEST= IS ALWAYS T1 FOR THE
                                 3470 * 1050 TERMINAL AND THE DATA= IS THE ERROR MESSAGE THAT IS SENT--
                                 3471 *
                                 3472 * THE LAST CHARACTER OF THE MESSAGE IS NL--SO THE CARRIAGE WILL BE
                                 3473 * RETURNED WITH A LINE FEED AT THE END OF THE PRINTING OF THE MESSAGE
                                 3474         ERRORMSG  X'0200000000',DEST=C'T1',                                X
                                                    DATA=C'E INSUFFICIENT BUFFERS FOR INCOMING MESSAGE '
                                 3486 *
                                 3487 * THE FOLLOWING ERROR MESSAGE SHOULD ONLY OCCUR WITH MAIN STORAGE
                                 3488 * QUEUEING WITH OR WITHOUT DISK BACKUP
                                 3489         ERRORMSG  X'0040000000',DEST=C'T1',                                X
                                                    DATA=C'E PERCENTAGE OF BUFFER UNITS IN BUFMAX ARE USED-SX
                                                    LOW DOWN '
                                 3501         ERRORMSG  X'0002000000',DEST=C'T1',                                X
                                                    DATA=C'E FORWARDED TO INVALID DESTINATION '
                                 3513 *
                                 3514 * INEND IS REQUIRED AS LAST DELIMITER OF INCOMING GROUP
                                 3515         INEND
                                 3519 *
                                 3520 ***
                                 3521 *
                                 3522 * OUTGOING GROUP HANDLES MESSAGES BEING SENT TO APPLICATION PROGRAM
                                 3523 *
                                 3524 * NO OUTMSG SUBGROUP WILL BE EXECUTED FOR A MESSAGE BEING TRANSFERRED
                                 3525 * FROM A TPROCESS QUEUE TO AN APPLICATION PROGRAM--
                                 3526 * SO OMIT OUTMSG IN THIS MESSAGE HANDLER
                                 3527 *
                                 3528 *
```



```
LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                ASM 0102 19.28 06/07/74

                                 3529 * OUTEND IS REQUIRED AS LAST DELIMITER OF OUTGOING GROUP
                                (39) 3530         OUTEND
                                 3541 *
                                 3542 * A LTORG SHOULD BE CODED AFTER LAST DELIMITER OF EACH MH IF MCP HAS
                                 3543 * MORE THAN 1 MH
001258                           3544         LTORG
                                 3545 *
                                 3546         END
```

Figure 160. A Message Control Program for Communication Applications (Part 20 of 20)

Defining the Buffers

User-defined areas of main storage receive any and all messages entering a TCAM network. Such areas, known as buffers, are used for handling, queueing, and transferring message segments between all lines and queueing media, and between queueing media and COBOL work areas.

In order to understand how the buffers are defined, it is necessary to distinguish between buffer units and buffers. TCAM has one buffer unit pool that contains buffer units of one size. Buffer units are the basic building blocks from which buffers are constructed (that is, buffer units are linked together to form buffers). Therefore, even though the buffers for line groups and for the application program may differ in size, each size specified should be a multiple of the size specified for a buffer unit, in order to use space optimally.

Three operands of the INTRO macro, (1), describe the TCAM buffer unit pool. As in the sample program shown in Figure 160, the operands that define the size of buffer units and specify the number assigned are KEYLEN, LNUNITS, and MSUNITS. The operands BUFSIZE, BUFIN, and BUFOUT, given in the DCB for line groups, (7c), and in the PCB, (6), for an application program, specify the buffer size and the number of buffers to be assigned initially for a receiving or sending operation. The manner in which the PCI= operand of the DCB for a line group, (7c), is coded greatly affects the number coded for LNUNITS in the INTRO macro and the numbers coded for the BUFIN and BUFOUT operands of the DCB for the line group.

Activating and Deactivating the Message Control Program

The TCAM message control program is assembled, link-edited, and executed like any other program running under an OS/VS system. The macros INTRO, OPEN, and READY, issued as a group, make up the data-set initialization and activation section of the message control program.

Orderly deactivation of the TCAM system must stop incoming and outgoing message traffic and create a checkpoint record. The user must ensure that the data sets for any application program using TCAM as its access method are closed before the MCP enters its deactivation section, which closes the MCP data sets. (It is suggested that the headers of messages transmitted to the COBOL programs contain a code that signals the COBOL program to go to the STOPRUN statement.) Finally, the MCP coding must return control to the OS/VS supervisor.

INTRO Macro: As the first macro executed in the message control program, INTRO, (1), establishes standard entry linkage, chains save areas, provides addressability, and saves the start parameter list pointer. (A description of the operands in the INTRO macro precedes the macro itself in the sample program.)

Note: The message below is issued if at least one of the following operands is omitted from the INTRO macro: STARTUP=, KEYLEN=, LNUNITS= , and (if DISK=YES is coded in the INTRO macro) CPB=.

```
00 IED002A SPECIFY TCAM PARAMETERS
```

The user may then enter the additional required parameters, changing certain other operands as desired.

OPEN Macro: The OPEN macros, (2), complete the initialization of the TCAM data sets and activate them for use. The TCAM data sets that must be activated in the MCP by OPEN macros are those for the message queues, (a), checkpoint, optionally (b), the line groups, (c), and the message log, optionally (d). If a snap dump is used, the user must also open the data set for snap, (e).

READY Macro: The READY macro, (3), must be the last instruction in the initialization and activation section of the MCP. When READY has executed, the system is ready to handle message traffic.

CLOSE Macro: An optional snap dump of the program begins the deactivation section. Then the first CLOSE macro instruction, (4), is executed. This deactivation section is not executed until all data sets in TCAM application programs have been closed. In the example, the user closes the line group data sets, (a), first; next the snap data set, (b); then the message log data set, (c); the checkpoint data set next to last, (d); and finally the message queues data set, (e).

Note: The data sets may be closed in any order provided that the checkpoint data set and the message queues data set are closed in the order indicated.

RETURN Macro: The assembler-language Load instruction is issued to restore register 13 with the address of the system save area, and the RETURN macro, (5), is issued to return control to the OS/VS supervisor.

Defining the MCP Data Sets and Process Control Blocks

The user must provide information that serves as an interface between the message control program and the application program. This information is contained in process control blocks (PCB) and is generated by the PCB macro.

The message control program must also describe the MCP data sets to be used. Two of the four possible types of data sets usually required by every message control program are the line group data set, if there are lines, and the message queues data set, if there are disk queues. The operation of the MCP requires that either a message queues data set or a line group data set be opened. A user employing main storage queueing for application-to-application program processing (who, therefore, does not need either of these data sets) must, nevertheless, open a dummy line to meet this requirement. An error message will be issued at the system console because no hardware is attached, but this message can be ignored.

If the user does not open a line and, therefore, does not need either a DCB for a line group or a TERMINAL entry, the assembly of the MCP, nevertheless, generates an error message for the undefined symbol of IEDQSTCS. The user can either define this symbol in this program with a dummy label or ignore the severity level of (8) in the link-edit step. The symbol IEDQSTCS need not be correctly defined when the user is running only application-to-application programs.

Either or both of the other two types of data sets -- the checkpoint data set and the log data set -- may be specified if needed. To describe data sets to the system, the user (via a DCB macro) defines a data control block (DCB) for each data set cited.

PCB Macro: A process control block, created through specification of the PCB macro (6), is required in the MCP for each active application program. The PCB macro is similar to the DCB for the line groups in that it specifies the name of the message handler to be used for messages being sent by or received from an application program, as well as buffer requirements. The TPROCESS macro (see the discussion under "Defining Terminal and Line Control Areas") refers to the name of the PCB macro.

In the sample program given in Figure 160 are three process control blocks -- PCBLK, (a), for a COBOL program running with terminals; PCBLK1, (b), for COBOL programs that simulate the sending of messages from a remote terminal; and PCBLK2, (c), for testing COBOL programs that take advantage of the queue structure feature. Having these three control blocks makes it possible for the COBOL program running with terminals to run at the same time as one of the other COBOL programs. In the example in this chapter, the TESTTP1 program simulates a terminal sending messages to the TESTTP2 program.

DCB Macro: A data control block, created through specification of the DCB macro, (7), is required for each data set referred to by the MCP. In the sample message control program, data control blocks are defined as follows:

- The message queues DCB macro, which defines a data control block for a message queues data set, MSGQ (a).
- The checkpoint DCB macro, which defines a checkpoint data set if the checkpoint facility is to be used, CHKPT (b).
- The line group DCB macro, which defines a line group data set, must be specified for each line group in the system. In the sample MCP, two line group data sets are defined -- the 1050 line group, named LN1050 (c), and the TWX line group, named LNTWX (d).
- The log DCB marco, which defines data sets for messages or message segments, should be specified for each secondary storage device on which messages or message segments may be logged. In the sample program, only one log DCB defining the MSGLOG data set, (e), is specified.
- The snap dump DCB macro, which defines the data set for a snap dump, should be specified only if the user wants a snap dump. In the sample program, the DUMP data set is defined, (f).

Defining Terminal and Line Control Areas

In writing an MCP, the user must provide information that identifies the remote stations, specifies their characteristics to the system, and tells how they are to be handled. Line control is the scheme of operating procedures and signals by which a teleprocessing system is controlled.

Line control concerns itself with such tasks as establishing contact between a sending and a receiving station, directing a message to a specific station on a

multistation line, handling priorities when two stations try to send at the same time, and performing a user-specified action when a station fails to respond to a message.

Several TCAM macros are available to the user for identifying stations and specifying how message transmission is to be handled. The TCAM macros used in the sample message control program given in Figure 148 -- TTABLE, TERMINAL, INVLIST, TLIST, and TPROCESS -- are described below. Two additional macros -- OPTION, which reserves space for an option field, and LOGTYPE, needed only for logging entire messages -- are also available to the COBOL user.

TTABLE Macro: The TTABLE macro, (8), defines the start and the end of the terminal table, needed to provide information about each station and application program.

TERMINAL Macro: The TERMINAL macro, (9), specified three times in the sample program, must be coded for each station that can accept messages (as well as for some terminals that can only enter messages), each group of non-switched terminals equipped with the hardware group-code feature, and each switched line to stations that do not uniquely identify themselves after calling the computer.

Specification of the TERMINAL macro places a station or line name and associated information in this terminal table. TERMINAL produces a single entry, a group entry, or a line entry. In the example, the T1 entry, (a), provides information about the 1050 terminal, the T2A entry, (b), information about the switched TWX line, and the T2 entry, (c), information about the TWX terminal on this line.

Notes:

1. The "UTERM=YES" specification in the TERMINAL macro for the switched TWX line creates an entry for the line. This gives the program the control information it needs to handle stations that call this line. After the station is identified by means of the ORIGIN macro in the MH, the program then refers to the TERMINAL entry for the station.

2. All TERMINAL macros for lines in a line group must be arranged in ascending relative line numbers. The TERMINAL macro for a particular line must immediately precede all TERMINAL macros for stations on that line. In the sample MCP, there is only one line per line group and one terminal per line, but this need not be true.

TPROCESS Macro: By placing the name of a queue for an application program, as well as associated information, in the terminal table, the TPROCESS macro, (10), helps connect a COBOL program with the message control program.

The user must specify one TPROCESS macro for each destination queue from which a COBOL program is to receive messages and at least one that is used when messages are sent by a COBOL program. (That is, one output TPROCESS entry is required for each application program running simultaneously.) The output TPROCESS entry is not the name of a queue. In the sample program, for example, twelve TPROCESS entries are specified. The PIN entry, (a), identifies an input destination queue for a COBOL program running with terminals; POUT identifies an output process entry.

Similarly, the P1, (c), and P2, (d), entries identify input destination queues for COBOL programs that simulate terminal input data, and the POUT1, (e), entry identifies an output process entry for such COBOL programs. The PQ1, PQ2, PQ3, PQ4, PQ5, PQ6, and PQOUT TPROCESS entries are used for COBOL programs that employ the queue structure feature.

Note: Because the PIN and POUT entries in the example refer to one process control block (PCBBLK) and the P1, P2, and POUT1 entries refer to another process control block (PCBBLK1), a program running with terminals can run concurrently with another program. This is also true of the PQ entries, which refer to PCBLK2.

TLIST Macro: An instruction that places the name of a list of a single, a group, or a process entry in the terminal table, the TLIST macro, (11), must be specified for each such list to be created. This list can be specified as either a distribution list or a cascade list. When a message is sent to a distribution list, the same message is sent to all locations on the list. When a message is sent to a cascade list, the message is transmitted to the listed destination with the fewest messages enqueued. In the sample message control program, the TLIST entry D1, (a), represents a distribution list entry. The list should not include a TPROCESS entry for a COBOL application program.

INVLIST Macro: An instruction that creates an invitation list entry containing the invitation characters for the stations on the line (in the order in which they are to be invited to send messages), the INVLIST macro, (12), must be issued for each line

in the system. However, one INVLIST macro suffices for all output-only lines to stations that do not use invitation sequences. Two INVLIST entries -- LIST1050, (a) , and LISTTWX, (b) -- appear in the sample program.

Note: Either a parameter of + in the INVLIST macro or an operator control command (see the section "Using TCAM Service Facilities" in this chapter) must initially activate a station for entering messages.

In the entry LIST1050, for example, 'T1 + 6215' indicates that the IBM 1050 terminal identified as T1 is active for entering messages. (6215 is the IBM 1050 transmission code representation of the polling characters A0 in hexadecimal notation.) Accordingly, the symbol 'T2A+' in the LISTTWX entry indicates an initially active line. (Note: The terminal name for the line, not the station, must be used.) For a TWX station, the '+' character would be followed by an ID sequence instead of the polling character used in the LIST1050 example. In the example, no ID sequence is given. The (CPUID) = operand in the INVLIST macro for the TWX terminal is required.

Designing the Message Handler

The major section in a message control program is the group of message handlers (MH), made up of sets of routines that examine and process control information in message headers (see Figure 149) and perform the functions necessary in preparing message segments for forwarding to their destinations. There is usually a message handler for each line group or active application program. Each message handler usually contains both an incoming and an outgoing group.

A message may consist of two parts -- the header, or control, portion and the text portion -- depending on the application. The sample message control program shown in Figure 160 contains four message handlers, as listed below. Three of these message handlers are based on a message header containing the information described in the comments that immediately precede the first sample message handler, (13) . The fourth message handler in the sample MCP, MHAPPAPP, handles messages with no headers.

- A message handler (MH1050) for input from and output to the IBM 1050 Data Communications System Terminal.
- A message handler (MHTWX) for input from and output to the Teletypewriter Exchange (TWX).
- A message handler (MHTRMAPP) for input from and output to an application program running with terminals.
- A message handler (MHAPPAPP) for input from and output to an application program that simulates terminal input data. This type of message handler can be used for testing without terminals or for handling messages sent from one application program to another, as in the sample COBOL programs TESTTP1 (see Figure 163) and TESTTP2 (see Figure 164).

Two kinds of macro instructions that may be included in a message handler are functional macros and delimiter macros. Functional macros perform the specific operations necessary for messages directed to the message handler. Delimiter macros classify and identify sequences of functional macro instructions and then direct control to the appropriate sequence. Figure 149 shows some of the functional macros that can be used with the delimiter macros in the incoming group and the outgoing group of the message handler. All of these macros are included in the sample message handler in Figure 160.

To decide which macro to place in which group, the user must understand which group is executed when. This is discussed in the description associated with Figure 159. The steps executed by a message handler are shown at the right-hand side of this figure. When messages are received from stations, the incoming group of a message handler for the line is executed before the outgoing group. However, when messages are sent to application programs, the outgoing group of the message handler for the application program is executed first. The decision boxes shown in Figure 159 are determined by the destination specified in the required FORWARD macro of a message handler (that is, if the destination is the name of a TPROCESS entry, processing is required in an application program; if, however, the destination is the name of a TERMINAL macro, no more processing is required).

Note: For descriptions of other macros that can be coded in an MCP, see the publication OS/VS TCAM Programmer's Guide.

Groups	Subgroups	Delimiter Macros	Functional Macros
		STARTMH*	
			CODE
			LOG
	Inheader		SETSCAN
Incoming	Subgroup	INHDR*	MSGTYPE
Group			ORIGIN
			FORWARD
			TERRSET
	Inbuffer		CUTOFF
	Subgroup	INBUF	MSGEDIT
	Inmessage	INMSG	CANCELMG
	Subgroup		ERRORMSG
		INEND*	
			MSGFORM
			MSGTYPE
			MSGEDIT
	Outheader		SETSCAN
	Subgroup	OUTHDR	DATETIME
			SEQUENCE
Outgoing			LOG
Group			SETEOF
	Outbuffer	OUTBUF	MSGEDIT
	Subgroup		CODE
	Outmessage	OUTMSG	HOLD
	Subgroup		ERRORMSG
		OUTEND*	

*The STARTMH macro is always required. If the message handler is to handle incoming messages, the INHDR, INEND, and OUTEND macros are also required. If the message handler is to handle outgoing messages, the OUTEND macro is required.

Figure 161. Macros that can be coded in a Message Handler

A discussion of sample message handlers for terminal line groups appears below. For discussions of the MHTRMAPP and MHAPPAPP message handlers, see the sections "A Message Handler for an Application Program Running with Terminals" and "A Message Handler for an Application Program that Simulates Input Data."

A MESSAGE HANDLER FOR THE TERMINAL LINE GROUPS: Because the message handlers for the 1050 line and the TWX line are similar (except for the difference in line control characters and the use of the 1050 for error messages), the description of the message handler for the 1050 (MH1050) given below should also suffice for the TWX line group (MHTWX).

The Incoming Group: The first macro in the MH1050 message handler is STARTMH, (13), in which the LC=OUT operand specifies that line control characters are to be removed. The first macro in the INHDR, (14), subgroup (CODE), (15), translates the incoming messages to EBCDIC. Then the LOG macro, (16), records the header on the log data set. Even though the CODE macro is part of the INHDR subgroup, all buffers of the message are translated from line code to EBCDIC -- not just the first (header) buffer. In the normal case, unless the line code is EBCDIC, the CODE macro should be placed first, as in this example. A CODE macro must be issued before an ORIGIN macro, since the name in the header is checked against the terminal names, which are in EBCDIC. The name in the header, therefore, cannot be located unless it has first been translated. The same translation requirements apply to such macros as SETSCAN, (17), in the example. In this case, if the C'$' in the message were not first translated to EBCDIC, the C'$' would have to be specified in line code.

The SETSCAN macro, (17), sets the scan pointer to "$" in the header, and the MSGTYPE macros, (18), that follow check the character in the next field (with fields separated by at least one blank character) for one of the four codes that represent possible message destinations. If the scan yields a match between a field in the incoming message and the code for one of the MSGTYPE macros, the macros between this MSGTYPE macro and the next MSGTYPE macro are executed. Control is then given to the next subgroup (INBUF), (22). When a MSGTYPE match is found, the ORIGIN macro, (19), is issued. The FORWARD macro, (20), which is always required, transmits the message to the destination specified.

If there is no match with any of the operands specified in the MSGTYPE macros, the last MSGTYPE macro, which has a blank operand field, is executed. The required FORWARD macro follows, and the TERRSET macro, (21), sets the user error bits in the error record for the message.

In the INBUF subgroup, (22), the CUTOFF macro, (23), limits the size of the incoming messages and checks for station malfunction. The insertion of the RECDEL character by the MSGEDIT macro, (24) allows for record delimiters in the message, needed when the COBOL program reads in segment mode. The INMSG subgroup, (25), checks the error bits in the error record for this message and either cancels the message via the CANCELMG macro, (26), and/or sends an error message to the 1050 terminal using the ERRORMSG macro, (27).

The INEND macro, (28) , a required delimiter macro, signifies the end of the incoming groups.

The Outgoing Group: The macros discussed below, known as the outgoing group, are executed when messages are transmitted to the 1050 terminal. In the OUTHDR subgroup, (29) , the MSGFORM macro, (30) , causes line control characters to be inserted in the outgoing message. (Unless the user provides line control characters himself, this macro must be coded.) The MSGTYPE macro determines the type of message, so that a message can be processed either as an ordinary message or as an error message.

For every error message, the SETSCAN macro returns the scan pointer to the beginning of the message, and the MSGEDIT macro inserts the "NL" character before the message text. Processing of error messages resumes in the OUTBUF subgroup, (33) , of the message handler.

For the non-error messages, the MSGEDIT macro also inserts "NL" at the beginning of the message. Then the SETSCAN macro sets the scan pointer to the period at the end of the message header so that pertinent information can be inserted there. The DATETIME macro, (31) , records in the message being sent the date and time this macro is executed. The SEQUENCE macro, (32) , inserts a sequence number, and the LOG macro records the control information contained in the message header.

In the OUTBUF subgroup, (33) , of this message handler, the MSGEDIT macro inserts an "NL" character for every record delimiter character in the message. Because in the incoming group the RECDEL character is inserted for every "NL" and "LF" character, for a message that is simply transmitted from one terminal to another the message handler appears to send the same line control characters it receives. For a message sent by a COBOL program, on the other hand, wether or not record delimiter characters remain depends on the mode specified in the RECEIVE or SEND statement. (That is, when the programmer receives a message in segment mode, the record delimiter character is removed; when the programmer receives a message in message mode, the record delimiter is not removed. Accordingly, when the programmer sends a message in segment mode, the record delimiter character is added; when the programmer sends a message in message mode, the record delimiter is not added.) The next MSGEDIT macro inserts 13 idle characters after every "NL" character placed in the message, to allow the terminal sufficient time to return its carriage before receiving the next line. Finally, the CODE macro translates from EBCDIC to line code when no more handling is required with macros that operate in EBCDIC.

Like the INMSG subgroup (see "The Incoming Group"), the OUTMSG subgroup, (34) , checks the error bit in the error record for the message and transmits error messages, if any, to the 1050 terminal. The HOLD macro, (35) , is invoked only if there are hardware errors. Accordingly, a terminal placed in HOLD status is not released until an operator control message is issued. The OUTEND macro, (36) , signifies the end of the outgoing group.

A MESSAGE HANDLER FOR AN APPLICATION PROGRAM RUNNING WITH TERMINALS: The MHTRMAPP message handler handles messages transmitted by a terminal for the application program that is sending and receiving messages from terminals. Like the message handler discussed earlier, MHTRMAPP includes both an incoming group and an outgoing group.

In this message handler, because messages are sent to the application program from a terminal, the outgoing group headed by the OUTHDR macro, (37) , is executed first. The first macro (MSGEDIT) deletes any characters (for example, "NL", "CR", or "LF") that have preceded "$" in the header. This step is necessary because of the application program's expectation of receiving a fixed-length header beginning with "$". The next macro (SETSCAN) sets the scan pointer over the "$" and the MSGTYPE field. Then the SEQUENCE macro numbers the messages sent to the application program, and the LOG macro records the information contained in the message header.

The next SETSCAN macro sets the scan pointer over the source field in the header so that it points instead to the EOF field. The SETEOF macro identifies the last message in a data file being processed by an application program. If the character specified at the location pointed to by the scan pointer (and given as an operand in the SETEOF macro) is "F", the first RECEIVE statement issued by the COBOL program after receipt of the message causes the MCP to enter an application program EODAD routine. As far as the COBOL user is concerned, this section sets the "ETI" indicator in the field referred to by the END KEY clause in the input communication description (CD). The OUTMSG subgroup is not included in this message handler because it is not executed for messages sent to an application program. Nevertheless, the OUTEND delimiter macro signifies the end of the outgoing group.

The macros in the incoming group of this message handler, headed by the INHDR macro, (38) , are executed when messages are received from the COBOL program. The LOG macro records the information contained in the header, and the FORWARD macro, which is always required, specifies "DEST=PUT" as the message destination. This will cause the message to be forwarded to the destination the COBOL program has indicated in the output CD. The INMSG subgroup that follows checks to see whether sufficient buffer units are available for the message and verifies that the destination specified is valid. The INEND delimiter macro then specifies the end of the incoming group.

A MESSAGE HANDLER FOR AN APPLICATION PROGRAM THAT SIMULATES TERMINAL INPUT DATA: The MHAPPAPP message handler is for messages having no header. As a result, the only macro in the outgoing group is the delimiter macro OUTEND, (39) , which is always required.

The incoming group contains both the INHDR, (40) , subgroup, containing the required FORWARD macro, and the INMSG subgroup, which checks for availability of sufficient buffer units and verifies that the destination specified is valid. The required INEND delimiter macro is present.

ANS STANDARD MCP REQUIREMENTS

If the MCP to be written is to conform with the 1974 ANS COBOL standard, using ENABLE/DISABLE, SEND ADVANCING, SEND to multiple destination, and automatic communications job scheduling, then the sample MCP shown in Figure 160 must be modified as follows.

ENABLE/DISABLE: Operator Command Interface

The ENABLE/DISABLE statements are implemented by COBOL by sending various TCAM commands to the MCP, where they are acted upon by a combination of TCAM and COBOL-provided code.

Operator command communication with TCAM is accomplished through two special TPROCESS entries and an associated PCB defined in the MCP for each COBOL application program that will execute ENABLE/DISABLE statements. These entries are accessed by the COBOL library via the special DD names, COBOPIN and COBOPOUT. COBOPOUT will be used to issue operator commands to TCAM while COBOPIN will be used to receive the response messages.

The TPROCESS entry corresponding to COBOPOUT must be identified as a secondary operator control station (TPROCESS macro option, SECTERM=YES) and must indicate the TPROCESS entry associated with COBOPIN as the destination for operator command response messages (TPROCESS macro option, ALTDEST=tprocessname). The message handler for the TPROCESS entry associated with COBOPOUT must include a TCAM CODE macro followed by a TCAM FORWARD macro specifying the TPROCESS entry associated with COBOPIN in its destination list (to provide a response to commands for which the operator control character string is invalid--i.e., not caught by the CODE macro).

The operator commands used to effect ENABLE/DISABLE INPUT (with or without terminal) require the job name of the TCAM MCP (i.e., the name specified on the JCL JOB card for the MCP). The COBOL object time library contains an eight-byte control section, ILBOMCPN, that provides the value for the MCP job name. The default value of the job name is "TCAM". If a different name must be used for the MCP, ILBOMCPN must be replaced with a control section containing the correct value. Figure 162 illustrates a sample procedure for replacing the MCP job name CSECT.

```
FIXMCPN         JOB  ...
//              EXEC ASMFCL,PARM.ASM='NODECK,OBJ',
//                   PARM.LKED='LIST,LET,XREF,NCAL,RENT'
//ASM.SYSIN     DD   *
ILBOMCPN        CSECT
                DC   CL8'mcpname'  MCP JOB NAME
                END
/*
//LKED.SYSLMOD DD   DSN=SYS1.COBLIB,DISP=OLD
//LKED.SYSIN   DD   *
    INCLUDE  SYSLMOD(ILBONBL)
    ALIAS  ILBONBL0
    NAME  ILBONBL(R)
/*
```

Figure 162. Replacing the MCP Jobname CSECT

ENABLE/DISABLE--KEY Phrase

The CONTROL option of the TCAM MCP INTRO macro must specify a value for the operator control string of from one to eight characters in length. Optionally, the MCP Terminal Table may contain the following definition:

```
COBOPT0 OPTION CLn
```

where n may be in the range of one to ten but must be at least as large as the length of the operator control string specified in the INTRO macro.

Any Terminal Table entry corresponding to a symbolic input queue, symbolic source, or symbolic destination that may be enabled or disabled may have a value specified for the COBOPT0 option. The value must begin with the operator control string and may be followed by any characters valid in an option field. (Note that the value will be padded with spaces on the right if less than n characters are specified.)

The value of the identifier or literal associated with the KEY phrase (the key) may be from one to ten characters in length. The first eight characters of the key are used to form a TCAM operator control character string; however, if the first eight characters of the key contain a slash ("/") character, only that portion of the key preceding the first slash will be used.

The value of the operator control string must be identical to the value specified for the CONTROL option of the INTRO macro. If the values match and if the COBOPT0 option field has been specified for the MCP Terminal Table entry corresponding to the queue, source, or destination to be enabled or disabled, the full value of the key is compared to the value of the COBOPT0 option.

The key is considered invalid if it does not constitute a valid operator control string or does not match the COBOPT0 field, if specified.

COBOL provides two levels of password protection:

1. The first eight characters of any key translate into a TCAM operator control string. The control string may be considered a "global" password.
2. The full key value is compared with the value of the COBOPT0 option field. It may be considered a "local" password.

Note that by omitting the COBOPT0 option field from a Terminal Table entry definition, the entry becomes accessable to any key translating into a valid operator control string (i.e., only "global" protection is provided).

Since more than one terminal table entry may be affected by a single ENABLE/DISABLE statement containing a single key value, all such entries must be accessable via the same key.

The key is validated for all input queues and sources or all output destinations prior to any actual enabling or disabling.

ENABLE/DISABLE INPUT TERMINAL

If this form of ENABLE/DISABLE is to be executed by COBOL application programs, the TCAM MCP Terminal Table must contain the following option field definition:

```
COBOPT1 OPTION FL1
```

The Terminal Table entry corresponding to each symbolic source that may be enabled or disabled must have one of the following values specified for the COBOPT1 option field:

0: the symbolic source is initially enabled

1: the symbolic source is initially disabled

In addition, the following COBOL macro must be coded preceding the first INMSG subgroup of any MCP message handler that processes messages from a symbolic source that may be disabled:

```
[symbol] ILBONBLT
```

Expansion of the macro generates an INMSG subgroup that is executed only if the COBOPT1 option field of the Terminal Table entry corresponding to the symbolic source is equal to one. When executed, the subgroup cancels the message from the disabled source. The ILBONBLT macro may be followed by any TCAM macro valid in an INMSG subgroup, except CANCELMG (e.g., to issue an error message indicating that a message from a disabled terminal has been cancelled).

If the COBOPT1 option field is not specified for the Terminal Table entry corresponding to a symbolic source that is to be enabled or disabled, the entry is considered to be improperly defined and the symbolic source is consequently unknown (CD STATUS-KEY = '20').

ENABLE/DISABLE INPUT (without TERMINAL)

If this form of ENABLE/DISABLE is to be executed by COBOL application programs, the TCAM MCP Terminal Table must contain the following option field definition:

```
COBOPT2 OPTION FL1
```

The Terminal Table entry corresponding to each symbolic input queue that may be enabled or disabled must have one of the following values specified for the COBOPT2 option field:

0: the symbolic input queue is initially enabled

1: the symbolic input queue is initially disabled

In addition, the following COBOL macro must be included in the initialization section of the TCAM MCP (i.e., between the INTRO and READY macros):

```
[symbol] ILBONBLQ  [TPRFD={YES|NO}]
                  ,[TLCBD={YES|NO}]
                  ,[TAVTD={YES|NO}]
                  ,[REGS={YES|NO}]
```

The parameters TPRFD, TLCBD, and TAVTD refers to the TCAM descriptive macros of the same name for the Buffer Prefix, Line Control Block, and Address Vector Table DSECT's, respectively. If YES is coded for these parameters, the macro expansion will contain those portions of the corresponding DSECT's required by the ILBONBLQ macro. NO should be coded if the DSECT's are generated elsewhere in the MCP.

If REGS=YES is coded, general register usage within the macro expansion will appear symbolically as R0 through R15 (this will cause all register usage to appear in the cross reference listing at the end of the MCP assembly listing). Assembler statements equating these values to actual register numbers must be provided elsewhere in the MCP.

The ILBONBLQ macro intercepts control during TCAM message enqueuing and suppresses enqueuing for disabled symbolic input queues.

If the COBOPT2 option field is not specified for the Terminal Table entry corresponding to a symbolic input queue that is to be enabled or disabled, the entry is considered to be improperly defined and the symbolic input queue is consequently unknown (CD STATUS-KEY = '20').

ENABLE/DISABLE OUTPUT

If this form of ENABLE/DISABLE is to be executed by COBOL application programs, the TCAM MCP must contain a HOLD macro. If the HOLD macro is not required by the logic of the MCP, a HOLD macro must be coded specifying an impossible combination of errors in the mask associated with the message error record (see the description of the Hold/Release Facility in the OS/VS TCAM Programmer's Guide).

The Terminal Table entry corresponding to a symbolic destination that may be disabled must not specify main-storage-only queuing. The line corresponding to the symbolic destination must be open at the time the DISABLE OUTPUT statement is executed and may not be opened idle. If

these conditions are not met, the symbolic destination is considered to be improperly defined for the operation and, hence, unknown (CD STATUS-KEY = '20').

Specifying Characteristics for Symbolic Destinations

The following characteristics are associated with each symbolic destination:

- Whether or not vertical positioning is supported.
- For devices supporting vertical positioning, whether or not page psoitioning (Forms Feed) is supported.
- Whether or not the device has a fixed line size.
- For devices with a fixed line size, the size of the line.
- Whether or not record delimiter characters are to be inserted between message segments (for this characteristic to be effective, the RECDEL option of the TPROCESS entry corresponding to the COBTPOUT DD card must specify a non-zero value).
- Whether or not a test should be made for a disabled destination condition during the execution of each SEND statement (CD STATUS-KEY = '10'). Specification of this characteristic will incur an execution-time performance degradation and should be used only when the knowledge that the destination is disabled is significant to COBOL application programs sending messages to it.

The default characteristics for any symbolic destination are:

- no vertical positioning
- no fixed line size
- insert record delimiters between message segments
- no test for disabled destination

If characteristics are to be specified for symbolic destinations, the TCAM MCP Terminal Table must contain the following option field definition:

```
COBOPT3 OPTION XL11
```

Terminal Table entries corresponding to symbolic destinations for which characteristics are to be specified must contain a value for the COBOPT3 option according to the following (note that the values are given in hexadecimal notation to agree with the definition of COBOPT3):

Bytes 1-8
: X'C3D6C7D6D7E3F373' (This is the hex value for 'COBOPT3=' which makes this option self-identifying.)

Byte 9
: Destination characteristics byte:

  Bit 0 (X'80')
  : If on, vertical positioning is supported for this destination. When the ADVANCING clause is specified or implied, NL (new line), CR (arrriage return) and/or FF (forms feed) control characters will be inserted into the message by COBOL. If off, the ADVANCING clause will be ignored by COBOL.

  Bit 1 (X'40')
  : (Meaningful only if bit 0 is on) If bit 1 is on, FF (forms feed) will be inserted for an ADVANCING PAGE request. If off, NL (new line) will be inserted for an ADVANCING PAGE request.

  Bit 2 (X'20')
  : If on, this device has a fixed line size (as specified in bytes 10-11); COBOL will do automatic line folding. If off, no automatic line folding will occur.

  Bit 3(X'10')
  : If on, a record delimiter character (if available) will be inserted by COBOL between message segments.

  Bit 4 (X'08')
  : If on, a test will be made to determine if this destination is disabled during te execution of each SEND statement. If off, no test will be made.

  Bits 5-7 (X'07')
  : (Reserved)

Bytes 10-11
: The line width, for fixed line size destinations (e.g., if the line width is to be 96, code (x'0060'). For variable line size destinations, code X'0000'.

To conform with the requirements of ANS 1974 COBOL, the following is required:

- bit 0 must be on for destinations which support vertical positioning.

- bit 1 must be on for destinations which support forms feed.
- bit 2 must be on, and bytes 10-11 must be non-zero, for destinations which have fixed line sizes.
- bit 3 must be off.
- bit 4 must be on.

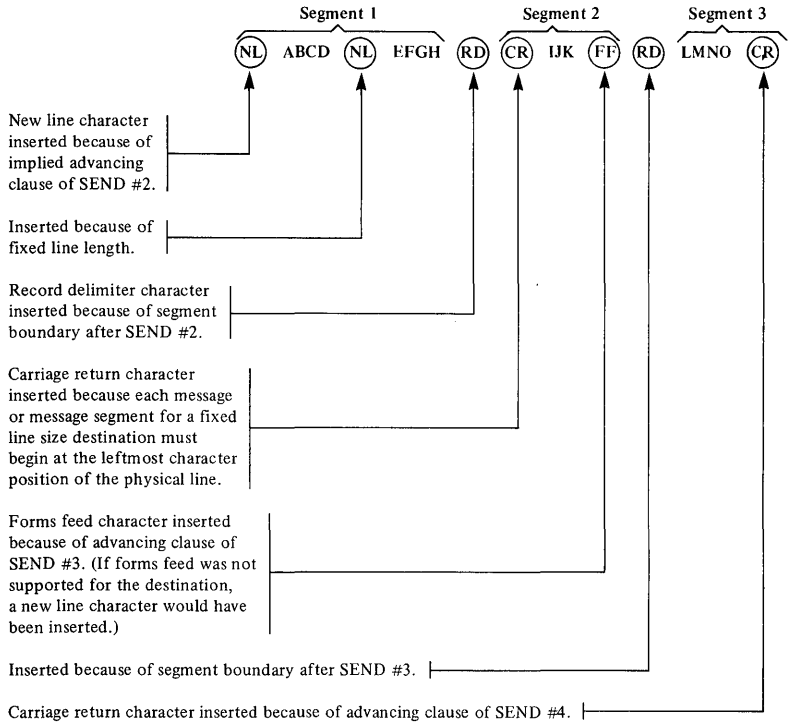
If vertical positioning has been specified for a destination (COBOPT3 byte 9, bit 0 on), the ADVANCING clause of the SEND statement causes COBOL to produce standard formatted messages containing text data, record delimiters, and the three EBCDIC/SNA control characters NL (new line), CR (carriage return), and FF (forms feed). Any specific device-dependent formatting required (e.g., insertion of idle characters) must be provided in the appropriate TCAM MCP message handlers.

When a SEND statement is executed that has an associated end of message (EMI) or end of group (EGI) indicator, the assembled messages for the destinations identified in the CD DESTINATION-TABLE are sent to TCAM. The contents of the messages will be a composite of text data and control characters (NL, CR, FF, and record delimiters) specified by information provided in the SEND statement ADVANCING clauses associated with the messages and the value of the COBOPT3 option fields for the destinations. Figure 163 illustrates the results of executing a sequence of SEND statements for a destination supporting both vertical positioning and a fixed line size.

Communications Job Scheduling (CJS)

It is sometimes desirable to schedule a COBOL object communications program only when there is work available for it to do (for example, to process messages which are entered infrequently or at unpredictable times). For this reason, a utility Communications Job Scheduler (CJS) is provided. This utility will monitor each of a set of user-specified TCAM queues, and schedule a user-specified job (using the OS/VS START command) when a user-specified number of messages is reached. The utility is itself a COBOL program, and thus may be readily modified to accomodate more complicated scheduling requirements (e.g., based on time-of-day, or overall system load).

SEND #	Msg. Text	End Ind.	Advancing Clause	Comments
1	ABCDEF	'0'	After page	Partial segment, advancing ignored
2	G H	ESI		"After Advancing 1 line" implied
3	IJK	ESI	Before page	No advancing between segments
4	LMNO	EMI	Before 0 lines	



Note: Assume COBOPT3 has specified vertical positioning, forms feed, fixed line size of 4 bytes, and insert record delimiters.

Figure 163. Example of Message Formation for a Fixed Line Size Destination Supporting Vertical Positioning

Preparing the CJS: The following steps may be used for installing and tailoring the CJS utility:

1. The source code for CJS (module name ILBOCJS) is extracted from the COBOL installation tape by following the standard installation procedures. (See OS/VS COBOL Installation Reference Material, Order No. SC28-6481).

2. If desired, this source code may be modified to meet special local requirements (e.g., to change the PROGRAM-ID, to change scheduling criteria, to provide error recovery, etc.).

3. The source code should then be compiled and link-edited using normal local options for COBOL programs. The resulting load module must be placed in an authorized library with an authorization code of one (see the description of the Authorized Program Facility (APF) in the relevant VS Planning and Use Guide).

4. The TCAM MCP must contain a Terminal Table TPROCESS entry for "SYSCJS" and an associated PCB. Starting and ending messages will be directed to this entry by COBOL when a scheduled job starts and ends.

5. JCL should be coded for the CJS.

6. A special reader procedure, COBURDR, must be added as a member of SYS1.PROCLIB. The source for COBURDR is extracted from the COBOL installation tape by following the standard installation procedures (see

OS/VS COBOL Installation Reference Material).

7. The partitioned data set identified by the COBURDR reader procedure DD card, IEFRDER, must contain a member for each job name specified on a CJS Queue Polling Record (see "Using the CJS" below). Each member contains the JCL required to execute the COBOL program that will process the queue causing the scheduling to occur. The PARM option of the EXEC card for the COBOL program must identify the complete queue structure name corresponding to the TCAM queue:

```
PARM=(.../QUEUE(q-name
      [,sub-q-1-name[,sub-q-2-name
      [,sub-q-3-name]]]))
```

The member must also contain DD cards allowing the COBOL program to access the scheduling TCAM queue and to write the starting and ending messages to the CJS.

Using the CJS: To execute the CJS, control records must be provided specifying the queue polling requirements and the jobs to be started. They have the following format:

Header Record (one only; must be first record)

Column 1
Record identifier: must be "H".

Columns 2-6

Number of seconds to wait between pollings: this number indicates the number of seconds (of elapsed time) that CJS should wait between the time it polls all queues and the time it does so again.

Columns 7-11
Number of repetitions: this number indicates how many times the pollings (followed by the wait) should take place before CJS automatically completes.

Queue Polling Record

Column 1
Record identifier: must be "Q".

Columns 2-9
Queue Name: the name of a TCAM queue that is to be polled.

Columns 10-17
Job name: specifies the name of a member in the partitioned data set identified by the IEFRDER DD card in the COBURDR reader procedure that contains the job to be scheduled when the message count limit is reached for the corresponding queue.

Column 18
Association code: an alphanumeric code which may associate two or more Queue Polling Records. When a polling entry causes a job to be scheduled, that polling entry and all other entries with the same association code will not be re-polled until an ending message is received from the scheduled job. This facility is useful in assuring that two jobs are not scheduled at the same time to process the same TCAM queue.

Columns 19-23
Message count limit: when the number of completed messages on the polled queue reaches or exceeds this value, the associated job will be scheduled.

(The value specified for the OCCURS clause for "POLL-ENTRY" in the CJS program must be large enough to provide an entry for each queue polling record).

If the CJS terminates successfully, a return code (completion code) of zero is given. However if an error condition is detected, a return code of eight is given.

Figure 164 illustrates the CJS scheduling process.

Figure 165 provides an example of CJS control records and JCL for a communications job scheduling application.

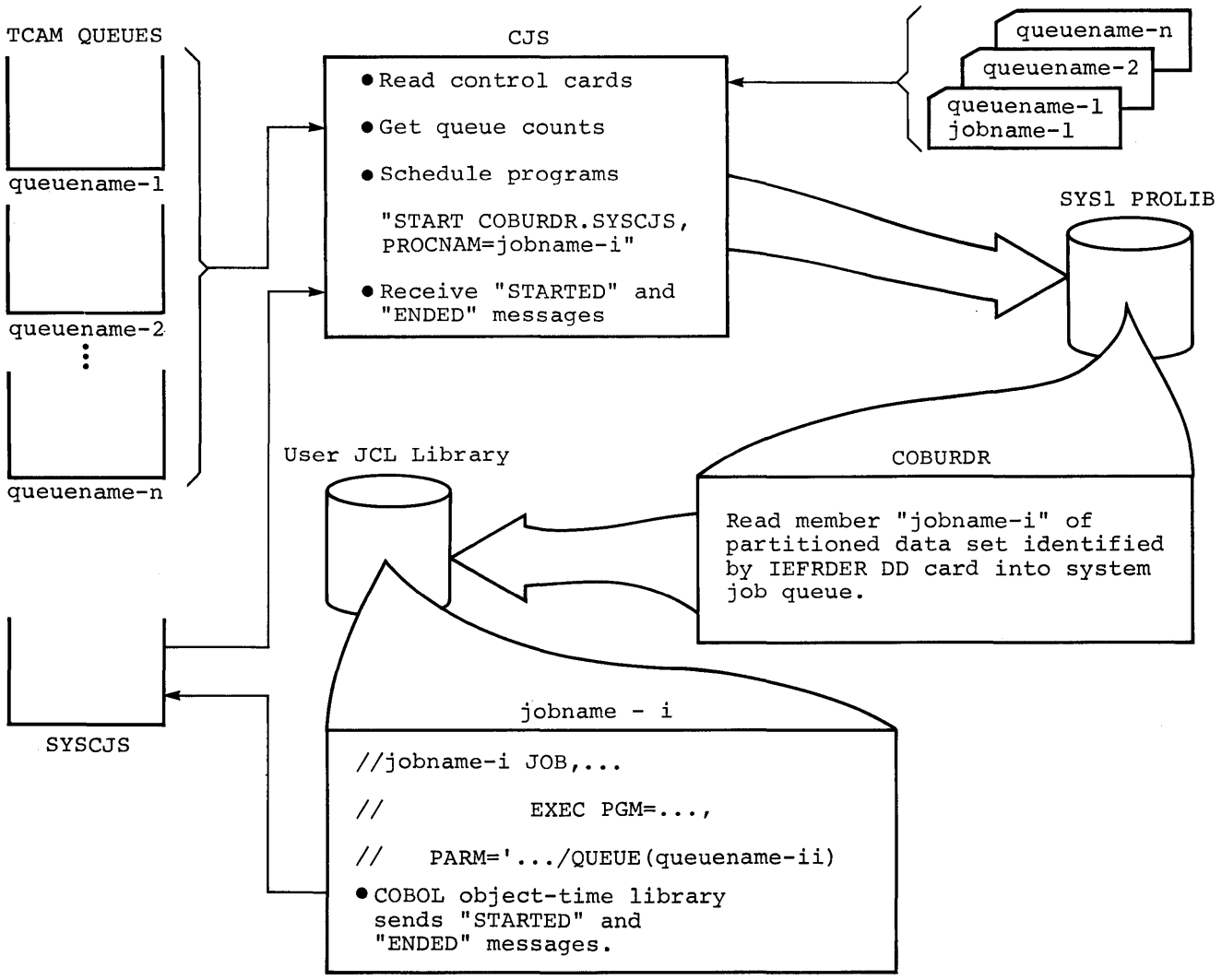


Figure 164. Communications Job Scheduling

In System Reader

```
//CJS      JOB   ...
//         EXEC  PGM=ILBOCJS
//STEPLIB  DD    DSN=SYS1.COBLIB,DISP=SHR
//SYSTPIN  DD    QNAME=SYSCJS
//SYSOUT   DD    SYSOUT=A
//SYSIN    DD    *
H0001000360
QQUEUE1  JOB1   00001
QQUEUE2  JOB2  A00004
QQUEUE3  JOB3  A00005
/*
```

In SYS1.PROCLIB

```
//COBURDR  PROC
           .
           .
           .
//IEFRDER  DD    DSN=userlib...
   (see OS/VS COBOL Instalation Reference
     Material)
```

In Userlib

```
Member:  JOB1

//JOB1     JOB   ...
//         EXEC  PGM=...,PARM='/QUEUE(SUBQ1)'
//SUBQ1    DD    QNAME=QUEUE1
           .
           .
           .
Member:  JOB2

//JOB2     JOB   ...
//         EXEC  PGM=...,PARM='/QUEUE(SUBQ2)'
//SUBQ2    DD    QNAME=QUEUE2
//SUBQ3    DD    QNAME=QUEUE3
           .
           .
           .
Member:  JOB3

//JOB3     JOB   ...
//         EXEC  PGM=...,PARM='/QUEUE(SUBQ3)'
//SUBQ2    DD    QNAME=QUEUE2
//SUBQ3    DD    QNAME=QUEUE3
           .
           .
           .
```

Figure 165. Sample CJS Application (Part 1 of 2)

Notes:

1. ILBOCJS must be an authorized program (via the OS/VS Authorized Program Facility)

2. SYSTPIN designates the MCP queue which will contain the messages sent by COBOL at the start and end of execution of a program which has been scheduled by the CJS.

3. SYSOUT is used for error messages (via the COBOL DISPLAY statement).

4. SYSIN holds the CJS control statements, as described in "Using the CJS" above.

5. The control statements in this example will do the following:

   - Every 10 seconds (of elapsed time), TCAM queues QUEUE1, QUEUE2, and QUEUE3 will be polled for their message count. After 360 polling intervals (approximately one hour), the CJS will terminate.

   - If QUEUE1 contains one or more messages, JOB1 will be started; QUEUE1 will not be polled again until an ENDED message is received specifying QUEUE1.

   - If QUEUE2 contains four or more messages, JOB2 will be started. Since both QUEUE2 and QUEUE3 have the same association code ("A"), neither will be polled again until an ENDED message is received specifying either QUEUE2 or QUEUE3.

   - If QUEUE2 contains less than four messages but QUEUE3 contains five or more messages, JOB3 will be started. Again, since both QUEUE2 and QUEUE3 have the same association code ("A"), neither will be polled again until an ENDED message is received specifying either QUEUE2 or QUEUE3.

6. Since both JOB2 and JOB3 reference the same TCAM queues (QUEUE2 and QUEUE3), they must be prevented from being scheduled at the same time. This is accomplished by specifying the same association code for each (when either one is scheduled, both are removed from the polling sequence).

Figure 165. Sample CJS Application (Part 2 of 2)

Summary of ANS Standard MCP Requirements

The MCP must contain provisions for only those ANS Standard features that will be used by COBOL application programs (e.g., if communications job scheduling is not to be used, the SYSCJS TPROCESS entry and its associated PCB need not be coded). An MCP containing provisions for all ANS Standard features is outlined in Figure 166. The following notes apply.

```
*****************************************************
*           MCP INITIALIZATION SECTION              *
*****************************************************
          INTRO       CONTROL=PASSWD,...
                                     (1)
           ...
          ILBONBLQ(2) ...
           ...
          READY       ...
           ...
*****************************************************
*             PROCESS CONTROL BLOCKS                *
*****************************************************
APPLPCB1  PCB (3)     MH=APPLMH,DATE=YES,(14)...
COBCMD1   PCB (4)     MH=COBCMDMH,BUFSIZE=80
COBCJS    PCB (5)     MH=COBCJSMH,BUFSIZE=51
           ...
*****************************************************
*                TERMINAL TABLE                     *
*****************************************************
          TTABLE      ...
COBOPT0(6) OPTION     CL9
COBOPT1(7) OPTION     FL1
COBOPT2(8) OPTION     FL1
COBOPT3(9) OPTION     XL11
CMDRDEST(10) OPTION   CL8
           ...
TERM1     TERMINAL(11) OPDATA=(PASSWD/T1,0,,                X
                       C3D6C2D6D7E3F37EA80008),...
QUEUE1(3) TPROCESS(12) PCB=APPLPCB1,DATE=YES,... (14)
QUEUE2(3) TPROCESS(13) PCB=APPLPCB1,DATE=YES,    (14)      X
                       OPDATA=(PASSWD/Q2,,1),...
COBTPOU1(3) TPROCESS   PCB=APPLPCB1,...
COBOPIN1(4) TPROCESS   PCB=COBCMD1,...
COBOPOU1(4) TPROCESS   PCB=COBCMD1,ALTDEST=COBOPIN1,(15)  X
                       SECTERM=YES,OPDATA=(,,,,COBOPIN1)(16)
SYSCJS(5) TPROCESS     PCB=COBCJS,...
           ...
*****************************************************
*      TYPICAL LINE GROUP MESSAGE HANDLER           *
*****************************************************
LINEMH    STARTMH     LC=...
           ...
*    IN-MESSAGE SUBGROUPS
          ILBONBLT(17)
           ...
          INMSG
           ...
          INEND
           ...
*    OUT-MESSAGE SUBGROUPS
          OUTMSG
           ...
          HOLD(18)       5X'FF',RELEASE,CONNECT=AND
           ...
```

Figure 166. ANS Standard MCP Requirements (part 1 of 2)

```
          OUTEND
           ...
*******************************************************
*   TYPICAL COBOL APPLICATION MESSAGE HANDLER         *
*******************************************************
APPLMH    STARTMH    LC=...
           ...
          INHDR
           ...
          FORWARD(18)  DEST=PUT
           ...
          OUTHDR
           ...
          SETEOF(20)   ...
           ...
          OUTEND
*******************************************************
*   ENABLE/DISABLE OPERATOR COMMAND MESSAGE HANDLER   *
*******************************************************
COBCMDMH  STARTMH     LC=OUT
           ...
          INHDR
          CODE (21)   NONE
           ...
          FORWARD     DEST=CMDRDEST(16)
           ...
          INEND
           ...
          OUTEND
           ...
*******************************************************
* COMMUNICATIONS JOB SCHEDULER (CJS) MESSAGE HANDLER  *
*******************************************************
COBCJSMH  STARTMH     LC=OUT
           ...
          OUTEND
```

Figure 166. ANS Standard MCP Requirements (Part 2 of 2)

(1) This parameter must be coded if ENABLE or DISABLE statements are to be executed by COBOL programs. The value specified constitutes the global portion of the enable/disable key and may consist of from one to eight characters without embedded blank comma or slash ("/") characters.

(2) This macro must be coded if DISABLE INPUT statements (without TERMINAL) are to be executed by COBOL programs.

(3) At least one PCB and its associated TPROCESS entries must be defined for each COBOL application program that can be executed concurrently (a PCB can be associated with only one application at a time). These PCB's and TPROCESS entries must specify DATE=YES.

(4) A PCB and two associated TPROCESS entries must be defined for each COBOL application program that can be executed concurrently and will execute ENABLE or DISABLE statements.

(5) If the Communications Job Scheduler utility (ILBOCJS) is to be executed, a PCB and associated TPROCESS entry must be coded.

(6) This option is required if local password protection is desired.

(7) This option is required if ENABLE/DISABLE INPUT TERMINAL statements are to be executed by COBOL programs.

(8) This option is required if ENABLE/DISABLE INPUT statements (without TERMINAL) are to be executed by COBOL programs.

(9) This option is required if characteristics are to be specified for symbolic destinations.

(10) This option allows the same message handler to be used for all PCB's required for ENABLE/DISABLE command processing (see item 4).

(11) The options for this entry specify:

1. a local password value of "PASSWD/T1".
2. the symbolic source represented by this entry may be enabled and disabled (it is initially enabled).
3. the symbolic destination represented by this entry has the following characteristics:
   - vertical positioning is supported
   - forms feed is not supported
   - the destination has a fixed line size of eight bytes.
   - record delimiters are not to be inserted between segments.
   - a test should be made during each SEND to see if this destination is disabled.

(12) The absence of the OPDATA option for this entry implies:

1. no local password protection
2. the symbolic input queue represented by this entry may not be disabled.

(13) The options for this entry specify:

1. a local password value of "PASSWD/Q2"
2. the symbolic input queue represented by this entry may be enabled and disabled (it is initially disabled).

(14) DATE=YES must be coded for TPROCESS entries representing symbolic input queues and their associated PCB's.

(15) This identifies COBOPIN1 as the destination for TCAM reply messages to operator commands received through COBOPOU1.

(16) The value of COBOPIN1 for the CMDRDEST option together with the FORWARD macro in the operator command message handler will send invalid operator commands back to the originating COBOL program (the program requires a response to each command issued).

(17) This macro must be coded if DISABLE INPUT TERMINAL statements are to be executed by COBOL programs for symbolic sources whose incoming messages are processed by this message handler.

(18) At least one HOLD macro must be coded in the MCP if DISABLE OUTPUT statements are to be executed by COBOL programs.

(19) This form of the FORWARD macro is required in message handlers that will process messages from COBOL programs.

(20) This macro may be used to cause an end of group indicator (EGI) to be passed to the COBOL program for the next RECEIVE statement executed after receipt of the end of message indicator (EMI) for the current message. The RECEIVE statement must specify the sam symbolic queue as that for the message causing the SETEOF macro to be executed and will have an accompanying text length of zero.

(21) This macro must be coded to allow TCAM to detect valid operator commands resulting from ENABLE/DISABLE statements executed by COBOL programs. A valid operator command will result if the key phrase of the ENABLE/DISABLE statement yields a valid global password. See item 16 for invalid operator commands.

JCL FOR THE MCP

This section names the parts of the MCP described earlier, explaining how to arrange them in relation to one another and how to assemble, link-edit, and execute a TCAM MCP. The five sections of an MCP include those previously discussed -- an activation and deactivation section, a data set definition section, a terminal and line control area section, a message handler section -- and an optional user routine section (that is, user subroutines called by a message handler, as well as exit routines referred to by the INTRO macro, by DCB macros, and by the STARTMH macro). The only stipulation about ordering these sections is that the activation and deactivation section must come first.

ASSEMBLING, LINK-EDITING, AND EXECUTING AN MCP

The assembly, link-edit, and execution steps of a TCAM MCP are similar to these steps for any other problem program running

under OS/VS. The job control statements given below for these three steps are guidelines only.

Assembling an MCP

A typical control card sequence for assembling a TCAM MCP is as follows:

```
//ASSEMBLY  JOB   MSGLEVEL=1
//STEP1     EXEC ASMFC
//ASM.SYSIN DD    *

{MCP Source Deck}
```

Link-Editing an MCP

The following is a typical control card sequence for link-editing an MCP:

```
//LINKEDIT  JOB   MSGLEVEL=1
//STEP1     EXEC  PGM=IEWL,PARM='XREF,LIST,
//                LET',REGION=128K
//SYSPRINT  DD    SYSOUT=A
//SYSUT1    DD    UNIT=SYSDA,
//                SPACE=(1024,(200,20))
//SYSLMOD   DD    DSNAME=SYS1.TCAMLIB,
                  DISP=OLD
//SYSLIB    DD    DSNAME=SYS1.TELCMLIB,    X
                  DISP=SHR
//SYSLIN    DD    *

{MCP Object Module}

            NAME  TCAMPROG(R)
```

Note: In this example, the MCP load module is to be placed in a user-created private library called SYS1.TCAMLIB.

Executing an MCP

The TCAM MCP is normally executed as the highest-priority task in the highest-priority partition or region in the system. It may have an equal priority, but it should never be assigned a lower priority. A typical control card sequence for executing an MCP is the following:

```
//EXECMCP   JOB  'EXECUTE MCP',MSGLEVEL=1, X
//                PRTY=12
//GOSTEP    EXEC PGM=TCAMPROG,REGION=100K
//STEPLIB   DD   DSNAME=SYS1.TCAMLIB,      X
//                DISP=SHR
//DD1050    DD   UNIT=025
//          DD   UNIT=026
//          DD   UNIT=027
//DD2740    DD   UNIT=015
//          DD   UNIT=016
//          DD   UNIT=017
//QFILE     DD   DSNAME=MSGQ,DISP=OLD
//LOGFILE   DD   DSNAME=LOGF,DISP=OLD
//SYSABEND  DD   SYSOUT=A
```

Notes:

1. In this example, the MCP has two line group data sets, each containing three lines; no checkpoint facility is included. (For a discussion of the DD cards for a checkpoint data set, see the section "Defining the Checkpoint Data Set.")

2. The QFILE DD statement is for a message queues data set residing on disk; QFILE is the name specified in the DDNAME= operand of the DCB macro for this data set, and MSGQ is the name of the data set specified by the DSNAME operand of the IEDQDATA DD statement for the IEDQXA utility used to preformat disk message queues data sets residing on disk (see the section "Defining the Message Queues Data Sets").

3. If the data set is not cataloged, the UNIT= and VOLUME= operands must be included in the DD statement for the disk message queues data set.

4. The //LOGFILE DD card must be included if the LOG data set is to be used.

Defining the Checkpoint Data Set: One DD statement that may or may not catalog the data set must be issued for the checkpoint data set. However, if it is not cataloged, the user should allocate the data set by specifying DISP=(NEW,KEEP) as in the example and subsequent uses of the data set must contain the UNIT= and VOL=SER=keyword operands, given below.

```
//CFILE     DD    DSNAME=CPDS,UNIT=2314,   X
//                VOL=SER,DB197,           X
//                SPACE=(TRK,(5)),         X
//                DISP=(NEW,KEEP)
```

After a checkpoint data set is set up and the MCP has terminated normally, the programmer should replace the DD card described above with one of the following type:

```
//CFILE     DD    DSNAME=CPDS,DISP=OLD,
                  VOL=SER=DB197,UNIT=2314
```

Defining Line Group Data Sets: The user must include in his job control statements at least one DD statement for each line group data set, but he has two options for handling these definitions.

1. If a UNITNAME macro is issued for a line group at system generation time, then a single DD statement may be issued for this line group at MCP execution time. For example, a UNITNAME macro could be issued to define a group of lines as follows:

   ```
   UNITNAME   UNIT=(040,041)
   NAME=GROUPLINE
   ```

   Where the two numerals in the UNIT=operand parameter represent the hardware addresses of two lines in a line group. At execution time for the MCP, the following DD statement might be issued for this line group:

   ```
   //LNS  DD  UNIT=(GROUPLINE,2)
   ```

   Where the line group data set would be made up of two lines defined by the UNITNAME macro.

2. A DD statement may be issued for each line in a line group, as in the DD cards for line group DD1050 and line group DD2740 in the sample JCL statements given in section "Executing an MCP."

3. The following DD cards were used to execute the sample message control program shown in Figure 160.

```
//LN1  DD  UNIT=040 (for the 1050 terminal)
//LN2  DD  UNIT=041 (for the TWX terminal)
```

Defining the Message Queues Data Sets: The number of message queues data sets required for an MCP depends on the types of queues, which, in turn, depend on the application. TCAM supports three types of data sets -- a main storage data set, a reusable disk set, and a nonreusable disk data set. (For checklists governing specification of the three types of message queues data sets, see the publication OS/VS TCAM Programmer's Guide.

TCAM expects the disk message queues (both reusable and nonreusable) to be totally preformatted. The COBOL user should engage the TCAM routine IEDQXA to perform this task prior to initially of a set of job control statements used to invoke this routine.

Note: The value given in the KEYLEN parameter must be the same as that specified in the KEYLEN operand of the INTRO macro (see the section "Defining the Buffers").

```
//JOBNAME  JOB  user information
//FORMATQ  EXEC PGM=IEDQXA
//SYSPRINT DD   SYSOUT=A
//IEDQDATA DD   DSN=MSGQ,DISP=(,CATLG),    X
//              SPACE=(CYL,(5,5),,CONTIG)  X
//              UNIT=(2314,1),             X
//              VOL=SER=333333,            X
//              DCB=(,KEYLEN=100)
/*
```

WRITING A TCAM-COMPATIBLE COBOL PROGRAM

Two of the chief processing applications for which COBOL programs can be written are inquiry processing and processing collected data. An inquiry-processing COBOL program receives messages from stations, processes the data, and then sends replies to the originating stations. Depending on the inquiry, the COBOL program may transmit either the information requested or a message stating that this information is unavailable and telling when it can be provided. The COBOL program that simply processes data collected by a message control program can either operate concurrently with the collection of data by the MCP or be loaded and initiated at a later time.

The sample COBOL communication program TESTTP2 (shown in Figure 171) represents an application of processing data. This program accepts messages transmitted from a remote station, formats the message, and then transmits each complete message to the destination specified. The COBOL program TESTTP1 (shown in Figure 170) simulates terminal input data. The user can, therefore, test an installation-written COBOL TP program by running it with the sample MCP and TESTTP1.

TESTING A COBOL TP PROGRAM

Depending on the status of an installation's teleprocessing system, the user can code any one of three sets of JCL to run a teleprocessing job. A system that is fully operational has a message control program with a user-designated message

handler for each type of teleprocessing situation expected, as well as remote terminal hook-ups. The user whose system is only partially developed or is still in the design stage may, nevertheless, wish to test COBOL teleprocessing programs using BSAM.

Accordingly, the JCL shown in Figure 167 is for a strictly BSAM situation (that is, for a communication program that is to be run without TCAM); the JCL shown in Figure 168 is for a quasi-terminal situation (that is, with TCAM but without terminals); and the JCL shown in Figure 169 is for a communications job running with a remote terminal. For both the non-terminal and the quasi-terminal situation an input data set must be created. To run a COBOL communications program with a terminal hook-up, only the communications program itself is needed.

To avoid unresolved external references from link edit when testing with BSAM and NORES, the programmer must point to the SYS1.TELCMLIB. (He may choose to override these unresolved references since they will not effect BSAM testing.) When testing with BSAM and RES, the programmer need not take any extra steps.

When testing TCAM the user must provide a LKED.SYSLIB card for SYS1.TELCMLIB when NORES is specified. A GO.STEPLIB card must be provided for SYS1.TELCMLIB.

```
//TESTTP1      JOB      user information
//             EXEC     COBUCLG
//COB.SYSIN    DD       *
               {Source deck for TESTTP1 program (Figure 163)}
/*
//GO.TSTTP     DD[1]    UNIT=2400,LABEL=(,NL),VOL=SER=NI195,DCB=(LRECL=50,BLKSIZE=50,   X
//                      RECFM=F,DEN=2)
//GO.COBTPOUT  DD[2]    UNIT=2314,VOL=SER=231400,DSN=&P1,DISP=(NEW,PASS),              X
//                      SPACE=(CYL,(1,2))
//TESTTP2      JOB      user-information
//             EXEC     COBUCLG
//COB.SYSIN    DD       *
               {Source deck for TESTTP2 program (Figure 166)}
/*
//GO.Q1        DD[3]    DSN=&P1,VOL=SER=231400,UNIT=2314,DISP=(OLD,PASS)
//GO.COBTPOUT  DD[4]    DSN=&P2,VOL=SER=231400,UNIT=2314,DISP=(NEW,PASS),              X
//                      SPACE=(CYL,(1,2))
//DUMPIT       JOB      user-information
//             EXEC[5]  PGM=xMASPZAP
//SYSLIB       DD       DSNAME=data set to be printed,UNIT=2314,VOL=SER=231400,        X
//                      DISP=OLD,DCB=DSORG=PS
//SYSPRINT     DD       SYSOUT=A
//SYSIN        DD       *
               ABSDUMP  ALL
/*
```

Notes:

1. Input sequential file with records of 50 characters each (BSAM JCL).
2. Output data set that simulates sending messages to a terminal named 'P1'.
3. Input data set that simulates reading messages from a terminal named 'P1'.
4. Output data set that simulates sending messages to a terminal named 'P2'.
5. This job prints out the records in the simulated data set. For further information, see the publication OS/VS Service Aids.
6. For OS/VS1: HMASPZAP; for OS/VS2: AMASPZAP.

Figure 167. Sample JCL for Testing a Communication Job Without TCAM.

```
//TESTTP1     JOB    user information
//           EXEC   COBUCLG
//COB.SYSIN   DD     *
                   {Source deck for TESTTP1 program (Figure 165)}
/*
//GO.STEPLIB  DD     DSN=SYS1.TELCMLIB,DISP=SHR
//GO.TSTTP    DD1    UNIT=2400,LABEL=(,NL),VOL=SER=NI195,DCB=(LRECL=50,BLKSIZE=50,    X
//                   RECFM=F,DEN=2)
//GO.COBTPOUT DD2    QNAME=POUT1
//TESTTP2     JOB    user information
//           EXEC   COBUCLG
//COB.SYSIN   DD     *
                   {Source deck for TESTTP2 program (Figure 166)}
/*
//GO.STEPLIB  DD     DSN=SYS1.TELCMLIB,DISP=SHR
//GO.Q1       DD3    QNAME=P1
//GO.COBTPOUT DD4    QNAME=POUT1
//DUMPIT      JOB    user information
//           EXEC   IEDQXC
//DISQ01      DD5    DSN=MSGQ,VOL=SER=DB197,UNIT=2314,DISP=SHR
//SYSPRINT    DD     SYSOUT=A
/*
```

Notes:

1. Input sequential file with records of 50 characters each. (This is the same JCL as in BSAM.)
2. Output is sent to an MCP message queue named 'P1', which is defined for processing by a COBOL program.
3. Input is received from the MCP message queue named 'P1'.
4. Output is sent to an MCP message queue named 'P2', which is defined for processing by a COBOL program.
5. This job prints out records in the MSGQ queue. For further information, see the publication OS/VS TCAM Programmer's Guide.

Figure 168. Sample JCL for Running a Communication Job in a Quasi-Terminal Environment.

```
//TESTTP2     JOB  user information
//           EXEC COBUCLG
//COB.SYSIN   DD   *
                 {Source deck for TESTTP2 program (Figure 154)}
/*
//GO.STEPLIB  DD   DSN=SYS1.TELCMLIB,DISP=SHR
//GO.Q1       DD1  QNAME=P1
//GO.COBTPOUT DD2  QNAME=POUT1
```

Notes:

1. The input is received from the MCP message queue 'P1'.
2. The output is sent to an MCP message queue defined for a terminal.

Figure 169. Sample JCL for Running a Communication Job with a Remote Terminal

COMMUNICATING BETWEEN A COBOL PROGRAM AND THE MCP

The TCAM message control program routes messages between a COBOL teleprocessing program and remote stations. Because the MCP performs the input/output operations necessary for the COBOL teleprocessing program, the user must establish an interface between these two programs by doing the following:

- Defining the interface
- Activating the interface
- Transferring messages between the COBOL program and the MCP
- Deactivating the interface

In each of the sections that follow, both COBOL statements and TCAM macros, as well as their relationship, are described as appropriate. The encircled numerals in this discussion refer to the sections similarly labeled in the sample COBOL teleprocessing program TESTTP2 shown in Figure 171.

Defining the Interface

The Communication Section in the COBOL program and the PCB and TPROCESS macros in the message control program set up the interface between the two programs.

Associating COBOL Symbolic Queue Names with TCAM Queues: COBOL associates the symbolic queue names specified in an input CD with TCAM queues according to the following algorithm:

1. If a queue structure definition (created by the queue structure utility, ILBOQSU) exists for the SYMBOLIC QUEUE name in the CD, the SYMBOLIC SUB-QUEUE-1 through SYMBOLIC SUB-QUEUE-3 names are used to derive a VS JCL ddname value from the definition.
2. If a queue structure definition does not exist for the SYMBOLIC QUEUE name in the CD and SYMBOLIC SUB-QUEUE-1 through SYMBOLIC SUB-QUEUE-3 names contain spaces, the first eight characters of the SYMBOLIC QUEUE name are used as the ddname.
3. If neither of the above steps yields a ddname, the queue is unknown (CD STATUS KEY = '20').

Any ddname produced by the algorithm above must have a corresponding DD card of the following format:

```
//ddname DD QNAME=procname
            [,DCB=BLKSIZE=n]
```

where procname is the name of the TCAM TPROCESS Terminal Table entry with which the input queue is to be associated, and n is the blocksize of the TCAM buffer used by COBOL. The BLKSIZE parameter is only effective for the first input or output queue opened (COBOL uses a single buffer for all TCAM queues) and has a default value of 200.

Identifying the Queue Structure Definition Data Set: The queue structure definitions created by the queue structure utility, ILBOQSU, are made available to COBOL by specifying the following DD card:

```
//COBTPQD DD data set information
```

Associating COBOL Symbolic Source Names with TCAM Sources: The first eight characters of each symbolic source name used by COBOL must be the name of a TCAM TERMINAL or TPROCESS Terminal Table entry that can act as a source for incoming messages. No special DD cards are required.

Associating COBOL Symbolic Destination Names with TCAM Destinations: The first eight characters of each symbolic destination name used by COBOL must be the name of a TCAM TERMINAL or TPROCESS Terminal Table entry that can act as a destination for outgoing messages. In addition, the following DD card is required:

```
//COBTPOUT DD QNAME=procname
              [,DCB-BLKSIZE=N]
```

WHERE PROCNAME IS THE NAME OF A TCAM TPROCESS Terminal Table entry that can act as a source for incoming messages and n is the blocksize of the TCAM buffer used by COBOL. The BLKSIZE parameter is only effective for the first input or output queue opened (COBOL uses a single buffer for all TCAM queues); it has a default value of 200.

This DD card is also required if a CD FOR INITIAL INPUT is defined in the COBOL program and the program is scheduled by the CJS.

Specifying the ENABLE/DISABLE Command Interface: If ENABLE or DISABLE statements are to be executed by the COBOL program, the following DD cards are required:

```
//COBOPOUT  DD  QNAME=procname-1
//COBOPIN   DD  QNAME=procname-2
```

where procname-1 is the name of a TCAM TPROCESS Terminal Table entry defined as a secondary operator control station, and procname-2 is the name of a TCAM TPROCESS Terminal Table entry to which operator command reply messages are to be sent (see "ENABLE/DISABLE: Operator Command Interface").

```
001010 IDENTIFICATION DIVISION.
001020 PROGRAM-ID.
001030     TESTTP1.
001080 DATE-COMPILED.  MAY 1, 1974
001100 REMARKS.       THE SAMPLE COBOL TELEPROCESSING PROGRAM THAT
           FOLLOWS SERVES AS A SIMPLE ILLUSTRATION OF THE COBOL TELE-
           PROCESSING FEATURE.  THIS PROGRAM READS IN A FILE OF 50-
           CHARACTER MESSAGES, TRANSMITTING THEM ONE BY ONE TO THE
           SPECIFIED DESTINATION.
001160
001170 ENVIRONMENT DIVISION.
001180 INPUT-OUTPUT SECTION.
001190 FILE-CONTROL.
001200     SELECT MASTER-FILE
001210     ASSIGN TO UT-2400-S-TSTTP.
002010 DATA DIVISION.
002020 FILE SECTION.
002030 FD  MASTER-FILE
002040     RECORDING MODE IS F
002050     LABEL RECORDS ARE STANDARD
002060     DATA RECORD IS RECORD1.
002070 01  RECORD1  PIC  X(50).
003010 WORKING-STORAGE SECTION.
003110
003120 01  IDENT-SEND.
003130     02  I-SEND   PIC   X(50).
      *  SET UP A WORK AREA OF 50 CHARACTERS
003150
      *  THE COMMUNICATION SECTION MUST BE SPECIFIED IN A COBOL PROGRAM
      *  THAT IS TO UTILIZE THE COBOL TELEPROCESSING FEATURE.  THE
      *  COMMUNICATION DESCRIPTION (CD) ENTRIES THAT APPEAR IN THIS
      *  GROUP OF SOURCE STATEMENTS ESTABLISH THE INTERFACE BETWEEN THE
      *  COBOL OBJECT PROGRAM AND THE MESSAGE CONTROL PROGRAM (MCP).
004010 COMMUNICATION SECTION.
004120 CD  CDNAME-OUT FOR OUTPUT
004130     TEXT LENGTH IS TEXTLNTH-OUT
      *  SPECIFY LENGTH OF OUTPUT MESSAGE.
004140     STATUS KEY IS STATKY-OUT
      *  PROVIDE INFORMATION ON MESSAGE STATUS.
004150     ERROR KEY IS ERRKY
      *  PROVIDE ERROR INFORMATION.
004160     SYMBOLIC DESTINATION IS SYMDES.
      *  SPECIFY OUTPUT QUEUE.
004170
005010 PROCEDURE DIVISION.
005020 START-JOB.
005030     DISPLAY 'BEGIN TESTTP1'.
      *  START THE COBOL TELEPROCESSING PROGRAM.
005040     OPEN INPUT MASTER-FILE.
      *  OPEN THE INPUT FILE.
005045 READ-ROUTINE.
005050     READ MASTER-FILE INTO IDENT-SEND
005060         AT END GO TO END-ROUTINE.
      *  PLACE INPUT RECORDS IN A WORK AREA UNTIL END OF FILE IS
      *  REACHED.
006010 SEND-ROUTINE1.
006020     MOVE 'P1' TO SYMDES.
      *  SET UP OUTPUT DESTINATION.
006040     MOVE 50 TO TEXTLNTH-OUT.
      *  IDENTIFY MESSAGE LENGTH AS 50.
006060     SEND CDNAME-OUT FROM IDENT-SEND WITH EMI.
      *  TRANSMIT A COMPLETE MESSAGE.
006070     PERFORM CHECK-SEND THRU CHECK-EXIT.
006080     GO TO READ-ROUTINE.
```

Figure 170. Creating a TCAM Data Set for Testing without Terminals (Part 1 of 2)

```
      *  EXECUTE USER-WRITTEN CODE FOR CHECKING ON THE SUCCESSFUL
      *  COMPLETION OF MESSAGE TRANSMISSION.  IF END OF FILE IS
      *  REACHED, GO TO END-OF-JOB ROUTINE.  OTHERWISE, GET THE NEXT
      *  RECORD.
  008010 CHECK-SEND.
  008020*
  008021*
  008022*
  008030*  USER CHECKING ROUTINE FOR DETERMINING THE
  008040*  SUCCESSFUL COMPLETION OF THE SEND.
  008050*
  008160
  008170 CHECK-EXIT.
  008180     EXIT.
**008180*
  008190
  011110 END-ROUTINE.
  011111     CLOSE MASTER-FILE.
      *  CLOSE THE INPUT FILE.
  011150     DISPLAY 'SUCCESSFUL END OF TESTTP1'.
      *  TERMINATE THE PROGRAM.
  011160     STOP RUN.
```

Figure 170. Creating a TCAM Data Set for Testing without Terminals (Part 2 of 2)

```
001010 IDENTIFICATION DIVISION.
001020 PROGRAM-ID.
001030     TESTTP2.
001080 DATE-COMPILED. MAY  1, 1974
001100 REMARKS.       THE SAMPLE COBOL TELEPROCESSING PROGRAM THAT
           FOLLOWS SERVES AS A SIMPLE ILLUSTRATION OF THE COBOL TELE-
           PROCESSING FEATURE.  THIS PROGRAM SETS UP A DESTINATION
           FOR INCOMING MESSAGES, AND THEN READS THEM, ONE BY ONE,
           INTO A WORK AREA.  THE PROGRAM BUILDS 50-CHARACTER MESSAGES
           AND SENDS THEM TO THE MCP WITH THE END-OF-MESSAGE (EMI)
           INDICATOR.  WHEN ALL THE INCOMING MESSAGES HAVE BEEN PRO-
           CESSED, THE MESSAGE 'SUCCESSFUL END OF TESTTP2' IS PRINTED
           ON THE CONSOLE, AND THE PROGRAM IS TERMINATED.
001120
001130
001170 ENVIRONMENT DIVISION.
001180 CONFIGURATION SECTION.
001190 INPUT-OUTPUT SECTION.
001200
002010 DATA DIVISION.
       WORKING-STORAGE SECTION.
003110
003120 01  IDENT-SEND.
003130     02  I-SEND   PIC  X(50).
003160 01  IDENT-REC.
003170     02  I-REC  PIC  X(50).
003190
      *  THE COMMUNICATION SECTION MUST BE SPECIFIED IN A COBOL PROGRAM
      *  THAT IS TO UTILIZE THE COBOL TELEPROCESSING FEATURE.  THE
      *  COMMUNICATION DESCRIPTION (CD) ENTRIES THAT APPEAR IN THIS
      *  GROUP OF SOURCE STATEMENTS ESTABLISH THE INTERFACE BETWEEN THE
      *  COBOL OBJECT PROGRAM AND THE MESSAGE CONTROL PROGRAM (MCP).
       COMMUNICATION SECTION.
004120 CD  CDNAME-OUT FOR OUTPUT
004130     TEXT LENGTH IS TEXTLNTH-OUT
      *  SPECIFY LENGTH OF OUTPUT MESSAGE.
004140     STATUS KEY IS STATKY-OUT
      *  PROVIDE INFORMATION ON OUTPUT MESSAGE STATUS.
004150     ERROR KEY IS ERRKY
      *  PROVIDE ERROR INFORMATION.
004160     SYMBOLIC DESTINATION IS SYMDES.
      *  SPECIFY OUTPUT QUEUE.
004170
**004020 CD  CDNAME-IN FOR INPUT
004030     SYMBOLIC QUEUE IS SYMQ
      *  IDENTIFY INPUT MESSAGE QUEUE.
004040     MESSAGE DATE IS MSGDATE
004050     MESSAGE TIME IS MSGTIME
      *  PROVIDE DATE AND TIME OF RECEIPT OF MESSAGE.
004060     SYMBOLIC SOURCE IS SYMSOURCE
      *  IDENTIFY THE MESSAGE SOURCE.
004070     TEXT LENGTH IS TEXTLNTH-IN
      *  SPECIFY THE EXPECTED LENGTH OF INPUT MESSAGE.
004080     END KEY IS ENDKY
      *  PROVIDE CODE FOR ACTIVATING END-OF-JOB ROUTINE.
      ** FOR A RECEIVE MESSAGE:
      *  A CODE OF 3 INDICATES END OF GROUP (EGI).
      *  A CODE OF 2 INDICATES END OF MESSAGE (EMI).
      *  A CODE OF 0 INDICATES RECEIPT OF LESS THAN A MESSAGE.
      ** FOR A RECEIVE SEGMENT:
      *  A CODE OF 3 INDICATES END OF GROUP (EGI).
      *  A CODE OF 2 INDICATES END OF MESSAGE (EMI).
      *  A CODE OF 1 INDICATES END OF SEGMENT (ESI)
      *  A CODE OF 0 INDICATES RECEIPT OF LESS THAN A SEGMENT.
```

Figure 171. A COBOL Program That Processes TCAM Messages (Part 1 of 2)

```
      ** HIERARCHY -- 0, ESI, EMI, ETI-WHEN MORE THAN ONE CONCURRENTLY-
      *    HIGH LEVEL APPEARS.
  004090     STATUS KEY IS STATKY-IN
       PROVIDE INFORMATION ON INPUT MESSAGE STATUS.
  004100     MESSAGE COUNT IS MCOUNT.
      *  SPECIFY MESSAGE COUNT FOR INPUT QUEUE.
  004110
 **002100
       PROCEDURE DIVISION.
(2)        DISPLAY 'BEGIN TESTTP2'.
       RECV-DATA.
  009040     MOVE 'Q1' TO SYMQ.
      *  SET UP INPUT DESTINATION.
(3) 009050     RECEIVE CDNAME-IN MESSAGE INTO IDENT-REC
  009055         NO DATA GO TO END-ROUTINE.
      *  ACCEPT INPUT MESSAGES, ONE BY ONE, AS ON A SEQUENTIAL FILE.
      *  WHEN ALL MESSAGES HAVE BEEN PROCESSED, INVOKE END-OF-JOB
      *  ROUTINE.
  009060 CHECK-RECEIVE.
  009070*
  009080*  USER CHECKING ROUTINE FOR DETERMINING THE
  009090*  SUCCESSFUL COMPLETION OF THE RECEIVE.
  009100*
  009110 PROCESS-DATA.
  009120*
  009130*  USER ROUTINE TO BUILD MESSAGE TO BE SENT.
  009140*
 **006010 SEND-ROUTINE1.
  006020     MOVE 'P2' TO SYMDES.
      *  SET UP OUTPUT DESTINATION.
      *  NOTE:  FOR THE NON-TERMINAL AND PARTIAL TERMINAL SITUATIONS,
      *  'P2' SHOULD BE SPECIFIED AS THE SYMBOLIC DESTINATION.  FOR
      *  A COBOL PROGRAM RUNNING WITH TERMINALS, 'T1' SHOULD BE
      *  SPECIFIED.
  006040     MOVE 50 TO TEXTLNTH-OUT.
      *  SPECIFY LENGTH OF OUTPUT MESSAGES.
(4) 006060     SEND CDNAME-OUT FROM IDENT-SEND WITH EMI.
      *  TRANSMIT FORMATTED MESSAGE, WITH THE CODE FOR A COMPLETE
      *  MESSAGE.
  006070     PERFORM CHECK-SEND THRU CHECK-EXIT.
      *  INVOKE USER-WRITTEN ROUTINE FOR CHECKING MESSAGE TRANSMISSION.
      *  ACCEPT THE NEXT MESSAGE FROM THE INPUT QUEUE.
  006090     GO TO RECV-DATA.
  007120
  008010 CHECK-SEND.
  008020*
  008030*  USER CHECKING ROUTINE FOR DETERMINING THE
  008040*  SUCCESSFUL COMPLETION OF THE SEND.
  008050*
  008170 CHFCK-EXIT.
  008180     EXIT.
  008190
  011110 END-ROUTINE.
  011150     DISPLAY 'SUCCESSFUL END OF TESTTP2'.
(5) 011160     STOP RUN.
```

Figure 171. A COBOL Program That Processes TCAM Messages (Part 2 of 2)

Defining Process Control Blocks: In the MCP the user must also code a process control block (PCB) for each active application program running with the MCP. The PCB macro specifies the name of the PCB process control block generated by the macro. The process control block is referred to in the TPROCESS macro (see "Defining the MCP Data Sets and Process Control Blocks").

Activating the Interface

The COBOL programmer coding a program for a teleprocessing application initializes work areas, (1) , and activates the COBOL program as for any other OS/VS application. In this application, the job begins with the use of the DISPLAY statement "BEGIN TESTTP2," (2). The COBOL programmer need not be concerned with how the interface is activated. The interface is activated when the first RECEIVE or SEND statement is issued.

Transferring Messages between the COBOL Program and the MCP

TCAM enables the application programmer to obtain messages from the MCP and to return response messages to the MCP. Specifically, the COBOL programmer can use either the RECEIVE statement or the SEND statement to transfer data between the MCP and the COBOL program, depending on the direction of the flow of data.

The RECEIVE Statement: This COBOL source statement causes transmission of message data from an input queue to a user-specified work area in the COBOL program. In the sample COBOL teleprocessing program shown in Figure 166, the RECEIVE statement, (3) , transfers data from the input queue referred to by SYMQ to a work area. The COBOL sentence before the RECEIVE statement is "MOVE 'Q1' TO SYMQ," so the data is received from Q1.

The SEND Statement: The COBOL source statement causes data from the COBOL program to be placed in an output queue for subsequent transmission. Accordingly, when the outgoing message has been formatted, the sample SEND statement, (4) , transmits it to the output destination referred to by SYMDES. The end-of-message indicator (EMI) signals a complete message. The first sentence in the paragraph labeled "SEND-ROUTINE1" is "MOVE 'P2' TO SYMDES," so the data is sent to P2.

Notes:

- For an additional example of the format of the RECEIVE statement and the SEND statement, see the section "Procedure Division" in the chapter on "Programming Techniques".
- The amount of data transferred from the MCP to a COBOL program by a single RECEIVE statement, or transferred from an application program to the MCP by a single SEND statement, is called a "work unit". Each work unit is processed in a user-designated work area in the COBOL program.

Deactivating the Interface

As in all American National Standard COBOL programs, the teleprocessing application user returns control to the system by issuing a STOP RUN statement, (5).

Note: So that the COBOL program can give control to the STOP RUN statement, the MCP writer should include in the message header a special code for the COBOL program. Although the sample MCP (Figure 160) has an action code field which includes such a code in the section of comments immediately preceding the MH1050 message handler, 13 , Figure 166 gives control to the STOP RUN statement only when there is no more data. This technique is acceptable for a COBOL program that receives a fixed amount of data, i.e., a program that is not continually looping waiting for data. Alternatively, the MCP macro SETEOF can be used to cause an EGI, which could be used as an indication that STOP RUN processing should be initiated.

Additional Interface Considerations

The information that follows is a summary of miscellaneous recommendations and/or restrictions that apply to the communication between the message control program and the COBOL application program.

1. The parameter DATE=YES must be coded in all input TPROCESS entries whose destination is a COBOL program and the parameter is also required in the PCB macro referenced by the TPROCESS macro. Inclusion of this parameter causes the date and time of message entry to be placed in the MESSAGE DATE and MESSAGE TIME clauses of the input CD (see "Communication Section" in the

chapter entitled "Programming Techniques").

2. The RECDEL= parameter must be coded in the TPROCESS macro of the MCP if the COBOL programmer is to accept (via the RECEIVE statement) or transmit (via the SEND statement) data in SEGMENT mode. The user may either include in the incoming message the delimiter specified in this parameter or insert it via a MSGEDIT macro (see the section "Designing the Message Handler" in this chapter).

3. The INITIATE macro cannot be used in a message handler for messages whose destination is a COBOL program. This macro would cause the MCP to transmit segments of a message to a destination queue before receiving the complete message. American National Standard COBOL, on the other hand, assumes that a complete message has been enqueued.

4. American National Standard COBOL removes the last character of a message if it is X'37' (which is the EBCDIC representation for the EOT character). This is the last character of a message from a terminal that has been translated in the MH of the MCP via the CODE macro, or that is not processed in conversational mode (which would have been specified by coding CONV=YES in the STARTMH macro).

5. An execution of the RECEIVE statement with the SEGMENT option results in the setting of the ESI (end of segment) indicator if end of segment is reached. When end of segment is also end of message, an end key of 2 indicating EMI is given. If the last two characters in the message are an end segment indicator and the end of message character, the user will receive the ESI indication first. Another RECEIVE will be necessary to receive the EMI indication. The RECEIVE from the EMI indication will set the TEXT LENGTH field of the input CD to zeros.

6. For a message transmitted from a COBOL program to the location specified in the SYMBOLIC DESTINATION clause of an output CD, the FORWARD macro in the inheader subgroup of the MH for the COBOL program must specify DEST=PUT as its operand.

USING TCAM SERVICE FACILITIES

TCAM allows for a variety of services in support of a COBOL communication system. Some of these services are provided automatically; others the user must specify. Some of the TCAM services are the following: operator control, error recovery, checkpoint/restart, message logging, debugging aids, and an on-line test feature. All of these TCAM aids are discussed in the publication OS/VS TCAM Programmer's Guide.

This chapter contains information concerning system requirements for the COBOL compiler, execution time, and the sort/merge feature. Additional information for use in estimating the main and auxiliary storage requirements is contained in the publication OS/VS COBOL Compiler and Library Installation Reference Material.

MINIMUM MACHINE REQUIREMENTS

The basic system requirements for use of the COBOL compiler are:

- A System/370 model, with the standard and decimal instruction sets. The floating-point instruction set is required if floating-point data items and fractional exponents are used in the program.
- Compiler Work Files -- Six utility data sets named SYSUT1, SYSUT2, SUSYT3, SYSUT4, SYSUT5 (if the SYMDMP option is specified) and SYSUT6 (if LVL option is specified). At least one mass storage device, such as an IBM 3340 Direct Access Storage Facility Storage Drive, for residence of the operating system and SYSUT1. Both the operating system and SUSYT1 may reside on the same volume. The data sets SYSUT2, SYSUT3, SYSUT4, SYSUT5 and SYSUT6 can reside on tape or on mass storage. If they reside on tape, there must be a tape volume for each data set. If they reside on mass storage, there must be enough space on the volume to accommodate the data sets.
- A device, such as the 3215 Printer-Keyboard, for direct operator communication.
- A device, such as a card reader or a tape unit, for the job input stream.
- A printer or tape unit for the system output file.

COMPILER SIZE REQUIREMENTS

At least 128K (131,072) bytes should be allocated in the SIZE option of the EXEC job control card that requests execution of the compiler. If less than this is specified, the system assumes the default value of 128K.

In most instances, the compiler will perform adequately with a 128K SIZE. However, compiler efficiency usually increases with a larger SIZE allocation. This is because of the availability of larger buffers for compiler files, and/or the reduction or elimination of dictionary spills. Also, certain situations may make a larger SIZE specification not only preferable but necessary; for example:

- A large or complex source program. Compiling such a program requires more space for compiler internal tables.
- User-specified compiler SYSUT data set blocking factors that require large buffers (see Appendix D).

On the other hand, while a generous SIZE allocation is usually advantageous, it must not exceed the amount of contiguous problem program storage available to the compiler in the region or partition. The following calculation can be used to obtain a rough approximation of the maximum SIZE allocation possible:

```
SIZE = REGION - X - N
```

where

X = total sizes of any other programs in the region (zero if the compiler is alone)

N = 6K for VS2
6K + (SWA if used) + (any job-step overhead) for VS1

The variable N accounts for storage used or fragmentation caused by the operating system. Under VS1, if the compiler is not the first job step, earlier job steps may have caused additional storage to be used or fragmented by the system, and this storage may not be freed until job termination. (This indefinite loss is noted in the above calculation as 'job-step overhead.')

OS/VS2 and the Region Parameter

COMPILATION: If the compiler is being executed under OS/VS2, the REGION parameter, specified as 128K bytes in the COBUC and COBUCLG cataloged procedures, becomes significant (see the section "Using

the Cataloged Procedures"). If the programmer wishes to override this value, he can specify a region size in either the JOB statement or in the EXEC statement of the compiler. The size specified should not be less than the value of SIZE in the PARM field of the EXEC statement.

The following examples illustrate both the default and the override cases:

Example 1

```
//JOB1   JOB   1234,J.SMITH
//STEP1  EXEC  COBUC
          .
          .
          .
```

In this example, the programmer accepts the REGION default value of 128K specified in the COBUC cataloged procedure.

Example 2

```
//JOB2   JOB   1234,J.SMITH
//STEP1  EXEC  COBUCLG,REGION=196K,  X
//             PARM.COB='SIZE=196K'
          .
          .
          .
```

In this example, the REGION default value is overridden.

EXECUTION: Priority schedulers require that the REGION parameter be specified for execution of object programs, unless the programmer is willing to accept default region size. The default value is established in the input reader procedure. The region size needed for the execution of the object program is the sum of the following values:

1. The size of the object module after it has been link-edited with all of the necessary object time subroutines.
2. The size of the input/output buffers being used, multiplied by the blocking factor (physical sequential files are double buffered if no blocking factor is specified).
3. The size of the data management routines and control blocks that are used (see the publication OS/VS2 Storage Estimates).
4. Any GETMAIN macro instruction executed for USE LABELS, etc.
5. An additional 6K bytes.
6. If the Sort/Merge feature is used, 15,360 bytes plus any additional main storage assigned via the SORT-CORE-SIZE special register.

Intermediate Data Sets Under OS/VS2, Release 1

SYSIN and SYSOUT data resides in intermediate direct-access data sets. These data sets are used by the system to temporarily hold all of the job's input and output data. For SYSOUT, the programmer must use override statements as described in "Using the Cataloged Procedures."

Output is placed in the SYSOUT intermediate data set. Since nothing is written out until the completion of the job, the programmer must make sure that the SYSOUT data set is large enough to hold all of the possible output data of his program. The SPACE parameter of the DD statement is specified for SYSOUT with a specified default value. If the programmer determines that his output will exceed the default value, he can do either or both of two things:

1. Specify blocking of his data set with the DCB parameter of an override DD statement
2. Override the compilation step of a compiled procedure by specifying the SPACE parameter. An example of a statement that can be used is:

```
//COB.SYSPRINT  DD  SPACE=(121,(500,50)),
//     UNIT=SYSSQ
```

Note: If the TRK or CYL subparameters of the SPACE parameter are used, the programmer should be aware that requests will differ depending upon the mass storage device used (2314, 3330 ..., etc.). To avoid this consideration, the average record-length subparameter can be used.

EXECUTION TIME CONSIDERATIONS

The amount of main storage must be sufficient to accommodate at least:

- The control program
- Data management support
- The load module to be executed

When the OPTIMIZE option is specified, the number of procedure blocks in the program cannot exceed 255. A procedure block is approximately 4096 bytes of Procedure Division code.

COBOL programs compiled with any of the symbolic debugging options (STATE, FLOW, SYMDMP) have execution time requirements that differ from those of similar programs compiled without these options. If the SYMDMP option is in effect, the data set it required at compile time (SYSUT5) must be present at execution time.

The total space required for object-time debugging should be calculated as follows:

$$S_{TS} = S_{DBG} + \left\{ \begin{array}{l} S_{FLW} + \left[\begin{array}{l} S_{SYMDMP} \\ S_{STN} \end{array} \right] \\ S_{STN} + \left[\begin{array}{l} S \\ FLW \end{array} \right] \\ S_{SYMDMP} + \left[\begin{array}{l} S \\ FLW \end{array} \right] \end{array} \right\}$$

where:

S_{TS} = the total space

S_{DBG} = the space allocated once and only once for a run containing any object-time debugging options

S_{FLW} = the space required for the FLOW option

S_{STN} = the space required for the STATE option

S_{SYMDMP} = the space required for the SYMDMP option

- $S_{DBG} = 3700$ bytes

- $S_{FLW} = (1208 + 4*nn + 10*P)$ bytes

  where

  nn = the number specified in the FLOW=nn parameter of the EXEC job control statement

  P = the total number of paragraph- names in a COBOL program

- $S_{STN} = (872 + 5*V)$ bytes

  where

  V = the number of verbs in the COBOL program (a number that is approximately equal to the number of statements in the program)

- $S_{SYMDMP} = (11250 + S_{TABLES} + S_{DM})$ bytes

where

S_{TABLES} = the size of tables for SYMDMP

S_{DM} = the size of data management required for SYMDMP

$S_{TABLES} = (72*PC + [19*LC + [8*ON] + 7*id] + [S_{ODOTAB}])$ bytes

where

PC = the number of program control cards

LC = the number of line control cards

ON = the number of line control cards with ON options

id = the number of identifiers requested on line-control cards

S_{ODOTAB} = the size of ODOTAB on the debug file (approximately 27 times the number of unique objects of OCCURS DEPENDING ON statements).

$S_{DM} = (818 + S_{BSAM} + [S_{QSAM}])$ bytes

where

S_{BSAM} = 800 bytes = the space required for BSAM modules (when not in the LPA)

S_{QSAM} = 1424 bytes = the space required for QSAM modules (when not on the LPA) and no QSAM files are used in the program

The input/output device requirements for execution of the problem program are determined from specifications made in the Environment Division of the source program.

SORT/MERGE FEATURE CONSIDERATIONS

The basic requirements for use of the Sort/Merge feature are:

- A System/370 with sufficient main storage to accommodate the load module to be executed, plus a minimum of 32,000 bytes for execution of the sort/merge program, and any additional main storage assigned to the sort/merge program via the SORT-CORE-SIZE special register.
- At least one mass storage device (which may be the system residence device) for residence of SYS1.SORTLIB.
- At least three tape units or one mass storage device for intermediate storage.
- The OS/VS Sort/Merge program product, 5740-SM1. (If only the Sort feature is to be used--without alternate collating sequence and without Merge--then the OS Sort/Merge program product, 5734-SM1, would suffice as well.)

The following is a sample COBOL program and the output listing resulting from its compilation, linkage editing, and execution. The program creates a blocked, unlabeled, physical sequential file, writes it out on tape, and then reads it back in. It also does a check on the field called NO-OF-DEPENDENTS. All data records in the file are displayed. Those with a zero in the NO-OF-DEPENDENTS field are displayed with the special character Z. The records of the file are not altered from the time of creation, despite the fact that the NO-OF-DEPENDENTS field is changed for display purposes. The individual records of the file are created using the subscripting technique. TRACE is used as a debugging aid during program execution.

The output formats illustrated in the listing are described in "Output." Individual parts of the listing are numbered in accordance with the numbers used in the chapter "Output."

Note: This program contains a logic error that causes abnormal termination at execution time so that the use of program checkout facilities can be illustrated. See the chapter "Symbolic Debugging Features."

```
00001   10001U IDENTIFICATION DIVISION.
00002   10002U PROGRAM-ID.  TESTRUN.
00003   10003U   AUTHOR. PROGRAMMER NAME.
00004   10004U   INSTALLATION. NEW YORK PROGRAMMING CENTER.
00005   10005U   DATE-WRITTEN. JULY 12, 1968.
00006   10006U DATE-COMPILED. AUG  6,1976.
00007   10007U   REMARKS. THIS PROGRAM HAS BEEN WRITTEN AS A SAMPLE PROGRAM FOR
00008   10008U     COBOL USERS. IT CREATES AN OUTPUT FILE AND READS IT BACK AS
00009   10009U     INPUT.
00010
00011   10010U ENVIRONMENT DIVISION.
00012   10011U CONFIGURATION SECTION.
00013   10012U   SOURCE-COMPUTER. IBM-360-H50.
00014   10013U   OBJECT-COMPUTER. IBM-360-H50.
00015   10014U INPUT-OUTPUT SECTION.
00016   10015U FILE-CONTROL.
00017   10016U     SELECT FILE-1 ASSIGN TO UT-2400-S-SAMPLE.
00018   10017U     SELECT FILE-2 ASSIGN TO UT-2400-S-SAMPLE.
00019
00020   10018U DATA DIVISION.
00021   10019U FILE SECTION.
00022   10020U FD  FILE-1
00023   10021U     LABEL RECORDS ARE OMITTED
00024   10022U     BLOCK CONTAINS 100 CHARACTERS
00025   10022U     RECORD CONTAINS 20 CHARACTERS
00026   10023U     RECORDING MODE IS F
00027   10024U     DATA RECORD IS RECORD-1.
00028   10025U   01 RECORD-1.
00029   10026U     02 FIELD-A PICTURE IS X(20).
00030   10027U FD  FILE-2
00031   10028U     LABEL RECORDS ARE OMITTED
00032   10029U     BLOCK CONTAINS 5 RECORDS
00033   10030U     RECORD CONTAINS 20 CHARACTERS
00034   10031U     RECORDING MODE IS F
00035   10032U     DATA RECORD IS RECORD-2.
00036   10033U   01 RECORD-2.
00037   10034U     02 FIELD-A PICTURE IS X(20).
00038
00039   10035U WORKING-STORAGE SECTION.
00040   10036U     77 KOUNT PICTURE S99 COMP SYNC.
00041   10037U     77 NOMBER  PICTURE S99 COMP SYNC.
00042   10037U   01 FILLER.
00043   10038U     02 ALPHABET PICTURE X(26) VALUE "ABCDEFGHIJKLMNOPQRSTUVWXYZ".
00044   10039U     02 ALPHA REDEFINES ALPHABET PICTURE X OCCURS 26 TIMES.
00045   10040U     02 DEPENDENTS PICTURE X(26)  VALUE "01234012340123401234012340
00046   10041U-    "0".
00047   10042U     02 DEPEND REDEFINES DEPENDENTS PICTURE X OCCURS 26 TIMES.
00048   10044U   01 WORK-RECORD.
00049   10045U     02 NAME-FIELD  PICTURE X.
00050   10046U     02 FILLER PICTURE X VALUE IS SPACE.
00051   10047U     02 RECORD-NO PICTURE 9999.
00052   10048U     02 FILLER PICTURE X VALUE IS SPACE.
00053   10049U     02 LOCATION PICTURE AAA VALUE IS "NYC".
00054   10050U     02 FILLER PICTURE X VALUE IS SPACE.
```

```
00055   10051U      02 NO-OF-DEPENDENTS PICTURE XX.
00056   10052U      02 FILLER PICTURE X(7) VALUE IS SPACES.
00057   10052I      01 RECORDA.
00058   10052K         02  A PICTURE S9(4) VALUE 1234.
00059   10052S         02  B REDEFINES A PICTURE S9(7) COMPUTATIONAL-3.
00060   10053U PROCEDURE DIVISION.
00061   10054U BEGIN.
00062   10055U*              THE FOLLOWING OPENS THE OUTPUT FILE TO BE CREATED
00063   10056U*    AND INITIALIZES COUNTERS.
00064   10057U STEP-1. OPEN OUTPUT FILE-1. MOVE ZERO TO KOUNT NOMBER.
00065   10058U*              THE FOLLOWING CREATES INTERNALLY THE RECORDS TO BE
00066   10059U*    CONTAINED IN THE FILE, WRITES THEM ON TAPE, AND DISPLAYS
00067   10060U*    THEM ON THE CONSOLE.
00068   10061U STEP-2. ADD 1 TO KOUNT, ADD 1 TO NOMBER, MOVE ALPHA (KOUNT) TO
00069   10062U      NAME-FIELD.
00070   10063U      MOVE DEPEND (KOUNT) TO NO-OF-DEPENDENTS.
00071   10064U      MOVE NOMBER TO RECORD-NO.
00072   10065U STEP-3. DISPLAY WORK-RECORD UPON CONSOLE. WRITE RECORD-1 FROM
00073   10066U      WORK-RECORD.
00074   10067U STEP-4. PERFORM STEP-2 THRU STEP-3 UNTIL KOUNT IS EQUAL TO 26.
00075   10068U*              THE FOLLOWING CLOSES OUTPUT AND REOPENS IT AS
00076   10069U*    INPUT.
00077   10070U STEP-5. CLOSE FILE-1. OPEN INPUT FILE-2.
00078   10071U*              THE FOLLOWING READS BACK THE FILE AND SINGLES OUT
00079   10072U*    EMPLOYEES WITH NO DEPENDENTS.
00080   10073U STEP-6. READ FILE-2 RECORD INTO WORK-RECORD AT END GO TO STEP-8.
00081   10073I         COMPUTE B = B + 1.
00082   10074U STEP-7. IF NO-OF-DEPENDENTS IS EQUAL TO "0" MOVE "Z" TO
00083   10075U      NO-OF-DEPENDENTS. EXHIBIT NAMED WORK-RECORD. GO TO
00084   10076U      STEP-6.
00085   10077U STEP-8. CLOSE FILE-2.
00086   10078U      STOP RUN.
```

INTRNL NAME	LVL	SOURCE NAME
DNM=1-148	FD	FILE-1
DNM=1-168	01	RECORD-1
DNM=1-189	02	FIELD-A
DNM=1-206	FD	FILE-2
DNM=1-226	01	RECORD-2
DNM=1-247	02	FIELD-A
DNM=1-267	77	KOUNT
DNM=1-282	77	NOMBER
DNM=1-298	01	FILLER
DNM=1-312	02	ALPHABET
DNM=1-330	02	ALPHA
DNM=1-348	02	DEPENDENTS
DNM=1-368	02	DEPEND
DNM=1-384	01	WORK-RECORD
DNM=1-408	02	NAME-FIELD
DNM=1-428	02	FILLER
DNM=1-442	02	RECORD-NO
DNM=1-461	02	FILLER
DNM=1-475	02	LOCATION
DNM=1-493	02	FILLER
DNM=2-000	02	NO-OF-DEPENDENTS
DNM=2-026	02	FILLER
DNM=2-040	01	RECORDA
DNM=2-060	02	A
DNM=2-071	02	B

BASE	DISPL	INTRNL NAME	DEFINITION	USAGE	R	O	Q	M
DCB=01		DNM=1-148		QSAM				F
BL=1	000	DNM=1-168	DS 0CL20	GROUP				
BL=1	000	DNM=1-189	DS 20C	DISP				
DCB=02		DNM=1-206		QSAM				F
BL=2	000	DNM=1-226	DS 0CL20	GROUP				
BL=2	000	DNM=1-247	DS 20C	DISP				
BL=3	000	DNM=1-267	DS 1H	COMP				
BL=3	002	DNM=1-282	DS 1H	COMP				
BL=3	008	DNM=1-298	DS 0CL52	GROUP				
BL=3	008	DNM=1-312	DS 26C	DISP				
BL=3	008	DNM=1-330	DS 1C	DISP	R	O		
BL=3	022	DNM=1-348	DS 26C	DISP				
BL=3	022	DNM=1-368	DS 1C	DISP	R	O		
BL=3	040	DNM=1-384	DS 0CL20	GROUP				
BL=3	040	DNM=1-408	DS 1C	DISP				
BL=3	041	DNM=1-428	DS 1C	DISP				
BL=3	042	DNM=1-442	DS 4C	DISP-NM				
BL=3	046	DNM=1-461	DS 1C	DISP				
BL=3	047	DNM=1-475	DS 3C	DISP				
BL=3	04A	DNM=1-493	DS 1C	DISP				
BL=3	04B	DNM=2-000	DS 2C	DISP				
BL=3	04D	DNM=2-026	DS 7C	DISP				
BL=3	058	DNM=2-040	DS 0CL4	GROUP				
BL=3	058	DNM=2-060	DS 4C	DISP-NM				
BL=3	058	DNM=2-071	DS 4P	COMP-3	R			

MEMORY MAP

TGT	00318	
SAVE AREA	00318	
SWITCH	00360	
TALLY	00364	
SORT SAVE	00368	0
ENTRY-SAVE	0036C	
SORT CORE SIZE	00370	
RET CODE	00374	
SORT RET	00376	
WORKING CELLS	00378	
SORT FILE SIZE	004A8	
SORT MODE SIZE	004AC	
PGT-VN TBL	004B0	
TGT-VN TBL	004B4	
RESERVED	004B8	
LENGTH OF VN TBL	004BC	
LABEL RET	004BE	
RESERVED	004BF	
DBG R14SAVE	004C0	
COBOL INDICATOR	004C4	
A(INIT1)	004C8	
DEBUG TABLE PTR	004CC	
SUBCOM PTR	004D0	
SORT-MESSAGE	004D4	
SYSOUT DDNAME	004DC	
RESERVED	004DD	
COBOL ID	004DE	
COMPILED POINTER	004E0	
COUNT TABLE ADDRESS	004E4	
RESERVED	004E8	
DBG R11SAVE	004F0	
COUNT CHAIN ADDRESS	004F4	
PRBL1 CELL PTR	004F8	
RESERVED	004FC	
TA LENGTH	00501	
RESERVED	00504	
PCS LIT PTR	0050C	
DEBUGGING	00510	
CD FOR INITIAL INPUT	00514	
OVERFLOW CELLS	00518	
BL CELLS	00518	
DECBADR CELLS	00524	
FIB CELLS	00524	
TEMP STORAGE	00528	
TEMP STORAGE-2	00530	
TEMP STORAGE-3	00530	
TEMP STORAGE-4	00530	
BLL CELLS	00530	
VLC CELLS	00538	
SBL CELLS	00538	
INDEX CELLS	00538	
SUBADR CELLS	00538	
ONCTL CELLS	00540	
PFMCTL CELLS	00540	
PFMSAV CELLS	00540	
VN CELLS	00544	
SAVE AREA =2	0054C	
SAVE AREA =3	0054C	
XSASW CELLS	00554	
XSA CELLS	00554	
PARAM CELLS	00554	
RPTSAV AREA	00558	
CHECKPT CTR	00558	
DEBUG TABLE	00558	

```
61    *BEGIN
                0005E4                          PN=02  0 EQU   *
                0005E4                          START    EQU   *
                0005E4  58 B0 C 080                      L     11,080(0,12)          PBL=1
                0005E8  58 F0 C 024                      L     15,024(0,12)          V(ILBOFLW1)
                0005EC  05 1F                            BALR  1,15
                0005EE  00000003D                        DC    X'0000003D'
64    *STEP-1
                0005F2                          PN=03    EQU   *
                0005F2  58 F0 C 024                      L     15,024(0,12)          V(ILBOFLW1)
                0005F6  05 1F                            BALR  1,15
                0005F8  00000040                         DC    X'00000040'
64     OPEN     0005FC  58 F0 C 028                      L     15,028(0,12)          V(ILBODBG4)
                000600  05 EF                            BALR  14,15
                000602  58 10 C 040                      L     1,040(0,12)           DCB=1
                000606  58 40 1 024                      L     4,024(0,1)
                00060A  D2 02 4 011 C 02D                MVC   011(3,4),02D(12)                      V(ILBOEXT0)
                000610  50 10 D 234                      ST    1,234(0,13)           SA3=1
                000614  92 0F D 234                      MVI   234(13),X'0F'         SA3=1
                000618  96 80 D 234                      OI    234(13),X'80'         SA3=1
                00061C  41 10 D 234                      LA    1,234(0,13)           SA3=1
                000620  D2 03 D 060 C 057                MVC   060(4,13),057(12)     WC=01         LIT+7
                000626  58 F0 C 030                      L     15,030(0,12)          V(ILBOQI00)
                00062A  05 EF                            BALR  14,15
                00062C  58 10 C 040                      L     1,040(0,12)           DCB=1
                000630  D2 03 D 060 C 05B                MVC   060(4,13),05B(12)     WC=01         LIT+11
                000636  58 F0 C 030                      L     15,030(0,12)          V(ILBOQI00)
                00063A  05 EF                            BALR  14,15
                00063C  58 70 D 200                      L     7,200(0,13)           BL =1
64     MOVE     000640  D2 01 6 000 C 050                MVC   000(2,6),050(12)      DNM=1-267     LIT+0
                000646  D2 01 6 002 C 050                MVC   002(2,6),050(12)      DNM=1-282     LIT+0
68    *STEP-2
                00064C                          PN=04    EQU   *
                00064C  58 F0 C 024                      L     15,024(0,12)          V(ILBOFLW1)
                000650  05 1F                            BALR  1,15
                000652  00000044                         DC    X'00000044'
68     ADD      000656  48 30 C 052                      LH    3,052(0,12)           LIT+2
                00065A  4A 30 6 000                      AH    3,000(0,6)            DNM=1-267
                00065E  40 30 6 000                      STH   3,000(0,6)            DNM=1-267
68     ADD      000662  48 30 C 052                      LH    3,052(0,12)           LIT+2
                000666  4A 30 6 002                      AH    3,002(0,6)            DNM=1-282
                00066A  40 30 6 002                      STH   3,002(0,6)            DNM=1-282
68     MOVE     00066E  41 40 6 008                      LA    4,008(0,6)            DNM=1-330
                000672  48 30 6 000                      LH    3,000(0,6)            DNM=1-267
                000676  5C 20 C 050                      M     2,050(0,12)           LIT+0
                00067A  1A 43                            AR    4,3
                00067C  5B 40 C 050                      S     4,050(0,12)           LIT+0
                000680  50 40 D 220                      ST    4,220(0,13)           SBS=1
                000684  58 E0 D 220                      L     14,220(0,13)          SBS=1
                000688  D2 00 6 040 E 000                MVC   040(1,6),000(14)      DNM=1-408     DNM=1-330
7C     MOVE     00068E  41 40 6 022                      LA    4,022(0,6)            DNM=1-368
                000692  48 30 6 000                      LH    3,000(0,6)            DNM=1-267
                000696  5C 20 C 050                      M     2,050(0,12)           LIT+0
                00069A  1A 43                            AR    4,3
                00069C  5B 40 C 050                      S     4,050(0,12)           LIT+0
                0006A0  50 40 D 224                      ST    4,224(0,13)           SBS=2
                0006A4  58 F0 D 224                      L     15,224(0,13)          SBS=2
```

```
                 0006A8  D2 00 6 04B F 000          MVC   04B(1,6),000(15)     DNM=2-0          DNM=1-368
                 0006AE  92 40 6 04C                0 MVI   04C(6),X'40'       DNM=2-3+1
71      MOVE     0006B2  48 30 6 002                LH    3,002(0,6)          DNM=1-282
                 0006B6  4E 30 D 210                CVD   3,210(0,13)         TS=01
                 0006BA  F3 31 6 042 D 216          UNPK  042(4,6),216(2,13)  DNM=1-442        TS=07
                 0006C0  96 F0 6 045                OI    045(6),X'F0'        DNM=1-442+3
72      *STEP-3
                 0006C4                       PN=05 EQU   *
                 0006C4  58 F0 C 024                L     15,024(0,12)        V(ILBOFLW1)
                 0006C8  05 1F                      BALR  1,15
                 0006CA  00000048                   DC    X'00000048'
72      DISPLAY  0006CE  58 F0 C 028                L     15,028(0,12)        V(ILBODBG4)
                 0006D2  05 EF                      BALR  14,15
                 0006D4  58 F0 C 034                L     15,034(0,12)        V(ILBODSP0)
                 0006D8  05 1F                      BALR  1,15
                 0006DA  0002                       DC    X'0002'
                 0006DC  00                         DC    X'00'
                 0006DD  000014                     DC    X'000014'
                 0006E0  00000208                   DC    X'00000208'         BL =3
                 0006E4  0040                       DC    X'0040'
                 0006E6  FFFF                       DC    X'FFFF'
72      WRITE    0006E8  58 F0 C 028                L     15,028(0,12)        V(ILBODBG4)
                 0006EC  05 EF                      BALR  14,15
                 0006EE  D2 13 7 000 6 040          MVC   000(20,7),040(6)    DNM=1-168        DNM=1-384
                 0006F4  58 10 C 040                L     1,040(0,12)         DCB=1
                 0006F8  18 21                      LR    2,1
                 0006FA  92 00 2 07A                MVI   07A(2),X'00'
                 0006FE  58 40 2 024                L     4,024(0,2)
                 000702  92 00 4 014                MVI   014(4),X'00'
                 000706  96 01 4 018                OI    018(4),X'01'
                 00070A  58 10 C 040                L     1,040(0,12)         DCB=1
                 00070E  58 F0 C 030                L     15,030(0,12)        V(ILBOQIO0)
                 000712  D2 03 D 060 C 05F          MVC   060(4,13),05F(12)   WC=01           LIT+15
                 000718  58 00 1 04C                L     0,04C(0,1)
                 00071C  44 00 1 060                EX    0,060(0,1)
                 000720  58 20 C 040                L     2,040(0,12)         DCB=1
                 000724  91 40 2 07A                TM    07A(2),X'40'
                 000728  92 00 2 07A                MVI   07A(2),X'00'
                 00072C  47 10 B 154                BC    1,154(0,11)         GN=07
                 000730  50 10 D 200                ST    1,200(0,13)         BL =1
                 000734  58 70 D 200                L     7,200(0,13)         BL =1
                 000738                       GN=07 EQU   *
                 000738                       GN=01 EQU   *
                 000738  58 10 D 22C                L     1,22C(0,13)         VN=01
                 00073C  07 F1                      BCR   15,1
74      *STEP-4
                 00073E                       PN=06 EQU   *
                 00073E  58 F0 C 024                L     15,024(0,12)        V(ILBOFLW1)
                 000742  05 1F                      BALR  1,15
                 000744  0000004A                   DC    X'0000004A'
74      PERFORM  000748  D2 03 D 228 D 22C          MVC   228(4,13),22C(13)   PSV=1            VN=01
                 00074E  41 00 B 172                LA    0,172(0,11)         GN=02
                 000752  50 00 D 22C                ST    0,22C(0,13)         VN=01
                 000756                       GN=02 EQU   *
                 000756  48 30 6 000                LH    3,000(0,6)          DNM=1-267
                 00075A  49 30 C 054                CH    3,054(0,12)         LIT+4
                 00075E  47 80 B 182                BC    8,182(0,11)         GN=03
```

```
                000762  47 F0 B 068                 BC    15,068(0,11)          PN=04
                000766                      GN=03   EQU   *
                000766  D2 03 D 22C D 228           MVC   22C(4,13),228(13)     VN=01           PSV=1
77      *STEP-5
                00076C                      PN=07   EQU   *
                00076C  58 F0 C 024                 L     15,024(0,12)          V(ILBOFLW1)
                000770  05 1F                       BALR  1,15
                000772  00000004D                   DC    X'00000004D'
77      CLOSE   000776  58 F0 C 028                 L     15,028(0,12)          V(ILBODBG4)
                00077A  05 EF                       BALR  14,15
                00077C  58 10 C 040                 L     1,040(0,12)           DCB=1
                000780  91 10 1 030                 TM    030(1),X'10'
                000784  05 50                       BALR  5,0
                000786  47 80 5 01A                 BC    8,01A(0,5)
                00078A  D5 01 1 02A C 063           CLC   02A(2,1),063(12)                      LIT+19
                000790  47 70 5 01A                 BC    7,01A(0,5)
                000794  58 20 1 04C                 L     2,04C(0,1)
                000798  48 20 1 052                 SH    2,052(0,1)
                00079C  50 20 1 04C                 ST    2,04C(0,1)
                0007A0  58 10 C 040                 L     1,040(0,12)           DCB=1
                0007A4  50 10 D 234                 ST    1,234(0,13)           SA3=1
                0007A8  92 40 D 234                 MVI   234(13),X'40'         SA3=1
                0007AC  96 80 D 234                 OI    234(13),X'80'         SA3=1
                0007B0  41 10 D 234                 LA    1,234(0,13)           SA3=1
                0007B4  D2 03 D 060 C 065           MVC   060(4,13),065(12)     WC=01           LIT+21
                0007BA  58 F0 C 030                 L     15,030(0,12)          V(ILBOQIO0)
                0007BE  05 EF                       BALR  14,15
                0007C0  58 10 C 040                 L     1,040(0,12)           DCB=1
                0007C4  D2 03 D 060 C 05B           MVC   060(4,13),05B(12)     WC=01           LIT+11
                0007CA  58 F0 C 030                 L     15,030(0,12)          V(ILBOQIO0)
                0007CE  05 EF                       BALR  14,15
                0007D0  58 50 D 1B0                 L     5,1B0(0,13)
                0007D4  50 50 D 200                 ST    5,200(0,13)           BL =1
                0007D8  58 20 C 040                 L     2,040(0,12)           DCB=1
                0007DC  91 01 2 017                 TM    017(2),X'01'
                0007E0  47 10 B 21C                 BC    1,21C(0,11)           GN=08
                0007E4  58 10 2 014                 L     1,014(0,2)
                0007E8  96 01 2 017                 OI    017(2),X'01'
                0007EC  1B 44                       SR    4,4
                0007EE  43 40 1 005                 IC    4,005(0,1)
                0007F2  4C 40 1 006                 MH    4,006(0,1)
                0007F6  41 00 4 008                 LA    0,008(0,4)
                0007FA  41 10 1 000                 LA    1,000(0,1)
                0007FE  0A 0A                       SVC   10
                000800                      GN=08   EQU   *
77      OPEN    000800  58 F0 C 028                 L     15,028(0,12)          V(ILBODBG4)
                000804  05 EF                       BALR  14,15
                000806  58 10 C 044                 L     1,044(0,12)           DCB=2
                00080A  58 40 1 024                 L     4,024(0,1)
                00080E  D2 02 4 011 C 02D           MVC   011(3,4),02D(12)                      V(ILBOEXT0)
                000814  50 10 D 234                 ST    1,234(0,13)           SA3=1
                000818  92 00 D 234                 MVI   234(13),X'00'         SA3=1
                00081C  96 80 D 234                 OI    234(13),X'80'         SA3=1
                000820  41 10 D 234                 LA    1,234(0,13)           SA3=1
                000824  D2 03 D 060 C 057           MVC   060(4,13),057(12)     WC=01           LIT+7
                00082A  58 F0 C 030                 L     15,030(0,12)          V(ILBOQIO0)
                00082E  05 EF                       BALR  14,15
```

```
                  000830  58 10 C 044                    L       1,044(0,12)             DCB=2
                  000834  D2 03 D 060 C 058              MVC     060(4,13),058(12)       WC=01           LIT+11
                  00083A  58 F0 C 030                    L       15,030(0,12)            V(ILBOQI00)
                  00083E  05 EF                          BALR    14,15
                  000840  58 80 D 204                    L       8,204(0,13)             BL =2
80      *STEP-6
                  000844                         PN=08   EQU     *
                  000844  58 F0 C 024                    L       15,024(0,12)            V(ILBOFLW1)
                  000848  05 1F                          BALR    1,15
                  00084A  00000050                       DC      X'00000050'
80      READ      00084E  58 F0 C 028                    L       15,028(0,12)            V(ILBODBG4)
                  000852  05 EF                          BALR    14,15
                  000854  58 10 C 044                    L       1,044(0,12)             DCB=2
                  000858  18 21                          LR      2,1
                  00085A  D2 02 2 021 C 03D              MVC     021(3,2),03D(12)                        GN=04+1
                  000860  D2 03 D 060 C 069              MVC     060(4,13),069(12)       WC=01           LIT+25
                  000866  58 F0 C 030                    L       15,030(0,12)            V(ILBOQI00)
                  00086A  05 EF                          BALR    14,15
                  00086C  50 10 D 204                    ST      1,204(0,13)             BL =2
                  000870  58 80 D 204                    L       8,204(0,13)             BL =2
                  000874  D2 15 6 040 8 000              MVC     040(20,6),000(8)        DNM=1-384       DNM=1-226
                  00087A  47 F0 B 29E                    BC      15,29E(0,11)            GN=05
                  00087E                         GN=04   EQU     *
80      GO        00087E  47 F0 B 306                    BC      15,306(0,11)            PN=010
                  000882                         GN=05   EQU     *
81      COMPUTE   000882  F8 70 D 210 C 056              ZAP     210(8,13),056(1,12)     TS=01           LIT+6
                  000888  FA 43 D 213 6 058              AP      213(5,13),058(4,6)      TS=04           DNM=2-71
                  00088E  F8 33 6 058 D 214              ZAP     058(4,6),214(4,13)      DNM=2-71        TS=04+1
82      *STEP-7
                  000894                         PN=09   EQU     *
                  000894  58 F0 C 024                    L       15,024(0,12)            V(ILBOFLW1)
                  000898  05 1F                          BALR    1,15
                  00089A  00000052                       DC      X'00000052'
82      IF        00089E  95 F0 6 04B                    CLI     04B(6),X'F0'            DNM=2-0
                  0008A2  47 70 B 2D2                    BC      7,2D2(0,11)             GN=06
                  0008A6  95 40 6 04C                    CLI     04C(6),X'40'            DNM=2-0+1
                  0008AA  47 70 B 2D2                    BC      7,2D2(0,11)             GN=06
82      MOVE      0008AE  92 E9 6 04B                    MVI     04B(6),X'E9'            DNM=2-0
                  0008B2  92 40 6 04C                    MVI     04C(6),X'40'            DNM=2-0+1
                  00C8B6                         GN=06   EQU     *
83      EXHIBIT   0008B6  58 10 C 070                    L       1,070(0,12)             LIT+32
                  0008BA  50 10 D 23C                    ST      1,23C(0,13)             PRM=1
                  0008BE  41 20 D 23C                    LA      2,23C(0,13)             PRM=1
                  0008C2  58 F0 C 028                    L       15,028(0,12)            V(ILBODBG4)
                  0008C6  05 EF                          BALR    14,15
                  0C08C8  58 F0 C 034                    L       15,034(0,12)            V(ILBODSP0)
                  0008CC  05 1F                          BALR    1,15
                  0008CE  8001                           DC      X'8001'
                  0008D0  10                             DC      X'10'
                  0008D1  00000B                         DC      X'00000B'
                  0008D4  0C000074                       DC      X'0C000074'             LIT+36
                  0008D8  0000                           DC      X'0000'
                  0008DA  00                             DC      X'00'
                  0008DB  000014                         DC      X'000014'
                  0008DE  0D000208                       DC      X'0D000208'             BL =3
                  0008E2  0040                           DC      X'0040'
                  0008E4  FFFF                           DC      X'FFFF'
```

```
83       GO        0008E6  47 F0 B 260                    BC    15,260(0,11)          PN=08
85       *STEP-8                                                                0
                   0008EA                          PN=010  EQU   *
                   0008EA  58 F0 C 024                     L     15,024(0,12)          V(ILBOFLW1)
                   0008EE  05 1F                           BALR  1,15
                   0008F0  00000055                        DC    X'00000055'
85       CLOSE     0008F4  58 F0 C 028                     L     15,028(0,12)          V(ILBODBG4)
                   0008F8  05 EF                           BALR  14,15
                   0008FA  58 10 C 044                     L     1,044(0,12)           DCB=2
                   0008FE  91 10 1 030                     TM    030(1),X'10'
                   000902  05 50                           BALR  5,0
                   000904  47 80 5 01A                     BC    8,01A(0,5)
                   000908  D5 01 1 02A C 063               CLC   02A(2,1),063(12)                      LIT+19
                   00090E  47 70 5 01A                     BC    7,01A(0,5)
                   000912  58 20 1 04C                     L     2,04C(0,1)
                   000916  48 20 1 052                     SH    2,052(0,1)
                   00091A  50 20 1 04C                     ST    2,04C(0,1)
                   00091E  58 10 C 044                     L     1,044(0,12)           DCB=2
                   000922  50 10 D 234                     ST    1,234(0,13)           SA3=1
                   000926  92 40 D 234                     MVI   234(13),X'40'         SA3=1
                   00092A  96 80 D 234                     OI    234(13),X'80'         SA3=1
                   00092E  41 10 D 234                     LA    1,234(0,13)           SA3=1
                   000932  D2 03 D 060 C 065               MVC   060(4,13),065(12)     WC=01         LIT+21
                   000938  58 F0 C 030                     L     15,030(0,12)          V(ILBOQIO0)
                   00093C  05 EF                           BALR  14,15
                   00093E  58 10 C 044                     L     1,044(0,12)           DCB=2
                   000942  D2 03 D 060 C 05B               MVC   060(4,13),058(12)     WC=01         LIT+11
                   000948  58 F0 C 030                     L     15,030(0,12)          V(ILBOQIO0)
                   00094C  05 EF                           BALR  14,15
                   00094E  58 50 D 1B0                     L     5,1B0(0,13)
                   000952  50 50 D 204                     ST    5,204(0,13)           BL =2
                   000956  58 20 C 044                     L     2,044(0,12)           DCB=2
                   00095A  91 01 2 017                     TM    017(2),X'01'
                   00095E  47 10 B 39A                     BC    1,39A(0,11)           GN=09
                   000962  58 10 2 014                     L     1,014(0,2)
                   000966  96 01 2 017                     OI    017(2),X'01'
                   00096A  1B 44                           SR    4,4
                   00096C  43 40 1 005                     IC    4,005(0,1)
                   000970  4C 40 1 006                     MH    4,006(0,1)
                   000974  41 00 4 008                     LA    0,008(0,4)
                   000978  41 10 1 000                     LA    1,000(0,1)
                   00097C  0A 0A                           SVC   10
                   00097E                          GN=09   EQU   *
86       STOP      00097E  58 F0 C 028                     L     15,028(0,12)          V(ILBODBG4)
                   000982  05 EF                           BALR  14,15
                   000984                          GN=010  EQU   *
                   000984  58 F0 C 038                     L     15,038(0,12)          V(ILBOSRV1)
                   000988  07 FF                           BCR   15,15
                   00098A  50 D0 5 008             INIT2   ST    13,008(0,5)
                   00098E  50 50 D 004                     ST    5,004(0,13)
                   000992  50 E0 D 054                     ST    14,054(0,13)
                   000996  91 20 D 048                     TM    048(13),X'20'         SWT+0
                   00099A  47 E0 F 02E                     BC    14,02E(0,15)
                   00099E  58 20 D 1B8                     L     2,1B8(0,13)
                   0009A2  91 40 D 049                     TM    049(13),X'40'         SWT+1
                   0009A6  47 E0 F 02E                     BC    14,02E(0,15)
                   0009AA  96 04 2 000                     OI    000(2),X'04'
```

```
0009AE  58 F0 2 038                    L     15,038(0,2)
0009B2  41 F0 F 004                    LA    15,004(0,15)
0009B6  07 FF                          BCR   15,15
0009B8  94 EF D 048                    NI    048(13),X'EF'          SWT+0
0009BC  58 F0 C 010                    L     15,010(0,12)           VIR=1
0009C0  05 EF                          BALR  14,15
0009C2  12 00                          LTR   0,0
0009C4  07 89                          BCR   8,9
0009C6  96 10 D 048                    OI    048(13),X'10'          SWT+0
0009CA  58 F0 C 014            INIT3   L     15,014(0,12)           VIR=2
0009CE  05 EF                          BALR  14,15
0009D0  05 F0                          BALR  15,0
0009D2  91 20 D 048                    TM    048(13),X'20'          SWT+0
0009D6  47 E0 F 016                    BC    14,016(0,15)
0009DA  58 00 B 048                    L     0,048(0,11)
0009DE  98 2D B 050                    LM    2,13,050(11)
0009E2  58 E0 D 054                    L     14,054(0,13)
0009E6  07 FE                          BCR   15,14
0009E8  96 20 D 048                    OI    048(13),X'20'          SWT+0
0009EC  41 60 0 004                    LA    6,004(0,0)
0009F0  41 10 C 00C                    LA    1,00C(0,12)
0009F4  41 70 C 00F                    LA    7,00F(0,12)            VIR=1-1
0009F8  05 50                          BALR  5,0
0009FA  58 40 1 000                    L     4,000(0,1)
0009FE  1E 4B                          ALR   4,11
000A00  50 40 1 000                    ST    4,000(0,1)
000A04  87 16 5 000                    BXLE  1,6,000(5)
000A08  41 10 C 03C                    LA    1,03C(0,12)            PN=01
000A0C  41 70 C 047                    LA    7,047(0,12)            VNI=1-1
000A10  05 50                          BALR  5,0
000A12  58 40 1 000                    L     4,000(0,1)
000A16  1E 4B                          ALR   4,11
000A18  50 40 1 000                    ST    4,000(0,1)
000A1C  87 16 5 000                    BXLE  1,6,000(5)
000A20  41 60 0 008                    LA    6,008(0,0)
000A24  41 10 C 048                    LA    1,048(0,12)            VNI=1
000A28  41 70 C 04F                    LA    7,04F(0,12)            LIT+0-1
000A2C  05 50                          BALR  5,0
000A2E  58 40 1 000                    L     4,000(0,1)
000A32  1E 4B                          ALR   4,11
000A34  50 40 1 000                    ST    4,000(0,1)
000A38  87 16 5 000                    BXLE  1,6,000(5)
000A3C  41 80 D 200                    LA    8,200(0,13)            OVF=1
000A40  41 60 0 004                    LA    6,004(0,0)
000A44  41 70 D 20F                    LA    7,20F(0,13)            TS=01-1
000A48  05 10                          BALR  1,0
000A4A  58 00 8 000                    L     0,000(0,8)
000A4E  12 00                          LTR   0,0
000A50  47 80 1 010                    BC    8,010(0,1)
000A54  1E 0B                          ALR   0,11
000A56  50 00 8 000                    ST    0,000(0,8)
000A5A  87 86 1 000                    BXLE  8,6,000(1)
000A5E  58 60 D 208                    L     6,208(0,13)            BL =3
000A62  58 70 D 200                    L     7,200(0,13)            BL =1
000A66  58 80 D 204                    L     8,204(0,13)            BL =2
000A6A  D2 07 D 22C C 048              MVC   22C(8,13),048(12)      VN=01          VNI=1
000A70  58 E0 D 054                    L     14,054(0,13)
```

```
000A74  07 FE                          BCR   15,14
000000  90 EC D 00C          INIT1  0  STM   14,12,00C(13)
000004  18 5D                          LR    5,13
000006  05 F0                          BALR  15,0
000008  45 80 F 010                     BAL   8,010(0,15)
00000C  E3C5E2E3D9E4D540                DC    X'E3C5E2E3D9E4D540'
000014  E5E2D9F1                        DC    X'E5E2D9F1'
000018  07 00                          BCR   0,0
00001A  98 9F F 024                     LM    9,15,024(15)
00001E  07 FF                          BCR   15,15
000020  96 02 1 034                     OI    034(1),X'02'
000024  07 FE                          BCR   15,14
000026  41 F0 0 001                     LA    15,001(0,0)
00002A  07 FE                          BCR   15,14
00002C  00000CA                         ADCON L4(INIT3)
000030  00000000                        ADCON L4(INIT1)
000034  00000000                        ADCON L4(INIT1)
000038  00000560                        ADCON L4(PGT)
00003C  00000318                        ADCON L4(TGT)
000040  000005E4                        ADCON L4(START)
000044  0000098A                        ADCON L4(INIT2)
000048                                  DS    15F
000084  00000000                        DC    X'00000000'
000088  F1F54BF5F84BF0F5                DC    X'F1F54BF5F84BF0F5'
000090  C1E4C74040F66B40                DC    X'C1E4C74040F66B40'
000098  F1F9F7F6                        DC    X'F1F9F7F6'
```

```
*STATISTICS*      SOURCE RECORDS =    86      DATA DIVISION STATEMENTS =    25      PROCEDURE DIVISION STATEMENTS =    21
*OPTIONS IN EFFECT*      SIZE =  1M1072  BUF =   12288  LINECNT = 57  SPACE1, FLAGW,   SEQ,   SOURCE
*OPTIONS IN EFFECT*        DMAP,   PMAP, NOCLIST, NOSUPMAP, NOXREF,   SXREF,   LOAD, NODECK, QUOTE, NOTRUNC,   FLOW=   10
*OPTIONS IN EFFECT*      NOTERM, NONUM, NOBATCH, NONAME, COMPILE=01, NOSTATE, NORESIDENT, NODYNAM, NOLIB, NOSYNTAX
*OPTIONS IN EFFECT*        OPTIMIZE,   SYMDMP, NOTEST,   VERB,   ZWB, SYST, NOENDJOB, NOLVL
*OPTIONS IN EFFECT*      NOLST , NOFDECK,NOCDECK, LCOL2,  L120,   DUMP , NOADV , NOPRINT,
*OPTIONS IN EFFECT*      NOCOUNT, NOVBSUM, NOVBREF, LANGLVL(2)
*OPTIONS IN EFFECT*      DEBUG FILE SIZE = 2     BLOCKS, 1024   BYTES
```

CROSS-REFERENCE DICTIONARY

DATA NAMES	DEFN	REFERENCE			
A	000058				
ALPHA	000044	000068			
ALPHABET	000043				
B	000059	000081			
DEPEND	000047	000070			
DEPENDENTS	000045				
FIELD-A	000029				
FIELD-A	000037				
FILE-1	000017	000064	000072	000077	
FILE-2	000018	000077	000080	000085	
KOUNT	000040	000064	000068	000070	000074
LOCATION	000053				
NAME-FIELD	000049	000068			
NO-OF-DEPENDENTS	000055	000070	000082		
NUMBER	000041	000064	000068	000071	
RECORD-NO	000051	000071			
RECORD-1	000028	000072			
RECORD-2	000036	000080			
RECORDA	000057				
WORK-RECORD	000048	000072	000080	000083	

PROCEDURE NAMES	DEFN	REFERENCE
BEGIN	000061	
STEP-1	000064	
STEP-2	000068	000074
STEP-3	000072	000074
STEP-4	000074	
STEP-5	000077	
STEP-6	000080	000083
STEP-7	000082	
STEP-8	000085	000080

CARD	ERROR MESSAGE	
14	IKF1163I-W	IBM-370 IS ONLY VALID COMPUTER-NAME. IBM-360 SPECIFICATION IGNORED.
58	IKF2190I-W	PICTURE CLAUSE IS SIGNED, VALUE CLAUSE UNSIGNED. ASSUMED POSITIVE.

APPENDIX B: COBOL LIBRARY SUBROUTINES

COBOL library subroutines perform operations that require such extensive coding that it would be inefficient to place the coding in the object module each time it is needed.

COBOL library subroutines are stored in the COBOL library (SYS1.COBLIB). The required subroutines are inserted in load modules by the linkage editor.

There are several major categories of COBOL library subroutines, namely: subprogram linkage, object-time program operations (i.e., data conversions, arithmetic operations, test conditions, data manipulation, data management, and special features), and object-time debugging. The categories are described in this order.

Figure 174 later in this chapter includes a list of COBOL library subroutines, their storage requirements, and the associated calling information.

In addition, Q routines, which are not classified as COBOL library subroutines, are used to calculate the length of variable-length fields and the location of variably located fields resulting from an OCCURS clause with a DEPENDING ON option.

SUBROUTINES FOR SUBPROGRAM LINKAGE

The subroutines that control the loading of library subroutines or subprograms and the exiting from programs or subprograms are described here.

ENTER Subroutine (ILBONTR0)

The ILBONTR0 subroutine is used (1) when the RESIDENT option is an effect, to load one copy of each subroutine called by the main program or any of its subprograms into any region/partition; and (2) when the DYNAM option is in effect, to call any subprogram specified in a CALL literal or CALL identifier statement, first loading it if it has not already been loaded into that region/partition. (If insufficient storage is available for loading, the user's ON OVERFLOW imperative statement is called, if one was specified.)

When a program finishes execution, this routine deletes all the subroutines called by the program except those subroutines that are being used by another program in the region/partition. It also deletes any subprogram in the CANCEL literal or CANCEL identifier statement.

NORES Initialization Subroutine (ILBOBEG0)

The ILBOBEG0 subroutine performs initialization functions when the NORES option is in effect. If the Subroutine Communications Area (SUBCOM) has not been link edited, it loads the area and checks whether its calling program is the main program or a subprogram.

Object-Time Options Subroutine (ILBOPRM0)

This subroutine is invoked dynamically (by either ILBONTR0 or ILBOBEG0) to scan the user's object-time options and set internal switches and options accordingly.

STOP RUN Subroutine (ILBOSRV0)

The ILBOSRV subroutine is called by all programs compiled by the OS/VS COBOL compiler. (For pre-OS/VS compatibility an ILBOSTP0 entry point is also provided.) This routine returns control to the system, if the calling program is the main program, or to the caller, if it is not.

STOP RUN Messages Subroutine (ILBOMSG0)

The ILBOMSG0 subroutine determines whether or not a message is to be issued. If a message is to be issued this subroutine formats and issues it.

STOP RUN Termination Subroutine (ILBOSTT0)

The ILBOSTT0 subroutine performs termination functions at the end of execution of the COBOL program if the ENDJOB option is in effect.

OBJECT-TIME PROGRAM OPERATIONS

COBOL LIBRARY CONVERSION SUBROUTINES

Eight numeric data formats are permitted in COBOL -- five external (for input and output) and three internal (for internal processing).

The five external formats are these: (1) external or zoned decimal, (2) external floating-point, (3) sterling display, (4) numeric edited, and (5) sterling report. The three internal formats are these: (1) internal or packed decimal, (2) binary, and (3) internal floating-point.

The conversions from internal decimal to external decimal, from external decimal to internal decimal, and from internal decimal to numeric edited are done in-line. The other conversions are performed by the COBOL library subroutines shown in Figure 172, and by the separate sign subroutine.

Separate Sign Subroutine (ILBOSSN0)

The ILBOSSN0 subroutine converts separately signed data-names to internal decimal format and then checks for a valid sign. If the sign is valid, this subroutine generates the corresponding overpunch in the receiving field. If not, it causes an object time message to be issued and the job to be terminated.

Subroutine Name and Entry Points	Conversion	
	From	To
ILBOEFL2	External Floating-point	Internal Decimal
ILBOEFL1	External Floating-point	Binary
ILBOEFL0	External Floating-point	Internal Floating-point
ILBOBID0[1]	Binary	Internal Decimal
ILBOBID1[1]		
ILBOBID2[1]		
ILBOBIE0[1]	Binary	External Decimal
ILBOBIE1[1]		
ILBOBIE2[1]		
ILBOBII0[2]	Binary	Internal Floating-point
ILBOBII1[2]		
ILBOTEF0[2]	Binary	External Floating-point
ILBOTEF1[2]		
ILBOTEF2	Internal Decimal	External Floating-point
IFBOTEF3	Internal Floating-point	External Floating-point
ILBOIDB0	Internal Decimal	Binary
ILBOIDB1	External Decimal	Binary
ILBODCI1	Internal Decimal	Internal Floating-point
ILBODCI0	External Decimal	Internal Floating-point
ILBOIFD0	Internal Floating-point	Internal Decimal
ILBOIFD1	Internal Floating-point	External Decimal

[1]The entry points used depend on whether the double-precision number is in registers 0 and 1, or 2 and 3, or 4 and 5, respectively.
[2]The entry points are for single-precision binary and double-precision binary, respectively.
[3]This entry point is used for calls from other COBOL library subroutines.

Figure 172. Functions of COBOL Library Conversion Subroutines (Part 1 of 2)

Subroutine Name and Entry Points	Conversion	
	From	To
ILBOIFB1	Internal Floating-point	Binary integer and a power of 10 exponent
ILBOIFB2[3]		
ILBOIFB0[3]	Internal Floating-point	Binary
ILBOIDR0	Internal Decimal	Sterling Report
ILBOIDT0	Internal Decimal	Sterling Non-Report
ILBOSTI0	Sterling Non-Report	Internal Decimal
ILBOCVB0	External decimal	Binary
ILBOCVB1	External decimal	Binary

[1]The entry points used depend on whether the double-precision number is in registers 0 and 1, or 2 and 3, or 4 and 5, respectively.
[2]The entry points are for single-precision binary and double-precision binary, respectively.
[3]This entry point is used for calls from other COBOL library subroutines.

Figure 172. Functions of COBOL Library Conversion Subroutines (Part 2 of 2)

Subroutine Name	Function
ILBOXMU0	Internal Decimal Multiplication (30 digits * 30 digits = 60 digits)
ILBOXDI0	Internal Decimal Division (60 digits/30 digits = 60 digits)
ILBOXPR0	Exponentiation of an Internal Decimal Base by a Binary Exponent
ILBOFPW0	Floating-point Exponentiation
ILBOGPW0[1]	Floating-point Exponentiation

[1]The ILBOGPW0 entry point is used if the exponent has a picture specifying an integer. The ILBOFBW0 entry point is used in all other cases.

Figure 173. Function of COBOL Library Arithmetic Subroutines

COBOL LIBRARY ARITHMETIC SUBROUTINES

Most arithmetic operations are performed in-line. However, involved calculations, such as exponentiation, and calculations with very large numbers, such as decimal multiplication of two 30-digit numbers, are performed by COBOL library subroutines. These subroutine names and their functions are given in Figure 173.

COBOL LIBRARY SUBROUTINES FOR TESTING CONDITIONS AT OBJECT TIME

Several subroutines are used to test conditions that determine the path of control the object program selects. Such subroutines are described below.

Class Test Subroutine (ILBOCLS0)

The ILBOCLS0 subroutine is used to perform class tests for variable-length items and those fixed-length items over 256 bytes long, to determine whether a field is alphanumeric.

Note: The following tables are placed in the library for use by the in-line coding generated and the subroutines called for by both class test and TRANSFORM:

ILBOATB0 -- alphabetic class test
ILBOETB0 -- external decimal class test
ILBOITB0 -- internal decimal class test
ILBOTRN0 -- transformation
ILBOUTB0 -- unsigned internal decimal class test
ILBOWTB0 -- unsigned external decimal class test

COMPARE Subroutine (ILBOVC00)

The ILBOVC00 subroutine compares two operands, one or both of which are of variable lengths. They may exceed 256 bytes.

Compare with Figurative Constant Subroutine (ILBOIVL0)

The ILBOIVL0 subroutine compares the identifier to a figurative constant. The figurative constant must always be the second operand. If it is first in the source program, the operands are reversed and the condition code to be passed on is inverted before this subroutine is called.

COBOL LIBRARY DATA MANIPULATION SUBROUTINES

Subroutines are used to manipulate data in main storage in response to the MOVE, TRANSFORM, STRING, and UNSTRING statements. (Data manipulation in response to the EXAMINE statement is performed in-line by the object program.)

MOVE Subroutine (ILBOVM00 and ILBOVM01)

This subroutine is used to handle some MOVE statements. The subroutine is also used for READ and WRITE statements processed in conjunction with the SAME RECORD AREA clause. The subroutine has two entry points, depending on the type of move: ILBOVM00 (left-justified) and ILBOVM01 (right-justified).

MOVE Subroutine for System/370 (ILBOSMV0)

This special MOVE subroutine is used when the length of the receiving field is either greater than 512 bytes or variable. The subroutine transfers characters to a right-justified receiving field.

MOVE to Alphanumeric-Edited Field Subroutine (ILBOANE0)

The ILBOANE0 subroutine moves a data-name, literal, or figurative constant into a right- or left-justified alphanumeric edited field.

MOVE to Numeric-Edited Field Subroutine (ILBONED0)

The ILBONED0 subroutine is called by the UNSTRING subroutine to move characters from a packed decimal field into a numeric-edited receiving field.

MOVE Figurative Constant (ILBLOANF0)

The ILBLOANF subroutine moves a figurative constant of more than one character into a right- or left-justified nonnumeric receiving field.

TRANSFORM Subroutine (ILBOVTR0)

The ILBOVTR0 subroutine translates variable-length items.

STRING Subroutine (ILBOSTG0)

The ILBOSTG0 routine combines the partial or complete contents of two or more subfield(s) into a single field. This routine transfers characters from the sending item(s) to the receiving item in the same way that moves from alphanumeric item(s) to alphanumeric item(s) are effected.

UNSTRING Subroutine (ILBOUST0)

The ILBOUST0 routine separates continuous data in a sending field, placing it in multiple receiving fields.

INSPECT Subroutine (ILBOINS0)

This subroutine performs operations for the INSPECT statement, doing specified tallying and replacing.

COBOL LIBRARY DATA MANAGEMENT SUBROUTINES

COBOL library subroutines are called to process the following verbs: DISPLAY, TRACE, EXHIBIT, ACCEPT, START (when generic key is specified), READ (QSAM or BSAM), WRITE (QSAM or BSAM), CLOSE (QSAM or BSAM), OPEN (QSAM or BSAM), REWRITE (QSAM), RECEIVE (TCAM), and SEND (TCAM); library subroutines are also called for I/O errors, printer spacing, alternate collating sequence, and printer overflow.

DISPLAY, TRACE, and EXHIBIT Subroutine (ILBODSP0)

The ILBODSP0 subroutine is used to print, punch, or type data, usually in limited amounts, on an output unit. TRACE and EXHIBIT are kinds of DISPLAY.

The acceptable forms of data for this subroutine are:

1. Display
2. External decimal
3. Internal decimal (converted by the subroutine to external decimal)
4. Binary (converted by the subroutine to external decimal)
5. External floating-point

Internal floating-point numbers must be converted to external floating-point numbers before the subroutine is called.

Note: If the contents of a data-name are such that when converted they will exceed 18 decimal digits, the ILBODSP0 subroutine cannot process them and the results are unpredictable.

DISPLAY Subroutine (ILBODSS0)

The ILBODSS0 subroutine prints or types data of a certain kind on SYSPRINT or at the console. This subroutine is used instead of ILBODSP0 when there are no requests by the program for TRACE or EXHIBIT, and no variable-length or floating-point items; when there are no requests for display upon SYSPUNCH; and when neither the RESIDENT nor the DYNAM option is in effect.

ACCEPT Subroutine (ILBOACP0)

The ILBOACP0 subroutine is called to read from SYSIN or from the operator's console at execution time. For SYSIN, a logical record size of 80 is assumed. If the size of the data item being accepted is less than 80 characters, the data must appear as the first set of characters within the input record. If the size of the data item is greater than 80 characters, as many records as necessary are read until the storage area allocated to the data item is filled. If the data item is greater than 80 characters, but is

not an exact multiple of 80, the remainder of the last logical record is not accessible. For the console, a maximum of 114 characters are accepted and either 114 characters or the length of the item, whichever is smaller, is moved to the operand named in the ACCEPT statement.

Generic Key START Subroutine (ILBOSTR0)

The ILBOSTR0 subroutine is called when a USING KEY clause is coded with the START verb for ISAM files. The subroutine formats the search argument so that data management can get control to search for the generic key.

Checkpoint Subroutine (ILBOCKP0)

The ILBOCKP0 subroutine generates a checkpoint record, continuing the status of a program when a checkpoint is taken. This record is written on a checkpoint data set.

Wait Subroutine (ILBOWAT)

This subroutine allows its caller to wait a specified amount of time.

Error Intercept Subroutine (ILBOERR0)

The ILBOERR0 subroutine is used to test for various error conditions, and passes control to the interpretive-statement specified in the INVALID KEY option phrase or to the USE FOR ERROR declarative section depending on the type of error and error handling options specified. The entry points used for error processing by ILBOERR0 are:

ILBOERR1 Physical Sequential Files

ILBOERR2 Direct and Relative Files Accessed Sequentially
ILBOERR3 Indexed Files Accessed Sequentially
ILBOEER4 Direct and Relative Files Accessed Randomly
ILBOERR5 Indexed Files Accessed Randomly

Error Intercept Subroutine (ILBOSYN0)

The ILBOSYN0 subroutine performs, for version 4 and OS/VS, the same operations as the ILBOERR0 subroutine described above. The following list shows the entry point that is used for the various access methods:

Entry Point	Access Method
ILBOSYN1	QSAM
ILBOSYN2	BSAM
ILBOSYN3	BDAM
ILBOSYN4	QISAM
ILBOSYN5	BISAM

Label Handling Subroutine (ILBOLBL0)

The ILBOLBL0 subroutine is called for beginning-of-volume and beginning-of-file label processing or at end-of-file and end-of-volume.

Printer Overflow Subroutine (ILBOPTV0)

The ILBOPTV0 subroutine is used to control printer overflow testing and page ejection.

Printer Spacing Subroutine (ILBOSPA0)

The ILBOSPA0 subroutine is used to control printer spacing.

BSAM WRITE/CLOSE and BDAM OPEN Subroutine (ILBOSAM0)

The ILBOSAM0 routine processes input/output statements for direct or relative files accessed sequentially. It also handles OPEN statements and CLOSE statements with the REEL option for directly organized output files accessed randomly.

BSAM READ Subroutine (ILBOSPN0)

The BSAM read routine reads segments of a logical record and assembles them into a complete logical record. The routine is called by a compiler-generated READ code for a spanned record direct BSAM file.

QSAM I/O Subroutine (ILBOQIO)

This subroutine handles the various I/O requests for COBOL QSAM files.

DCB Exit Subroutine (ILBOEXT0)

The ILBOEXT0 subroutine is called as a DCB exit routine during an OPEN for a QSAM file to add the record format (RECFM) parameter options to the standard DCB.

VSAM Initialization Subroutine (ILBOINT0)

The ILBOINT0 subroutine is used to obtain virtual storage for the VSAM File Control Block (FCB) associated with each VSAM File Information Block (FIB).

VSAM Open and Close Subroutine (ILBOVOC0)

The ILBOVOC0 subroutine is used for all VSAM open and close requests.

VSAM Action Request Subroutine (ILBOVIO0)

The ILBOVIO0 subroutine is used for all START, READ, REWRITE, WRITE, and DELETE verbs that refer to VSAM files.

RECEIVE Subroutine (ILBOREC0)

For RECEIVE requests, the ILBOREC0 subroutine transfers a message, a message segment, or part of a message or message segment from the message control program to the COBOL application program. For ACCEPT MESSAGE COUNT requests, the subroutine returns the number of completed messages in the queue structure. This routine always updates the input communication description (CD) entry.

RECEIVE Initialization Subroutine (ILBORNT0)

The ILBORNT0 subroutine is included for pre-OS/VS compatibility purposes only. It builds the control block that communicates with the input queue associated with the cdname specified in the RECEIVE statement.

Queue Analyzer Object-Time Subroutine (ILBOSQA0)

The ILBOSQA0 subroutine is included for pre-OS/VS compatibility purposes only. It is called by the ILBOREC0 routine if the COBTPQD data set is present. This routine searches the COBTPQD data set for a member that corresponds to the name in the SYMBOLIC QUEUE field (defined in the COBOL source statements). If a match is found, the analyzer reads the member into main storage, using it to validate the SYMBOLIC SUB-QUEUE name(s) in the input CD of the COBOL source program. The analyzer also identifies the first valid DD name for the queue structure and gives this name to the ILBOREC0 routine.

Queue Structure Description Subroutine (ILBOQSU0)

The ILBOQSU0 subroutine creates a partitioned data set with one member for each queue structure defined in the COBOL-like source statements. This routine also generates a printed listing of the structure element, as well as of error messages, if any.

Message Count Subroutine (ILBOMSC)

This subroutine provides the caller with a count of the complete messages on a specified queue.

Queue Structure Scan (Communications) Subroutine (ILBOQSS)

This subroutine traverses the queue structure identified by the input CD and returns successive ddnames representing elementary subqueue names.

Job Scheduler Subroutine (ILBOSCD)

This subroutine schedules a specified job by issuing the operator START command.

ENABLE/DISABLE Subroutine (ILBONBL)

This subroutine modifies the message control tables in response to ENABLE and DISABLE statements.

Communications Job Scheduler Utility (ILBOCJS)

This utility schedules jobs when there are messages on TCAM queues for them to process.

Declarative Save Area Chaining Subroutine (ILBOCHN0)

The ILBOCHN0 subroutine acquires a dynamic save area for a Declarative entry condition; chains together the save areas created for the Error and Label Declaratives routines; releases save areas from the chain when the Declarative routine processing is complete.

GETCORE Subroutine (ILBOCMM0)

The ILBOCMM0 subroutine issues the GETMAIN and FREEMAIN macro instructions for the COBOL program or for any COBOL library subroutine requiring storage additions or deletions. The subroutine chains together the information about the storage areas acquired through the GETMAIN macro instruction and releases this information from the chain when the FREEMAIN macro instruction is issued for that area of storage.

SEND Subroutine (ILBOSND0)

The ILBOSND0 subroutine transfers a message, a message segment, or part of a message or message segment from the COBOL application program to the message control program. This routine always updates the output CD entry.

SEND Initialization Subroutine (ILBOSNT0)

The ILBOSNT0 subroutine (ILBOSNT0) subroutine is included for pre-OS/VS compatibility purposes only. It builds the control block that communicates with the output queue associated with the cdname specified in the SEND statement.

COBOL LIBRARY SUBROUTINES FOR SPECIAL FEATURES

Subroutines are used for some of the special features of COBOL:

- Sort/Merge feature
- Table handling feature (SEARCH statement)
- Segmentation feature (GO TO statement)
- 3886 Optical Character Reader
- ABEND request
- Alternate collating sequence

Also, a subroutine is called in response to the use of the following special registers: CURRENT-DATE, DATE, DAY, TIME, and TIME-OF-DAY.

Sort/Merge Feature Subroutine (ILBOSRT0)

The ILBOSRT0 subroutine acts as an interface between the COBOL calling program and the Sort/Merge program via the entry point name SORT.

Merge Subroutine (ILBOMRG0)

The ILBOMRG0 subroutine acts as an interface between the COBOL calling program and the Sort/Merge program for merge operations.

Sort Subroutine (ILBOSMG0)

The ILBOSMG0 subroutine is used by the Merge subroutine for sorting required by merge operations.

Sort Debug Subroutine (ILBOSDB0)

The ILBOSDB0 subroutine is used for sorting a debug data set.

Alternate Collating Sequence Compare Subroutine (ILBOACS)

This subroutine handles the various forms of non-numeric comparisons, using an alternate program collating sequence (if specified). It also handles "native" collating sequences.

SEARCH Subroutine (ILBOSCH0)

The ILBOSCH0 subroutine performs a binary search on a specified level of a table. It is used for the SEARCH ALL statement.

Segmentation Subroutine (ILBOSGM0)

The ILBOSGM0 subroutine is included for pre-OS/VS compatibility purposes only. It is used to load segments of a program that are not in main storage and to pass control from one segment to the other.

GO TO DEPENDING ON Subroutine (ILBOGDO0)

The ILBOGDO0 subroutine uses the value of a particular data-name as an index into a list of constants for each PN specified and then transfers control to the proper PN. If the value of the data-name is greater than the number of PN's specified, control returns to the next instruction after the calling sequence. ILBODGO0 also handles transfer of control between segments, and any necessary segment reinitialization.

Date-and-Time Subroutine (ILBODTE0)

This group of subroutines performs five functions in response to the use of the special registers CURRENT-DATE, DATE, DAY, TIME, and TIME-OF-DAY. The list below indicates the function of each of the entry points, and the format of each result in the receiving field of the specified MOVE or ACCEPT statement.

ILBODTE0 -- month/day/year

ILBODTE1 -- hour minute second

ILBODTE2 -- year month day

ILBODTE3 -- year day

ILBODTE4 -- hour minute second hundredth of a second

3886 Optical Character Reader Interface Subroutine (ILBOOCR0)

The ILBDOCR0 subroutine handles all input/output operations with the 3886 Optical Character Reader and builds the OCR File Control Block required for this purpose.

ABEND Request Subroutine (ILBOABN0)

The ILBOABN0 subroutine is used to process all ABEND requests.

OBJECT-TIME DEBUGGING

The options available for object-time debugging include: the statement number option (STATE), the flow trace option (FLOW), the count option (COUNT), the symbolic debugging option (SYMDMP), and the USE FOR DEBUGGING declarative. The subroutines for the first two options provide debugging information at abnormal termination of a program; the subroutines for the other options provide debugging information either at abnormal termination or dynamically during the execution of a program. All of these subroutines are under the control of and are supervised by the debug control subroutine ILBODBG0. The debug control subroutine is described first, followed by the subroutines that are called in response to the specification of the STATE, FLOW, SYMDMP, and COUNT options.

Debug Control Subroutine (ILBODBG0)

The ILBODBG0 subroutine is called once at entry point ILBODBG0 for each COBOL program for which any of the debugging options have been specified. This subroutine handles linkage and input/output for the STATE, FLOW, and SYMDMP options. It also produces the program name, the completion code, and the last PSW message at the time of the abnormal termination.

Use-for-Debugging Subroutine (ILBOBUG)

This subroutine handles invocation of USE FOR DEBUGGING declaratives, including filling in of the DEBUG-ITEM special register.

Flow Trace Subroutine (ILBOFLW0)

The ILBOFLW0 subroutine produces a formatted trace of the last "n" of COBOL procedures executed prior to an ABEND. It initializes, builds, and writes out the flow trace table.

Statement Number Subroutine (ILBOSTN0)

The ILBOSTN0 subroutine processes the STATE option and determines both the card number and the verb number for the last statement executed before the ABEND, and then generates a message containing this information.

Symbolic Dump Subroutine (ILBOD10 and ILBOD20)

The ILBOD10 subroutine is called when the SYMDMP option is in effect; this routine calls other modules as necessary for SYMDMP initialization. The ILBOD20 subroutine services SYMDMP output requests from DBG0. SYMDMP generates the following information as output on the SYSDBOUT data set: a copy of all SYMDMP control statements; diagnostic messages; dynamic dumps of user-selected data areas at strategic points during program execution; an abnormal termination statement number message; and the complete abnormal termination dump. In addition, modifications are made to the COBOL program in main storage if dynamic dumping is requested for the program.

Note: When SYMDMP services are requested for a job step, the sequence of events is, in general, as follows: (1) initialization -- for the first COBOL program in a job step, then for all other COBOL programs in that job step, and finally for independent program segments; (2) processing -- first for dynamic dump requests, and then for abnormal termination dumps.

SYMDMP Error Message Subroutine (ILBODBE0)

The ILBODBE0 subroutine is called by the PRINT routine of the debug control subroutine to format the appropriate error message in the SYSDBOUT output buffer.

For additional information on the FLOW, STATE, and SYMDMP options and their relationship to other COBOL options, see the chapter entitled "Symbolic Debugging Features" and the section "Options for the Compiler" in the chapter entitled "Job Control Procedures."

COUNT Initialization Subroutine (ILBOTC00)

The ILBOTC00 subroutine initializes the count common area, gets space for and initializes the count chain, and initializes the count chain pointer in the object module TGT.

COUNT Frequency Subroutine (ILBOCT10)

The ILBOCT10 subroutine updates the appropriate node counter by one and saves the caller's count-block number in the count chain.

COUNT Termination Subroutine (ILBOTC20)

The ILBOTC20 subroutine is called at termination of object module execution to determine if there are programs being monitored. If so, it calls subroutine ILBDTC30 to write execution statistics, and if the termination is normal, calls ILBDDBG8 to close the debug print file. If the termination is not normal, the debug print file is left open for debugging information.

COUNT Print Subroutine (ILBOTC30)

The ILBOTC30 subroutine computes and writes execution statistics on the debug print file upon termination of the program being monitored.

OBJECT-TIME DEBUGGING UNDER INFORMATION MANAGEMENT SYSTEM (PP5734-XX6, 5740-XX2)

SPIE Subroutine (ILBOSPI0)

In order to ensure correct debug on SYSDBOUT, the ILBOSPI0 subroutine is called by an explicit CALL statement written by the COBOL programmer in his source program to be compiled with the FLOW, STATE, SYMDMP, and/or COUNT options.

There should be one CALL statement written at the beginning of the Procedure Division and following each ENTRY statement in the program.

There should be one CALL statement written at each exit point in the program, i.e., preceding each GOBACK, EXIT PROGRAM, or STOP RUN statement.

These CALL statements are effective only in a COBOL program compiled with FLOW, STATE, SYMDMP, or COUNT options. They must be executed as a logical pair only once per COBOL run unit. If COBOL program A calls COBOL program B, either A or B or both can be compiled with debugging options, but only the highest level program compiled with debugging options should contain CALL 'ILBOSPI0' statements. The first execution of ILBOSPI0 issues a SPIE macro instruction to trap the old program PSW in the event of a program check before STAE gets control at abnormal termination. The second execution of ILBOSPI0 resets any previous SPIE at task normal termination. At abnormal termination, ILBODBG0 will reset the previous SPIE.

Finally, any CALL 'ILBOSPI0' statements written in a COBOL program compiled with none of the above options cause the subroutine to return control with no action (SPIE is not issued).

Note. When this facility is used, the PSW in a SYSUDUMP will point to the SVC 13 in ILBOSPI.

CALLING AND STORAGE INFORMATION

Figure 174 includes a list of COBOL library subroutines, their storage requirements, and the associated calling information. The subroutines are arranged alphabetically by the characters following 'ILBO'. The list includes subroutines that are called directly by the object program --primary subroutines--and the subroutines they call--secondary subroutines. Some subroutines (for example, ILBOANE) function as both primary and secondary subroutines.

The superscripts that accompany several of the entries refer to footnotes at the end of the table. Footnotes that appear with the names of subroutines indicate routines that are conditionally obtained, that are secondary subroutines only, or that may never reside in the OS/VS2 link pack area (LPA) or the OS/VS1 resident reenterable routine area (RRR). The footnotes that appear with some of the numeric values indicate whether the information represents a maximum value, a minimum value, or an estimated value. In all cases, the numeric values represent decimal bytes rounded off to the nearest 50.

For descriptions of the primary subroutines and of the major secondary subroutines, see the sections of this appendix entitled "Subroutines for Subprogram Linkage," "Object-Time Program Operations," and "Object-Time Debugging."

Primary Subroutine	Calling Information	Size*	Dynamic Work Area	Secondary Subroutines	Size	Dynamic Work Area	Re-entrant
ILBOABN (ABEND)	Called by compiled code	150		None			Yes
ILBOACP (ACCEPT)	Called by compiled code	582	100	ILBOCMM	1000		Yes
ILBOACS (Alternate compare)	Called by compiled code and ILBOUNS, ILBOSTG, ILBOSCH, and ILBOSMG	260					Yes
ILBOANE (MOVE alphanumeric-edited field)	Called by compiled code and by ILBOUST	328	0	None			Yes
ILBOANF (MOVE figurative constant)	Called by compiled code	120	0	None			Yes
ILBOATB (Alphabetic table for class test)	Used for ILBOCLS	272	0	None			Yes
ILBOBEG (NORES initialization)	Called by compiled code and ILBOSRV	296		None			Yes
ILBOBID (Binary to internal decimal)	Called by compiled code	136	0	None			Yes
ILBOBIE (Binary to external decimal)	Called by compiled code	136	0	None			Yes
ILBOBII (Binary to internal floating-point)	Called by compiled code and by ILBODCI, ILBOEFL	488	0	None			Yes
ILBOBUG (USE FOR DEBUGGING)	Called by compiled code	2060	560[2]	ILBOCHN ILBOCMM	450 1000		Yes
ILBOCHN (Save area chaining)	Called by ILBOSYN and ILBOLBL	432		ILBOCMM	1000		Yes
ILBOCJS (Job Scheduler)	Called by operating system			ILBOMSC ILBOSCD ILBOWAT	300 200 150	7220	No (also non-re-usable
ILBOCKP (Checkpoint)	Called by compiled code	74	0	None			Yes
ILBOCLS (Class test)	Called by compiled code	168	0				Yes

*Size given is an estimate.

Figure 174. Calling and Storage Information for COBOL Library Subroutines (Part 1 of 7)

Primary Subroutine	Calling Information	Size	Dynamic Work Area	Secondary Subroutines	Size	Dynamic Work Area	Re-entrant
ILBOCMM (GETCORE)	Called by compiled code, ILBOCHN, ILBOSRT, ILBOSNT, ILBORNT, ILBONTR, ILBOCVB, ILBOUST, ILBODBG, ILBODSP, ILBOACP, ILBOFLW, ILBOSTN, ILBOD10, ILBOD12, ILBOD21, ILBOREC, ILBOSND, ILBONBL, ILBOQSS, ILBOBUG, ILBOINS, and ILBOQIO	956		None			Yes
ILBOCOM[4 9] (Subroutine communications)	Link-edited or loaded by compiled code and by ILBOSRV; used by most COBOL library subroutines	410	0	None			No
ILBOCT1 (COUNT frequency)	Called by compiled code	224		None			Yes
ILBOCVB (Decimal to binary/binary to decimal)	Called by compiled code and by ILBOUST and ILBOSTG	1042	300[5]	ILBOCMM	1000	800	Yes
ILBODBG	Called by compiled code if FLOW, STATE, or SYMDMP is specified	3638	952 [7]				Yes
				ILBOCMM	950		
				ILBODBE[7 8 9]	1312	0	
				ILBOSTN[7 8 9]	776	96	
				ILBOFLW[7]	1096	600[3 7]	
				ILBOD01[7 8 9]	650	0	
ILBOD01[7 8 9]	Called by ILBODBG if SYSMDP is specified	728	0	ILBOD10[7 8 9]	2648	4000[7]	Yes
				ILBOD11[7 8 9]	775	0	
				ILBOD12[7 8 9]	1810	0	
				ILBOD13[7 8 9]	1576	0	
				ILBOD14[7 8 9]	1480	0	
				ILBOD20[7 8 9]	1122	0	
				ILBOD21[7 8 9]	1680	25/OD0[7]	
				ILBOD22[7 8 9]	2265	0	
				ILBOD23[7 8 9]	3882	0	
				ILBOD24[7 8 9]	2783	0	
				ILBOD25[7 8 9]	1222	0	
				ILBDD26[7 8 9]	2278	0	
ILBODCI (Decimal to internal floating-point)	Called by compiled code	240		ILBOIDB	150	0	Yes
ILBODCR0	Called by explicit call	560	0	None			Yes
ILBODSP (DISPLAY, TRACE, EXHIBIT)	Called by compiled code	3520	104	ILBOCMM	950		Yes

Figure 174. Calling and Storage Information for COBOL Library Subroutines (Part 2 of 7)

Primary Subroutine	Calling Information	Size	Dynamic Work Area	Secondary Subroutines	Size	Dynamic Work Area	Re-entrant
ILBODSS[4] (DISPLAY)	Called by compiled	1034	0	None			No
ILBODTE (Date, day, and time)	Called by compiled code	504	0	None			Yes
ILBOEFL (Conversion from external floating-point)	Called by compiled code	600	0	ILBOIOB ILBOBII	150 500	0 0	Yes
ILBOERR (Error intercept)	Called by the system	674	0	None			Yes
ILBOETB (External decimal table for class test)	Used by ILBOCLS	268	0	None			Yes
ILBOEXT (DCB exit)	Called by the system	80		None			Yes
ILBOFLW[7]	Called by ILBODBG and compiled code if FLOW is specified	1340	600[37]	ILBOCMM	1000		Yes
ILBOFPW (Floating-point exponentiation)	Called by compiled code	816	0	None			Yes
ILBOGDO (GO TO DEPENDING ON)	Called by compiled code	280	0	None			Yes
ILBOGPW (Floating-point exponentiation to a binary exponent)	Called by compiled code	96	0	None			Yes
ILBOIDB (Decimal to binary)	Called by compiled code or by ILBODCI	128	0	None			Yes
ILBOIDR (Internal decimal to sterling report)	Called by compiled code	1680	0	None			Yes
ILBOIDT (Internal decimal to sterling non-report)	Called by compiled code	696	0	None			Yes
ILBOIFB (Internal floating-point to decimal or binary)	Called by compiled code or by ILBOIFD or ILBOTEF	350	0	None			Yes
ILBOIFD (Internal floating to decimal or binary)	Called by compiled code	224	0	ILBOIFB			Yes

Figure 174. Calling and Storage Information for COBOL Library Subroutines (Part 3 of 7)

Primary Subroutine	Calling Information	Size	Dynamic Work Area	Secondary Subroutines	Size	Dynamic Work Area	Re-entrant
ILBOINS (INSPECT)	Called by compiled code	1730	820	ILBOACS ILBOCMM ILBOCVB	300 1000 1050		Yes
ILBOINT (VSAM initialization)	Called by compiled code	250		None			Yes
ILBOITB (Internal decimal table for class test)	Called by compiled code	280	0	None			Yes
ILBOIVL (Comparison with figurative constant)	Called by compiled code	80	0	None			Yes
ILBOLBL (Label handling)	Called by the system	480		ILBOCHN	450		Yes
ILBOMRG (Merge)	Called by ILBOSRT	830		None			Yes
ILBOMSC (Message count)	Called by ILBOCJS	260	7220	None			Yes
ILBOMSG (STOP RUN message)	Called by ILBOSRV	250		ILBODBG[7]	2850		Yes
ILBONBL (ENABLE/DISABLE)	Called by compiled code	2720	550	ILBOCMM ILBOQSS ILBOREC	1000 820 3250	4800[3]	Yes
ILBONTR (RES initialization)	Called by compiled code	3064	384[2]	ILBOPRM[7]	1130		Yes
ILBOOCR[11] (Optical character reader)	Called by compiled code	1402	0	None			No
ILBOPRM (Object-time parameters)	Called by ILBONTR or ILBOBEG	1130		ILBONSND	3300		Yes
ILBOPTV (Printer overflow)	Called by compiled code	152	0	None			Yes
ILBOQIO (QSAM I/O)	Called by compiled code	1390	168[2]	ILBOCMM ILBOSYN ILBOSRV[7] ILBOSPA[7]	100 1050 1000 1800		Yes
ILBOQSS (Queue scan)	Called by ILBOREC, ILBONBL	820	4800[3]	None			Yes
ILBOQSU[4,10] (Queue structure utility program)	Called by JCL	6290	4000	None			No (also non-reusable)

Figure 174. Calling and Storage Information for COBOL Library Subroutines (Part 4 of 7)

Primary Subroutine	Calling Information	Size	Dynamic Work Area	Secondary Subroutines	Size	Dynamic Work Area	Re-entrant
ILBOREC[10] (RECEIVE)	Called by compiled code, ILBOSND, and ILBONBL	3200	160 per queue block, 200 per buffer	ILBOCMM ILBOQSS[7]	950 830	PDS member SIZE	Yes
ILBOSAM (BSAM WRITE and CLOSE/BDAM OPEN)	Called by compiled code	1230	0	None			Yes
ILBOSCD (Scheduler starter)	Called by compiled code	190		None			Yes
ILBOSCH (SEARCH)	Called by compiled code	1022	0	None			Yes
ILBOSDB (Sort debug)	Called by ILBOSMG	1048		None			Yes
ILBOSGM (Segmentation)	Called by compiled code	510	0	ILBODBG	2000[1]	600[2]	Yes
ILBOSMG (Sort)	Called by compiled code	3270		ILBOMRG, ILBOACS	858 300		Yes
ILBOSMV (MOVE to right-justified field for System/370)	Called by compiled code	64	0	None			Yes
ILBOSND[10] (SEND)	Called by compiled code, ILBOPRM, AND ILBOSRV	3230	1140[2] and 200 per buffer	ILBOREC[7] ILBOCMM	3200 1000		Yes
ILBOSPA (Printer spacing)	Called by compiled code	1770	0	None			Yes
ILBOSPIO[11]	Called by explicit call	280	0	None			Yes
ILBOSRT (Sort)	Called by compiled code	1230	200	ILBOCMM ILBOMRG	1000 850		Yes
ILBOSRV (STOP RUN)	Called by a program compiled by the COBOL or other compiler	990	0	ILBOBEG ILBOMSG ILBOSTT[7] ILBOSND[7]	300 250 400 3300		No
ILBOSSN (Separately signed numeric	Called by compiled code	232[1]	0	ILBOSRV ILBODBG	1000 2000	0 600[2]	Yes
ILBOSTG (STRING)	Called by compiled code	680	0	ILBOCVB ILBOACS	1050 300	300	Yes
ILBOSTI (Sterling non-report to internal decimal)	Called by compiled code	600	0	None			Yes

Figure 174. Calling and Storage Information for COBOL Library Subroutines (Part 5 of 7)

Primary Subroutine	Calling Information	Size	Dynamic Work Area	Secondary Subroutines	Size	Dynamic Work Area	Re-entrant
ILBOSTN[7,8,9] (Statement number option)	Called by ILBODBG if STATE is specified	790	96	ILBOCMM	1000		Yes
ILBOSTR (START with generic key)	Called by compiled code	96	0	None			Yes
ILBOSTT (STOP RUN termination)	Called by ILBOSRV	380		ILBONTR	2900		Yes
ILBOSYN (Error intercept)	Called by the system	1010		ILBOCHN	450		Yes
ILBOTC0 (COUNT initialization	Called by ILBODBG	800		ILBODBG1	2850		Yes
ILBOTC2 (COUNT termination)	Called by compiled code, ILBODBG, ILBOABX, and ILBOSRV	950		ILBOTC3 ILBODBG	4600 2850		Yes
ILBOTC3 (COUNT print)	Called by ILBOTC2	4600		ILBODBG1 ILBODBG8	2850 2850		Yes
ILBOTEF (Conversion to external floating-point)	Called by compiled code or by ILBOD23	688	0	ILBOBIE	150	0	Yes
ILBOTRN (TRANSFORM table)	Used by ILBOVTR	272	0	None			Yes
ILBOUST (UNSTRING)	Called by compiled code	2100	250[5]	ILBONED[7,8] ILBOCMM ILBOANE[7,8] ILBOCVB[7]	1400 1000 350 1050	 0 	Yes
ILBOUTB (Unsigned internal decimal table for class test)	Called by compiled code	144	0	None			Yes
ILBOVCO (Variable-length comparison)	Called by compiled code	520	0	None			Yes
ILBOVIO (VSAM action request)	Called by compiled code	7000		ILBOCKP ILBOCHN	500 100		Yes

Figure 174. Calling and Storage Information for COBOL Library Subroutines (Part 6 of 7)

Primary Subroutine	Calling Information	Size	Dynamic Work Area	Secondary Subroutines	Size	Dynamic Work Area	Re-entrant
ILBOVMO (Variable-length name)	Called by compiled code	560	0	ILBOSRV ILBODBG[7]	300 2000[1]	0 600	Yes
ILBOVOC (VSAM open and close)	Called by compiled code	8000		ILBOCKP ILBOCHN	500 100		Yes
ILBOVTR (TRANSFORM)	Called by compiled code	144	0	None			Yes
ILBOWAT (Wait)	Called by ILBOCJS	140		None			Yes
ILBOWTB (Unsigned external decimal table for class test)	Used by ILBOCLS	272	0	None			Yes
ILBOXDI (Decimal division)	Called by compiled code and by ILBOXPR	280	0	None			Yes
ILBOXMU (Decimal multiplication)	Called by compiled code and by ILBOXPR	192	0	None			Yes
ILBOXPR (Decimal fixed-point exponentiation)	Called by compiled code	680	0	ILBOXDI	300	0	Yes

Notes:

1. The size given is an estimate.
2. The size given is a minimum.
3. The size given is a maximum.
4. The subroutine indicated may never reside in the OS/VS2 link pack area (LPA) or the OS/VS1 resident reusable routine area (RRR).
5. The 256-byte storage area obtained by subroutine ILBOCVB is used by subroutine ILBOUST.
6. Because the ILBDODBG; subroutine dynamically loads and deletes subroutines as they are needed, depending on the options specified, it is possible only to estimate a minimum and/or a maximum amount of storage used by any one of the debugging options. For each storage estimate given below, the effect of possible core fragmentation is not considered.
   a. Basic debug package -- 3768 bytes
   b. Debug with the STATE option -- 4640 bytes
   c. Debug with the FLOW option -- 5464 bytes
   d. Debug with SYMDMP option -- 14,000 bytes minimum and 20,000 bytes maximum.
7. The subroutine or dynamic work area indicated as obtained conditionally.
8. The subroutine indicated is never called as a primary subroutine.
9. The subroutine indicated must be on-line at execution time.
10. The subroutine indicated may require SYS1.TELCMLIB to be in-line at execution time.
11. The subroutine indicated is called explicitly by CALL statements in the COBOL program.

Figure 174. Calling and Storage Information for COBOL Library Subroutines (Part 7 of 7)

APPENDIX C: FIELDS OF THE DATA CONTROL BLOCK

In this appendix, each field of the data control block is listed by the name of the operand of the assembler-language macro instruction that can specify a value for that field. Figures 175 through 179 illustrate the data control blocks for sequential, direct, relative, and indexed files. Some of the data control block fields can be referred to with the DCB parameter of the DD statement. However, any field filled in by the COBOL compiler cannot be overridden except for the indexed file OPTCD field in which the L-subparameter is set by the compiler using DCB exit.

Values for fields for which no entry appears in the column headed "COBOL Source" may be supplied by the DD statement or by the data set label.

For information concerning the specification of values for data control block fields, see the DCB macro instruction for the different file processing techniques in the publication OS/VS Data Management Macro Instructions.

Note: The DCB subparameters are discussed under "User Defined Files" in the chapter "User File Processing."

Data Control Block Field	Explanation of Field	COBOL Source	Applicable DD Statement DCB Subparameters
BFALN	Alignment	(COBOL specifies double-word boundary)	
BFTEK	Buffering technique (S or E)	(COBOL specifies S)	
BLKSIZE	Maximum length of block	BLOCK CONTAINS Data record description	BLKSIZE
BUFCB	Address of buffer pool	SAME AREA	
BUFL	Length of each buffer		
BUFNO	Number of buffers assigned to DCB	RESERVE	BUFNO=N(default=2)
BUFOFF			(BUFOFF=[n|l])
DDNAME	Name of DD statement	ASSIGN clause	
DSORG	Access method	ASSIGN clause ACCESS clause	
EODAD	Address of user's end-of-data-set exit routine for input data set	READ...AT END	
EROPT (see note)	Error option		(EROPT=[ACC|SKP|ABE])
EXLST	Address of exit list	Used by the compiler for USE...LABEL, etc.	
LRECL	Logical record length	FD entry	LRECL
MACRF	Type of macro instruction	OPEN INPUT, READ OPEN OUTPUT, WRITE OPEN I-O, READ, WRITE REWRITE	
OPTCD	Optional service provided by control program	CODE-SET	(OPTCD=[W|C|WC|T|Q])
RECFM	Characteristics of records in data set	RECORDING MODE Record description ADVANCING POSITIONING BLOCK CONTAINS APPLY RECORD-OVERFLOW	(RECFM=D)
SYNAD	Address of error exit routine	Used by compiler for INVALID KEY and USE AFTER ERROR	RECFM={S|T}

Note: If the COBOL program contains FILE STATUS or USE AFTER ERROR/EXCEPTION clauses for the QSAM file, EROPT=ACC should be specified; otherwise, the COBOL program will never receive control when certain abend situations arise.

Figure 175. Data Control Block Fields for Physical Sequential Files (QSAM)

Data Control Block Field	Explanation of Field	COBOL Source	Applicable DD Statement DCB Subparameters
BLKSIZE	Maximum length of block	Data record description	
DDNAME	Name of DD statement	ASSIGN clause	
DSORG	Access method	ASSIGN clause ACCESS clause	
EODAD	Address of end-of-data-set exit (input)	READ...AT END	
EXLST	Address of exit list	USE...LABEL PROCEDURE	
KEYLEN	Length of key	ACTUAL KEY[1] (length of ACTUAL KEY - 4)	
LRECL	Logical record length	FD entry	LRECL
MACRF	Type of macro instruction	OPEN INPUT, READ OPEN OUTPUT, WRITE (DIRECT ONLY)	
OPTCD	Optional service to be provided by control program		[OPTCD=W|T]
RECFM	Characteristics of records in data set	RECORDING MODE Record description APPLY RECORD-OVERFLOW	
SYNAD	Address of error exit routine	USE AFTER ERROR INVALID KEY	

[1]Direct files only; for relative files, the field is 0.

Figure 176. Data Control Block Fields for Direct and Relative Files Accessed Sequentially (BSAM)

Data Control Block Field	Explanation of Field	COBOL Source	Applicable DD Statement DCB Subparameters
BLKSIZE	Maximum length of block	Data record description	
DDNAME	Name of DD statement	ASSIGN clause	
DSORG	Access method	ASSIGN clause ACCESS clause	
EXLST	Address of exit list	USE...LABEL, etc.	
KEYLEN	Length of key for each physical record	ACTUAL KEY[1] (length of ACTUAL KEY - 4)	
LIMCT	Search limits		LIMCT=n (OPTCD=E must be specified)
MACRF	Type of macro instruction	OPEN INPUT, READ OPEN OUTPUT, WRITE (DIRECT ONLY) OPEN I-O, READ, WRITE (DIRECT ONLY), REWRITE	
OPTCD	Option service to be provided by the control program		OPTCD=E/W
RECFM	Characteristics of records of data set	RECORDING MODE APPLY RECORD-OVERFLOW Record description	
SYNAD	Address of error exit routine	Used by compiler for INVALID KEY and USE AFTER ERROR	

[1]Direct files only, for relative files this field is 0.

Figure 177. Data Control Block Fields for Direct and Relative Files Accessed Randomly (BDAM)

Data Control Block Field	Explanation of Field	COBOL Source	Applicable DD Statement DCB Subparameters
BFALN	Buffer alignment (F or D)	(COBOL specifies D)	
BKLSIZE	Maximum length of block	BLOCK CONTAINS	BLKSIZE
BUFCB	Address of buffer pool	SAME AREA	
BUFNO	Number of buffers assigned to DCB	RESERVE	BUFNO=N(default=2)
CYLOFL	Number of overflow tracks for each cylinder		CYLOFL=XX
DDNAME	Name of DD statement	ASSIGN clause	
DSORG	Access method	ACCESS clause ASSIGN clause	
EODAD	Address of user's end-of-data-set exit routine for input data set	READ...AT END	
EXLST	Address of exit list	Used by the compiler	
KEYLEN	Length of key for each logical record	RECORD KEY	
LRECL	Logical record length	FD entry	LRECL
MACRF	Type of macro instruction	OPEN INPUT, READ, START OPEN OUTPUT, WRITE OPEN I-O, READ, START, REWRITE	
NTM	Maximum number of cylinder index tracks		NTM=XX
OPTCD	Optional services		OPTCD=I|R|W|Y|M|U|L (must also have NTM=M)
RECFM	Characteristics of records in data set	RECORDING MODE RECORD DESCRIPTION BLOCK CONTAINS	
RKP	Relative position of record key in logical record	RECORD KEY	
SYNAD	Address of error exit routine	Used by the compiler for INVALID KEY, USE AFTER ERROR	

Figure 178. Data Control Block Fields for Indexed Sequential Files Accessed Sequentially (QISAM)

Data Control Block Field	Explanation of Field	COBOL Source	Applicable DD Statement DCB Subparameters
BFALN	Buffer alignment (F or D).	(COBOL specifies D)	
DDNAME	Name of DD statement.	ASSIGN clause	
DSORG	Access method.	ACCESS clause ASSIGN clause	
EXLST	Address of exit list.	Used by the compiler	
KEYLEN	Key length.	NOMINAL KEY	
LRECL	Logical record length.	FD entry	
MACRF	Type of macro instruction.	OPEN INPUT, READ OPEN I-O, READ, WRITE, REWRITE,	
MSHI	Address of area for highest level index of data set.	APPLY CORE-INDEX	
MSWA	Address of area reserved for control program. Required for variable length records.	TRACK-AREA	
SMSI	Size for area provided for highest level index of the data set.	APPLY CORE-INDEX	
SMSW	Number of bytes reserved for main storage work area.	TRACK-AREA	

Figure 179. Data Control Block Fields for Indexed Sequential Files Accessed Randomly (BISAM)

APPENDIX D: COMPILER OPTIMIZATION

In general, compilation is faster when:

1. Options in the EXEC statement are specified to:
   a. Make more main storage available (the SIZE option and the JCL REGION parameter)
   b. Optimize the space available for buffers (the BUF option)
   c. Suppress output (the NOSOURCE, NODECK, NOLOAD, and the SUPMAP options, among others)
   d. Suppress object code if one or more E-level messages are generated (CSYNTAX option).
2. The maximum block size for a compiler data set is specified.
3. A disk configuration and separate channels for utility data sets are used.
4. Separate devices (i.e., not the same mass storage unit) on the same channel are used.

Compilation time is also affected by the speed of the devices allocated to the data sets. For example, a tape device is faster than a printer for printed output. The blocking information that follows applies to OS/VS1 or OS/VS2.

PERFORMANCE CONSIDERATIONS

The OS/VS COBOL Compiler, provides additional opportunities for saving either main storage or time. For example, specification of the Optimized Code Feature, the COBOL Library Management Feature, the Dynamic Subprogram Feature, or all three of these features, can result in a considerable saving in main storage. The notes given below provide additional performance information on programs run with these and other new features.

- When the Optimized Code Feature is requested, via the OPTIMIZE compiler option, execution time is reduced for non-I/O bound programs; however, compilation time is increased.
- Specification of the COBOL Library Management Facility, via the RESIDENT compiler option, results in a saving of both main storage and secondary storage, as well as of time at the link-edit step and the initial program load for the program.
- Dynamic invocation and release of COBOL subprograms, specified by the DYNAM compiler option, also results in savings in main storage.
- A syntax checking compilation, specified by the SYNTAX or CSYNTAX compiler option, saves machine time. Depending on which compiler options are chosen, as well as the various source program statements, compile time can be reduced greatly.

The symbolic dump feature, specified by the SYMDMP option, and source-level debugging through USE FOR DEBUGGING declaratives, can save much debugging time. However, use of either of these features can decrease performance expectations for programs run with it. That is, such programs require additional time for the compile, link-edit, and execute job steps. They also require more main storage than programs run without this feature.

For information about requesting any of these options,see the section "Options for the Compiler" in the chapter on "Job Control Procedures". For information about USE FOR DEBUGGING, see the chapter "Program Checkout."

BLOCK SIZE FOR COMPILER DATA SETS

The blocking factor specified for compiler data sets other than utility data sets must be permissible for the device the data set is on. In addition, for the SYSLIN data set, it must be permissible for the linkage editor used. (Any block size specified for a utility data set in a DD statement is overridden by the compiler.) If a block size other than the default option is needed, it can be requested by specifying the BLKSIZE subparameter of the DCB parameter in the DD statement for the data sets. The format of the subparamete is:

```
DCB=(BLKSIZE=nnn)
```

where nnn is equal to N times the logical record size in bytes, and $1 \leq N \leq M$. M is equal to the blocking factor permissible for the device, and, in the case of SYSLIN, to the blocking factor permissible for the linkage editor used.

If blocking is desired, the record format for SYSPRINT [DCB=(RECFM=nnn)] should be specified as FBA. The record format for SYSIN, SYSLIN, SYSPUNCH, and SYSLIB should be specified as FB.

Note: For queued sequential data sets, the RECFM subparameter of the DD statement may optionally be specified at object time, permitting the programmer to specify the standard block option (for data sets with recording mode F) or the track overflow option for the data set. (The track overflow option is equivalent to writing an APPLY RECORD-OVERFLOW clause in the source program.) Use of the standard block option (particularly for direct-access devices having the Rotational Positional Sensing feature) results in the significant I/O performance improvement.

Fixed-block single volume data sets as created by COBOL are standard (except possibly when extended using the DISP=MOD parameter of the DD statement). Multivolume data sets as created by COBOL are standard if the volume switching occurs through automatic end-of-volume procedures. If, however, the programmer issues a CLOSE REEL/UNIT statement, then he must ensure that the number of logical records in the volume is an integral multiple of n, where a BLOCK CONTAINS n RECORDS clause (or an equivalent BLOCK CONTAINS CHARACTERS clause) has been specified in the source program. The standard block option and the track overflow option are mutually exclusive.

The logical record size for SYSPRINT and SYSUT6 is 121 bytes. The logical record size for SYSIN, SYSLIN, SYSPUNCH, and SYSLIB is 80 bytes.

Note: For compile, link-edit, and execute cases when labeled volumes are used, RECFM and BLKSIZE must be given for SYSLIN in the compile step only. If BLKSIZE is specified for SYSPUNCH, LRECL must also be specified.

HOW BUFFER SPACE IS ALLOCATED TO BUFFERS

Once the amount of space available for a compilation is determined, the compiler subtracts the amount required for itself. From the space remaining, it then computes the space available for utility and input/output data set buffers. If space still remains, the compiler makes use of it for internal processing.

Once the amount of space available for buffers is determined, the compiler calculates how this space is to be divided. First, it computes the amount of space required for the buffers of the input/output data sets. From the space remaining, it determines the maximum buffer size, and hence block size, possible for a utility data set. The four required utility data sets SYSUT1 through SYSUT4 all have the same block size. Thus, the block size of a utility data set is dependent on the amount of space available for buffers. If a block size has been specified in a DD statement for a utility data set, it is overridden.

A larger buffer size for a utility data set allows for faster processing. However, if the program being compiled takes up a large amount of the available storage, a smaller space for buffers enables the compiler to use more main storage for internal processing.

The following describes how the space available for buffers is determined and how it is allocated to buffers.

Let A represent the total space that can be allocated to these buffers. It is determined as follows:

1. If neither the BUF nor the SIZE option of the PARM parameter of the EXEC statement is specified, A equals the default value for buffer space. This value is specified at system generation time. The minimum value is 4096 bytes.

2. If the SIZE option is specified, but BUF is not, A equals (SIZE - 96K) / 4 plus the default value for buffer space.

3. If BUF is specified (whether or not SIZE is specified), A equals the value specified for BUF.

   Note: The minimum difference between SIZE and BUF must always be equal to or greater than the difference between the minimum SIZE value and the minimum BUF value (131,072 bytes - 12,288 bytes).

4. If BUF is smaller than 4096 a warning message is printed and the minimum value is assumed. If BUF is too large to allow minimum table space for compilation, a warning message is printed and the default value (or the

minimum value, if the default value is also too large) is assumed.

The programmer must make sure that the amount of buffer space allocated by the system is sufficient, taking into consideration the block sizes specified for the compiler data sets. The allocated buffer space is divided as follows:

1. Let B represent the amount of buffer space to be allocated for input/output data sets. B is computed as either equal to:

   2 times the block size of SYSPRINT + SYSIN + SYSLIB

   or

   2 times the block size of SYSPRINT + SYSPUNCH + SYSLIN

   whichever is larger. The maximum allowable value of B is A - 1280 bytes (1280 bytes less than the total buffer size). If the computed value is greater than the maximum allowable value, a diagnostic message is printed and compilation is abandoned.

   Note: When the BATCH option is in effect, an additional SYSIN buffer is required. The first formula above then becomes:

   2 times the block size of SYSPRINT + SYSIN + SYSIN + SYSLIB

   If the block sizes are not specified in the DD statements, the following default values are assumed:

Data Set	Default Value (bytes)
SYSIN	80
SYSLIN	80
SYSPUNCH	80
SYSLIB	80
SYSPRINT/SYSUT6	121 or 133
SYSTERM	121
SYSUT5	512

Notes: The default for SYSPRINT/SYSUT6 is 133 if the L132 option is in effect. The 512-byte block size for SYSUT5 cannot be overridden.

2. Let C represent the amount of buffer space to be allocated for each utility data set. Therefore, C equals the block size of data sets, SYSUT1, SYSUT2, SYSUT3, and SYSUT4, respectively.

   $$\text{If } A \leq 6B, \text{ then } C = \frac{A - B}{5}$$

   $$\text{If } A > 6B, \text{ then } C = \frac{A}{6}$$

   If C > maximum block size permitted for any device a utility data set is on, then the maximum block size is the value chosen for C. The minimum block size for SYSUT1, SYSUT2, SYSUT3, and SYSUT4 is 256 bytes.

APPENDIX E: INVOCATION OF THE COBOL COMPILER AND COBOL COMPILED PROGRAMS

The COBOL compiler can be invoked by a problem program at execution time through the use of the ATTACH or the LINK macro instruction, i.e., dynamic invocation. Dynamic invocation of COBOL compiled programs can be accomplished through the use of the LINK, ATTACH, or LOAD macro instruction.

INVOKING THE COBOL COMPILER

The problem program must supply the following information to the COBOL compiler:

- The options to be specified for the compilation
- The ddnames of the data sets to be used during processing by the COBOL compiler
- The header to appear on each page of the listing

```
┌──────────┬─────────┬──────────────────────────┐
│Name      │Operation│Operand                   │
├──────────┼─────────┼──────────────────────────┤
│[symbol]  │ LINK    │EP=IKFCBL00,              │
│          │ ATTACH  │PARAM=(optionlist         │
│          │         │  [,ddnamelist],          │
│          │         │  [,headerlist]),VL=1     │
└──────────┴─────────┴──────────────────────────┘
```

where:

EP
: specifies the symbolic name of the COBOL compiler. The entry point at which execution is to begin is determined by the control program (from the library directory entry).

PARAM
: specifies, as a sublist, address parameters to be passed from the problem program to the COBOL compiler. The first fullword in the address parameter list contains the address of the COBOL option list. The second fullword contains the address of ddname list. If standard ddnames are to be used and no header list is specified, this list may be omitted. If standard ddnames are to be used and a header list is specified, this entry should contain the address of a halfword of binary zeros, aligned on a halfword. The last fullword contains the address of the header list. This list may be omitted.

option list
: specifies the address of a variable length list containing the COBOL options specified for compilation. For additional details, see the description of the EXEC statement in the chapter "Job Control Procedures." This address must be written even though no list is provided.

  The option list must begin on a halfword boundary. The two high-order bytes contain a count of the number of bytes in the remainder of the list. If no options are specified, the count must be zero. The option list is free form with each field separated from the next by a comma. No blanks or zeros should appear in the list.

ddname list
: specifies the address of a variable length list containing alternative ddnames for the data sets used during COBOL compiler processing. If standard ddnames are used, this operand may be omitted.

  The ddname list must begin on a halfword boundary. The two high-order bytes contain a count of the number of bytes in the remainder of the list. Each name of less than eight bytes must be left justified and padded with blanks. If an alternate ddname is omitted from the list, the standard name will be assumed. If the name is omitted within the list, the 8-byte entry must contain binary zeros. Names can be omitted from the end merely by shortening the list.

  All utility data sets passed to the compiler must be physical sequential (for example, DSORG=PS must be their type of organization).

The sequence of the 8-byte entries in the ddname list is as follows:

ddname 8-byte Entry	Name for Which Substituted
1	SYSLIN
2	not applicable
3	not applicable
4	SYSLIB
5	SYSIN
6	SYSPRINT
7	SYSPUNCH
8	SYSUT1
9	SYSUT2
10	SYSUT3

11	SYSUT4
12	SYSTERM
13	SYSUT5
14	SYSUT6

header list
: specifies the address of a variable-length list containing information to be included in the heading on each page of the listing. The list must begin on a halfword boundary. The two high-order bytes should contain a 4 (the count of the number of bytes in the new heading information). The next four bytes of the list should contain the page number at which the heading is to start, in EBCDIC format.

VL
: specifies that the sign bit is to be set to 1 in the last fullword of the address parameter list.

When the COBOL compiler completes processing, a return code is placed in register 15. For additional details, see the discussion of the COND parameter in the chapter "Job Control Procedures."

INVOKING COBOL COMPILED PROGRAMS

Linkage editor control cards should be specified as follows:

1. For the PROGRAM-ID program-name, a NAME card.
2. For each ENTRY literal-1, an ALIAS card should be specified in a COBOL program that is to be dynamically invoked.

APPENDIX F: SOURCE PROGRAM SIZE CONSIDERATIONS

This appendix contains information to aid the programmer in determining how his source program affects usage of space at compilation time and linkage editing time.

COMPILER CAPACITY

The capacity of the COBOL compiler is limited by two general conditions: (1) the total space available must be sufficient for compilation and (2) an individual table may not have a length greater than 32,767 bytes, with the exception of the ADCON and cross-reference tables. If either of these conditions is not met during compilation, one of the following error messages will be issued:

```
IKF0001I-D SIZE PARAMETER TOO SMALL FOR
           THIS PROGRAM.

IKF0010I-D A TABLE HAS EXCEEDED THE MAXIMUM
           PERMISSIBLE SIZE.
```

In either case, compilation is terminated. However, in the first case, the program may be recompiled with a larger SIZE parameter. The size of the ADCON and cross-reference tables is not limited to 32,767 bytes.

If a table overflows, the following error message will be generated, and the user will need to rerun the program in a larger region.

```
IKF6007I-D TABLE OVERFLOW.  PMAP LOAD
           MODULE OR DECK WILL BE
           INCOMPLETE.  INCREASE SIZE
           PARAMETER.
```

Minimum Configuration SOURCE PROGRAM Size

The compiler will accept and compile a 2000 card program in its minimum main storage allocation (128K). Of course, the various reader procedures may affect the value required for SIZE and BUF parameters. The compiler will allocate the minimum required amounts that are 256 bytes for each of the 4 intermediate files, 80 bytes for each system file with the exception of SYSPRINT for which 121 or 133 bytes are allocated. Double buffering will be assumed.

EFFECTIVE STORAGE CONSIDERATIONS

The amount of main storage within the compiler's partition and the limitation on the size of an individual internal table are two factors that limit the capacity of the compiler. The limitation on the size of internal tables can, in some instances, be overcome by the spilling over of some tables onto external devices. However, spilling over may cause a severe degradation of performance. The main storage limitation should not be reached by any reasonable use of the language. However, within a limited storage capacity excessive use of certain features and combination of features in the language could make compilation impossible. Some of the features that significantly affect storage usage are the following:

1. Text Punch Table

   Each entry occupies 8 bytes. This table is not limited to the maximum size of 32,767 bytes. Entries are based on the:

   - Number of 4096-byte segments in the Working-Storage Section
   - Number of 4096-byte segments in a file buffer area
   - Number of referenced procedure-names
   - Number of implicit procedure-name references such as those generated by IF, SEARCH, and GENERATE statements, ON SIZE ERROR, ON OVERFLOW, INVALID KEY, and AT END options, the OCCURS clause with the DEPENDING ON option, USE sentences, and the Segmentation feature.
   - Number of files

2. Procedure-name Table

   This table contains the number of definitions written in a section and unresolved procedure references. Procedure references are resolved at the end of a section if the definition of the procedure-name is in that section or a preceding section. Therefore, forward references beyond a section impact space. Approximately 900 unqualified entries are possible. A maximum number of 16,255 entries may be specified.

3. OCCURS DEPENDING ON Table

   This table contains an entry for each unique object of an OCCURS clause with the DEPENDING ON option. The size of an entry is 2 + length of name + length of each qualifier bytes.

4. Index Table

   An entry is made for each INDEXED BY clause consisting of 11 bytes for each index.

5. File Table

   An entry is made for each file specified in the program. Each entry occupies 60 bytes of storage.

6. Report Writer Tables

   A considerable amount of information is maintained for each RD such as controls, sums, headings, footings, routines to be generated, and so on. The contents of the table are increased by qualification and subscripting in the Report Section. Approximately 30 reports can be processed without exceeding the limit of the table.

7. Dictionary Table

   An entry is made for each procedure-name and each data-name in the program. A procedure entry consists of (7 or 9 + length of name) bytes. A data entry consists of (length of name + n) bytes, where n is determined by the attributes of the data item. Some of the features that contribute to the value n are:

   - One byte for each character in a numeric edited or alphanumeric edited item picture
   - Five bytes for an elementary item with a Sterling Report picture clause
   - Three bytes for an item subordinate to an OCCURS clause

8. Literal Tables

   The total length of all literals may not exceed 32511 bytes. No more than 16255 literals may be specified.

9. Miscellaneous Tables

   The presence of the following items causes entries to be made into tables that affect the total space required for compilation.

   - SAME [RECORD] AREA clause
   - Subscripting
   - Intermediate Arithmetic Results
   - Complex Arithmetic Expressions
   - Complex Logical Expressions
   - APPLY clauses
   - Special-Names
   - RERUN clauses
   - Error messages
   - XREF
   - Segmentation feature
   - USE FOR DEBUGGING

LINKAGE EDITOR CAPACITY

Some COBOL program and linkage editor considerations are listed below as a further guide in preparing a source program. Consult the publication OS/VS Linkage Editor and Loader, for additional information on linkage editor capacities and processing.

1. All COBOL object programs consist of a single CSECT (control section). The size of the object module may be determined by looking at the location of the last instruction in INIT3 in the object code listing (see the section entitled "Output") or from the END card.

2. The size of the object module is greatly increased by any of the following:

   a. The blocking factor and alternate area reservation of randomly accessed files

   b. The specification of the SAME AREA clause for sequentially accessed files

3. RLD (Relocation List Dictionary) cards are part of the load module, and are used by the linkage editor to compute the address constants for the load module. The number of RLDs produced by the compiler can be determined by the following formula:

   number of RLDs = number of unique subprograms called + number of COBOL library routines called

4. The output text of the compiler is written out in a sequence that differs from the order indicated by the location counters contained in each output item. This sequence difference may result in a strain on the facilities of the linkage editor.

5. VALUE clauses in the Working-Storage Section may result in many discontinuous text records.

6. The object module produced by the COBOL compiler will not be in ascending address by card order prior to the linkage editor step.

APPENDIX G: INPUT/OUTPUT ERROR CONDITIONS

This appendix consists of two sections, each describing detailed information available to the programmer after an input/output error:

Section 1 is a sample listing of the possible contents of certain fields within data-name-1 of the error declaratives GIVING clause. This listing is presented as a guide for the programmer in analyzing his own program at the COBOL source level.

Section 2 describes pertinent input/output error conditions according to access method, including guidance in the use of the INVALID KEY and error declarative features.

Note: More detailed information for use in diagnosing error conditions may be found in OS/VS Data Management Macro Instructions and in either OS/VS1 System Data Areas, Order No. SY28-0605 or OS/VS2 System Data Areas, Order No. ST68-0606 (Release 1) or SYB8-0606 (Release 2 or later).

Section 1

Figure 180 is a sample listing of a constant area within one of the modules used in SYNADAF processing to formulate the descriptive error message ultimately reported to the programmer in data-name-1 of the error declarative GIVING option. Part 1 of the table is used to fill in the 6-byte field referred to as OPERATION-ATTEMPTED, bytes 85-90 of data-name-1. Part 2 is the source of the 6-byte field known as ACCESS-METHOD. Bytes 108-128 contain this field, the exact six bytes depending on the device type -- unit record, magnetic tape, mass storage.

Note: COBOL supports only QSAM, BSAM, BDAM, BISAM, QISAM and VSAM at the source level.

Part 3 of the table provides the contents of the most important field, the 15-byte ERROR-DESCRIPTION, bytes 92-106. Parts 1 and 3 are arranged by access method, in general, and a number of entries in one part are repeated in the other.

The programmer may use this listing as a guide to determining the nature of his error within the declarative by examining the appropriate fields. By analyzing these fields with the aid of Section 2 of this appendix, the programmer can determine what are the consequences of the error and what can be done about it.

Important: First, if either the OPERATION-ATTEMPTED or ERROR-DESCRIPTION field specifies "unknown" in some way, the contents of data-name-2 are probably invalid, as described in "Error Processing for COBOL Files" earlier in this publication. Referencing data-name-2 in this situation should not be done since it may cause an abend.

Second, the module which contains data similar to Figure 180 is part of the operating system and, as such, is subject to possible modification with each new release -- including insertion or deletion of fields, modification of descriptions, and so forth. In order to ascertain the contents of this constant area in use at a particular installation, the programmer may consult the microfiche listing of the modules.

The following modules issue SYNADAF messages in bytes 92 to 106 of the message buffer:

IGC0106H
IGC0206H
IGC0306H
IGC0406H
IGC0606H
IGC0706H
IGC0806H
IGC0906H

The contents of actual messages issued can be found by consulting the microfiche for these modules.

If microfiche listings are not available, the sample job in Figure 181 may be executed to get a dump of the appropriate module; the fields described above may then be extracted from this dump.

```
*              SYNAD ANALYZE AND FORMAT SVC                  *      0140001
*                     INITIAL LOAD                           *      0180001
DMLIST1  DC C'READ  '            FILL IN FOR TYPE OF OPERATION  6720001
         DC C'WRITE '                                           6740001
         DC C'CNT-OP'                                           6760001
         DC C'POINT '            POINT OPERATION                6780001
         DC C'UNKN'               UNKNOWN OPERATION             6800001
DMLIST2  DC C'GET   '                                           6820001
         DC C'PUT   '                                           6840001
         DC C'UNKN'                                             6860001
         DC C'UNKN'                                             6880001
         DC C'SETL'                                             6900001
*    ACCESS METHOD TYPES                                        6920001
QSAMS    DC C'QSAM '                                            6940001
BSAMS    DC C'BSAM '                                            6960001
BPAMS    DC C'BPAM '                                            6980001
BDAMS    DC C'BDAM '                                            7000001
BISAMS   DC C'BISAM'                                            7020001
QISAMS   DC C'QISAM'                                            7040001
BTAMS    DC C'BTAM '                                            7060001
QTAMS    DC C'QTAM '                                            7080001
GAMS     DC C'GAM  '                                            7100001
*          FOR EXCP, BPAM, BSAM, QSAM, AND BDAM                 0330001
ERRORS   DC CL15' CHAN CTL CK'   ERROR MESSAGES                 7320001
         DC CL15'INTF CTL CK'                                   7350001
         DC CL15'PROG CHECK'                                    7380001
         DC CL15'PROT CHECK'                                    7410001
         DC CL15'CHAIN CHECK'                                   7440001
         DC CL15'UNKNOWN COND.'                                 7470001
         DC CL15'END OF FILE'                                   7500001
         DC CL15'WRNG.LEN.RECORD'                               7530001
UNKNOWN  DC CL15'UNKNOWN COND.'                                 7560001
         SPACE 1                                                8130001
FCBLST   DC CL15'NO ERROR STATUS'                               8160001
EXTENT   DC CL15'OUT OF EXTENT'                                 8190001
         DC CL15'PREV D.E. ERROR'                               8220001
PURGED   DC CL15'PURGED REQUEST'                                8250001
         DC CL15'R.HA.R0 - ERROR'                               8280001
*                  FOR BDAM AND BISAM                           0330001
ERRLIST  DC CL15'RECORD NOT FND '                               5490001
         DC CL15'RECD LGTH CHECK'                               5520001
         DC CL15'NO SPACE AVAILB'                               5550001
         DC CL15'INVALID REQUEST'                               5580001
         DC CL15'UNCOR I/O ERROR'                               5610001
         DC CL15'UNREACHBLE BLCK'                               5640001
         DC CL15'OVERFLOW RECORD'                               5670001
         DC CL15'DUPLICATE RECRD'                               5700001
ERROR1   DC CL15'NOT IN SRCH.LMT'                               6300001
         DC CL15'WRNG.LEN.RECORD'                               6330001
         DC CL15'SPACE NOT FOUND'                               6360001
         DC CL15'INVALID REQUEST'                               6390001
         DC CL15'  I/O ERROR   '                                6420001
         DC CL15'  END OF DATA '                                6450001
         SC CL15' ERROR NOT I/O '                               6480001
         DC CL15'EXCLUSIVE CNTRL'                               6510001
ERROR2   DC CL15'  INPUT DCB    '                               6570001
         DC CL15'LIMCT=0-EX.SRCH'                               6600001
         DC CL15' OUT OF EXTENT '                               6630001
         DC CL15'CAPACITY RECORD'                               6660001
```

Figure 180. Sample Constant Area Used in SYNADAF Processing (Part 1 of 3)

```
         DC CL15' INCORRECT KEY '                                    6690001
         DC CL15'INVALID OPTIONS'                                    6720001
         DC CL15'FIX.LEN.KEY''F'''                                   6750001
         DC CL15' UNKNOWN ERROR '                                    6780001
*                    SECONDARY LOAD                          *       0360001
*                     FOR QISAM                                      0440001
NOTAPI   DC CL14'NOT APPLICABLE'                                     7280001
UNKNOWN  DC CL15' UNKNOWN COND. '                                    7320001
COND1    DC CL15'KEY NOT FOUND '                                     7680001
EXTENTS  DC CL15' OUT OF EXTENT '                                    7720001
         DC CL15'SPACE NOT FOUND'                                    7760001
         DC CL15'INVALID REQUEST'                                    7800001
         DC CL15'SEQUENCE CHECK '                                    7840001
         DC CL15'DUPLICATE RECRD'                                    7880001
*              SYNAD ANALYZE AND FORMAT SVC                  *       0210001
*                 UNIT CHECK ANALYSIS                                0270001
*           FOR EXCP, BPAM, BSAM, QSAM, AND BDAM                     0330001
UNKNOWN  DC CL15'UNKNOWN COND. '                                     7560001
UCKERS   DC CL15'EQP CHECK'                                          7590001
         DC CL15'BUS OUT CK'                                         7620001
         DC CL15' CMD REJECT'                                        7650001
         DC CL15' INT REQ '        MICR                  S19033 7660001
         DC CL15'DATA CHECK'                                         7680001
         DC CL15'OVER RUN'                                           7710001
*   DUMMY ITEM INSERTED TO INSURE PROPER MSG FIELD                   7717001
         DC CL15'UNKNOWN DUMMY                           S19033 7724001
         DC CL15' LATE STKR SEL'     MICR 1419/1275 SCU  S19033 7731001
         DC CL15'TRACK COND CK'                                      7740001
         DC CL15'WORD COUNT ZERO'                                    7770001
         DC CL15'INV CMD SEQ'                                        7800001
         DC CL15' POSITION CK'      TCR                  S19033 7810001
         DC CL15'SEEK CHECK'                                         7830001
         DC CL15'DATA C.CHECK'                                       7860001
         DC CL15' OPERATOR ATTN'    MICR                 S19033 7870001
         DC CL15'TRACK OVERRUN'                                      7890001
         DC CL15'CYL END  '                                          7920001
         DC CL15'INVALID SEQ'                                        7950001
         DC CL15'NO REC FOUND'                                       7980001
         DC CL15'FILE PROT'                                          8010001
         DC CL15'MISSING A.M.'                                       8040001
         DC CL15'OVRF INCP'                                          8070001
         DC CL15'UNKNOWN COND. '                                     8100001
*************************************************************
*              SYNAD ANALYZE AND FORMAT SVC
*              ADDITIONAL SECONDARY LOAD
*              FOR QISAM, BTAM, AND GAM
*************************************************************
         DC CL15' CHAN CTL CK'
         DC CL15'INTF CTL CK'
         DC CL15'PROG CHECK'
         DC CL15'PROT CHECK'
         DC CL15'CHAIN CHECK'
         DC CL15'UNKNOWN COND. '
         DC CL15'END OF FILE'
         DC CL15'WRNG.LEN.RECORD'
         DC CL15'UNKNOWN COND.'
         DC CL15'EQP CHECK'
         DC CL15'BUS OUT CK'
         DC CL15' CMD REJECT'
         DC CL15'DATA CHECK'
```

Figure 180. Sample Constant Area Used in SYNADAF Processing (Part 2 of 3)

```
         DC CL15'DATA CHECK'
         DC CL15'OVER RUN'
         DC CL15'TRACK COND CK'
         DC CL15'WORD COUNT ZERO'
         DC CL15'INV CMD SEQ'
         DC CL15'SEEK CHECK'
         DC CL15'DATA C.CHECK'
         DC CL15'TRACK OVFL'
         DC CL15'CYL END  '
         DC CL15'INVALID SEQ'
         DC CL15'NO REC FOUND'
         DC CL15'FILE PROT'
         DC CL15'MISSING A.M.'
         DC CL15'OVERFL INCP'
         DC CL15'UNKNOWN COND. '
```

Figure 180. Sample Constant Area Used in SYNADAF Processing (Part 3 of 3)

```
//IGC0106H       JOB   ACCTING-INFO
//ZAPSTEP1       EXEC  PGM=IMASPZAP
//SYSLIB         DD    DSN=SYS1.SVCLIB,DISP=OLD
//SYSPRINT       DD    SYSOUT=A
//SYSIN          DD    *
      DUMPT  IGC0106H  ALL
        .
        .
        .
/*
```

Note: The DSN shown is for VS1; for VS2
the DSN would be SYS1.LPALIB

Figure 181. A Sample Job to get a Dump of a Constant Area

An INVALID KEY condition can usually be remedied by merely changing the key and trying the operation again. This technique of altering the key can be used, on a READ, to determine if a record with a particular key already exists in a file; this may be regarded as a test to determine the flow of logic for a particular update operation.

An input/output error condition is usually not easily remedied, and quite often the only operation possible is to close the file.

The following corrective actions are presented according to access method and further broken down according to error condition. If VSAM (Virtual Storage Access Method) is being used, see "Error Processing Options" and "Status Key Settings for Action Requests" in the chapter "VSAM File Processing".

QSAM (physical sequential)

INVALID KEY Condition:

- SPACE NOT FOUND. For a file opened as OUTPUT, no more space exists to contain another record. Processing is limited to a CLOSE, but the file may be further processed as INPUT or I-O.

Input/Output Error Conditions:

- INPUT ERROR.
- OUTPUT ERROR.

  For these two conditions, the user may return to the system from the declarative, thus executing the processing option (EROPT) specified on his DD card:

  1. ABE (default) -- terminate the step with an abend.
  2. SKP -- skip to the next block.
  3. ACC -- accept the block in error and continue processing.

QISAM (indexed sequential)

INVALID KEY Conditions:

- LOWER KEY LIMIT NOT FOUND. The value specified for NOMINAL KEY before a START statement does not have a match in the file. Processing may be continued.
- SEQUENCE CHECK. For a file opened OUTPUT, an attempt was made to add a record whose RECORD KEY was not greater than that of the last record added. Processing may be continued.
- DUPLICATE RECORD. An attempt was made to add a record to a file whose RECORD KEY was already present in the file. Processing may be continued.

Input/Output Error Conditions:

- SPACE NOT FOUND. No space was available in the currently accessible prime area to add the record. Current OUTPUT processing is limited to a CLOSE.
- UNREACHABLE BLOCK (INPUT OR I-O).
- UNCORRECTABLE OUTPUT ERROR.

  For these last two conditions, the user can attempt the operation again; possibly the problem is transient. If the error persists, processing is limited to a CLOSE.
- UNCORRECTABLE INPUT ERROR. The user can attempt to bypass the block in error by executing sufficient READ operations to force the next block into main storage. If the error does not persist on the next block, processing may be continued; otherwise it is limited to a CLOSE.

BISAM (indexed random)

INVALID KEY Conditions:

- RECORD NOT FOUND. The record corresponding to the value of NOMINAL KEY or READ was not found in the file. Processing may be continued.
- DUPLICATE RECORD. An attempt was made to add a record with a key which already exists in the file. Processing may be continued.

Input/Output error Conditions:

- SPACE NOT FOUND. No space was available in the currently accessible prime or overflow area to add the record. Depending on the physical makeup of the file, adds for other keys may be possible. Processing may be continued.

- INVALID REQUEST. A logic error in the source program exists: for example, an attempt to REWRITE a record for which no valid READ was done. Processing is limited to a CLOSE.
- UNCORRECTABLE INPUT/OUTPUT ERROR.
- UNREACHABLE BLOCK (INDEX CANNOT BE READ).

  For these last two conditions, the user can attempt the operation again; possibly the problem is transient. If the error persists, processing is limited to a CLOSE.

BDAM (direct and relative, random)

INVALID KEY Conditions:

- RECORD NOT FOUND. A record corresponding to the value of the key was not found in the file. Processing may be continued.
- END-OF-DATA RECORD READ. The end-of-data set indicator has been read as a result of the value of the ACTUAL or NOMINAL KEY. This is really an indication that the value of the key is outside the limits of the data set. Processing may be continued. It must be emphasized that this is an extremely rare occurrence.
- INVALID REQUEST. This may be caused by two separate conditions:

  1. BLOCK OUTSIDE LIMITS OF DATA SET. The value of the ACTUAL or NOMINAL KEY was found to reference a disk address outside the space occupied by the data set. Processing may be continued.
  2. FIXED LENGTH KEY WITH X'FF'. An attempt was made to add a fixed length record whose ACTUAL KEY has HIGH-VALUE in the first byte of its symbolic portion. Processing may be continued.

Input/Output Error Conditions:

- SPACE NOT FOUND. An attempt has been made to add a record to the data set, and all space allocated to the data set has been filled. No further WRITE to the data set can be executed, but processing may be continued.
- UNCORRECTABLE ERROR, I-O OR NON-I-O. Processing is limited to a CLOSE.

BSAM (direct and relative, sequential)

INVALID KEY Condition:

- SPACE NOT FOUND. For a file opened as OUTPUT, no more space exists to add a record. Processing is limited to a CLOSE, but the file may be further processed as INPUT or I-O.

Input/Output Error Conditions:

- INVALID REQUEST. A logic error in the source program exists: for example, an attempt to REWRITE a record for which no valid READ was done. Processing is limited to a CLOSE.
- INPUT ERROR.
- OUTPUT ERROR.

  For these last two conditions, the user can attempt the operation again; possibly the error is transient. If the error persists, processing is limited to a CLOSE.

APPENDIX H: CREATING AND RETRIEVING INDEXED SEQUENTIAL DATA SETS

Indexed data sets (ISAM) are created and retrieved using special subsets of DD statement parameters and subparameters. They can occupy up to three different areas of space:

- Prime Area -- This area contains data records and related track indexes. It exists for all indexed data sets.
- Overflow Area -- This area contains overflow from the prime area when new data records are added. It is optional.
- Index Area -- This area contains master and cylinder indexes associated with the data set. It exists for any indexed data set that has a prime area occupying more than one cylinder.

Indexed data sets must reside on mass storage volumes. Because an Indexed data set can be associated with more than one type of unit, it is not usually cataloged.

Creating an Indexed Data Set

Indexed data sets are created with from one to three DD statements. One of the statements must define the prime area. If additional areas are to be defined, the DD statements must appear in the following sequence:

1. Index area
2. Prime area
3. Overflow area

This order must be maintained even if one of the statements is absent. Only the first DD statement defining the data set can contain a name field. Other statements, if any, must have a blank name field.

The subset of DD statement parameters used to create an indexed data set excludes the asterisk, DATA, DUMMY, DDNAME, SYSOUT, SUBALLOC, and SPLIT parameters. The remaining DD statement parameters -- DSNAME, UNIT, VOLUME, LABEL, DCB, DISP, SPACE, SEP, and AFF -- are all valid. However, certain restrictions must be followed in using these parameters.

DSNAME: Required. In addition to giving the data set name, the DSNAME parameter identifies the area being defined, i.e., DSNAME=name(INDEX), DSNAME=name(PRIME), and DSNAME=name(OVERFLOW).

Notes:

- If the data set is temporary, name is replaced with &&name.
- If only one DD statement is used to define the entire data set, DSNAME=name(PRIME) or DSNAME=name should be used.

UNIT: Required, unless VOLUME=REF is used. The first subparameter identifies a mass storage unit. If separate statements for the prime and index areas are included, request the same number of units for the prime area as there are volumes. The DEFER subparameter cannot be specified on any of the statements. Another way of requesting units is by using the unit affinity subparameter, AFF.

Notes:

- DD statements for prime and overflow areas must indicate the same type of unit.
- The DD statement for the index area can indicate a unit type different than the others.

VOLUME: Optional. Can be used to request private volumes (PRIVATE), to retain private volumes (RETAIN), or to make specific volume references (SER or REF).

LABEL: Optional. Can be used to specify a retention period (EXPDT or RETPD) and/or password protection (PASSWORD).

DCB: Required. Can be used to complete the data control block if it has not been completed by the processing program. Either DSORG=IS or DSORG=ISU must be included in the list of attributes, even though this attribute was provided in the processing program. If more than one DD statement is used to define the data set, the DCB parameters in the statements must not contain conflicting attributes.

DISP: Optional. Must be coded to keep the data set (KEEP), to catalog it (CATALG), or to pass it to a later job step (PASS). An indexed data set can be cataloged using CATLG only if all three areas are defined by the same DD statement.

Note:

- Indexed data sets defined by more than one DD statement can be cataloged by using the system utility program IEHPROGM, provided all volumes reside on the same type of unit. The utility program IEHPROGM is described in the publication OS/VS Utilities.

SPACE: Required. Space must be requested using either the recommended nonspecific allocation technique or the more restricted absolute track (ABSTR) technique. All DD statements used to define the data set must request space using the same technique.

If the nonspecific space allocation technique is used, space must be requested in units of cylinders (CYL). The quantity of space requested is assigned to the area identified in the DSNAME parameter. If more than one unit is requested, this quantity of space is allocated to each volume used by the data set. Incremental space cannot be requested for indexed data sets. If one DD statement is used to define both the index and prime areas, the size of the index must be indicated in the SPACE parameter of the DD statement defining the prime area. The subparameters RLSE, MXIG, ALX, and ROUND cannot be used. Contiguous space can be requested on each of the volumes occupied by the data set with the subparameter CONTIG. If CONTIG is coded on one of the statements, it must be coded on all of them.

If the absolute track technique of allocating space is used, the number of tracks must be equivalent to an integral number of cylinders. The address of the beginning track must correspond with the first track of a cylinder other than the first cylinder on a volume. If more than one unit is requested, space is allocated beginning at the specified address and continuing through the volume and onto the next volume until the request has been satisfied. If one DD statement is used to define both the index and prime areas, indicate the size of the index (in tracks) in the SPACE parameter of the DD statement defining the prime area. This number must also be equivalent to an integral number of cylinders.

Notes:

- The first volume to be allocated for the prime area of an indexed data set cannot be the volume from which the system is loaded (the IPL volume).

- Space can be requested on more than one volume only on the DD statement that defines the prime area.

SEP AND AFF: Optional. Channel separation from earlier data sets can be requested on any of the DD statements in the group. In order to have areas of an indexed data set written using separate channels, units should be requested by their actual address (e.g., UNIT=190).

Figure 182 illustrates a valid set of DD statements for creating an indexed data set. Note that each area is defined by a separate DD statement.

```
//OUTPUT4  DD  DSNAME=MHB(INDEX),UNIT=2305,DCB=DSORG=IS,                X
//             SPACE=(CYL,10,,CONTIG),DISP=(,KEEP)

//           DD  DSNAME=MHB(PRIME),DCB=DSORG=IS,UNIT=(2305,2),         X
//               VOLUME=SER=(334,335),DISP=(,KEEP),                    X
//               SPACE=(CYL,25,,CONTIG)

//           DD  DSNAME=MHB(OVFLOW),DCB=DSORG=IS,UNIT=2305,            X
//               VOLUME=SER=336,SPACE=(CYL,25,,CONTIG),DISP=(,KEEP)
```

Figure 182. Creating an Indexed Data Set

CRITERIA			Restrictions on Unit Types and Number of Units Requested	Resulting Arrangement of Areas
Number of DD Statements	Types of DD Statements	Index Size Coded?		
3	INDEX PRIME OVFLOW	-	PRIME and OVFLOW must specify the same unit type.	Separate index, prime, and overflow areas.
2	INDEX PRIME	-	None	Separate prime and overflow areas, with an index at the end of the prime area.
2	PRIME OVFLOW	No	Both statements must specify the same type of unit.	Prime area and overflow area with an index at its end.
2	PRIME OVFLOW	Yes	Both statements must specify the same unit type. The statement defining the prime area cannot request more than one unit.	Prime area with embedded index and overflow area.
2	PRIME	No	None	Prime area with index at its end. Unused index areas, if any, used for overflow.
1	PRIME	Yes	Cannot request more than one unit.	Prime area with embedded index area.

Figure 183. Area Arrangement for Indexed Data Sets

The manner in which the areas of an indexed data set are arranged is based primarily on two criteria:

- The number of DD statements used to define the data set.
- The types of DD statements used (as reflected in the DSNAME parameter).

An additional criterion arises when a DD statement is not included for the index area:

- The index size and whether or not it has been coded in the SPACE parameter of the DD statement defining the prime area.

Figure 183 illustrates the arrangements resulting from various permutations of the foregoing criteria. In addition, it points out restrictions on the number and type of units that can be requested for each permutation.

Retrieving an Indexed Data Set

Indexed data sets are retrieved with the DD statement parameters DSNAME, UNIT, VOLUME, DCB, and DISP. Channel separation requests can be made using the SEP and AFF parameters. If all areas of the data set reside on the same type of unit, the entire data set can be retrieved with one DD statement. If the index resides on a different type of unit, two DD statements must be used.

DSNAME: Required. Identify the data set by its name. If it was passed from a previous step, identify it by a backward reference or its temporary name. Do not include the terms INDEX, PRIME, or OVFLOW.

UNIT: Required, unless the data set was passed on one volume. Identify the unit type. If the data set resides on more than one volume and all units are the same type, request the total number of units required by all areas. If the index area resides on a different type of unit, use two DD statements, each

indicating the number of units of the specified type required.

VOLUME: Required, unless the data set was passed on one volume. Identify the volumes by their serial numbers (SER), listed in the same sequence as they were when the data set was created.

DCB: Required, unless the data set was passed. This parameter is used to complete the data control block if it was not completed in the program. Include either DSORG=IS or DSORG=ISU.

DISP: Required. Identify the data set as OLD or MOD and give its new disposition, to change its disposition.

Figure 184 shows how to retrieve the indexed data set created by the illustration in Figure 182.

```
//INPUT    DD  DSNAME=MHB,DCB=DSORG=IS,UNIT=2305,DISP=OLD
//            DSNAME=MHB,DCB=DSORG=IS,UNIT=(2305,3),DISP=OLD,        X
//            VOLUME=SER=(334,335,336)
```

Figure 184. Retrieving an Indexed Data Set

APPENDIX I: CHECKLIST FOR JOB CONTROL PROCEDURES

This checklist illustrates general job control procedures for compiler, linkage editor, and execution processing. More than one example may be used for a job step. The checklist is intended as an aid to preparing procedures, not as an inclusive list of the options and parameters.

COMPILATION

Figure 185 shows a general job control procedure for a compilation job step. The following cases demonstrate how to add to or modify the general procedure to obtain various processing options.

Case 1: Compilation Only -- No Object Module Is to Be Produced

The general procedure should be used. A listing is produced. It will include the default or specified options of the PARM parameter that affect output. Any diagnostic messages are listed, unless listing of warning messages is suppressed by the FLAGE option of the PARM parameter and only warning messages are produced.

Case 2: Source Module from Input Stream

Modify the end of the procedure as follows:

```
//SYSIN  DD  *
    (source module)
/*
```

If the DD * convention is used, the source module must follow. If another job step follows the compilation, the EXEC statement for that step follows the /* statement, or the last source statement.

Case 3: Object Module Is to Be Punched

Add the statement:

```
//SYSPUNCH  DD  SYSOUT=B
```

Note: If DECK is not the installation default condition, it must be specified in the PARM parameter of the EXEC statement.

Case 4: Object Module Is to Be Passed to Linkage Editor

Add the statement:

```
//SYSLIN     DD    DSNAME=(subparms),      X
//                 UNIT=SYSDA,             X
//                 SPACE=(subparms),       X
//                 DISP=(MOD,PASS)
```

Note: If LOAD is not the installation default condition, it must be specified in the PARM parameter of the EXEC statement.

```
//jobname  JOB  acctno,name,MSGLEVEL=1
//stepname EXEC PGM=IKFCBL00,PARM=(options)
//SYSUT1   DD   UNIT=SYSDA,SPACE=(subparms)
//SYSUT2   DD   UNIT=SYSDA,SPACE=(subparms)
//SYSUT3   DD   UNIT=SYSDA,SPACE=(subparms)
//SYSUT4   DD   UNIT=SYSDA,SPACE=(subparms)
//SYSPRINT DD   SYSOUT=A
//SYSIN    DD   DSNAME=dsname,UNIT=SYSSQ,VOLUME=(subparms),          X
//             DISP=(OLD,KEEP)
```

Figure 185. General Job Control Procedure for Compilation

Case 5: Object Module Is to Be Saved

The object module can be saved by cataloging it, by keeping it, or by adding it as a member of a library. Add the SYSLIN statement as shown in examples A, B, or C.

- A. Cataloging

```
//SYSLIN   DD  DSNAME=dsname,                X
                  NEW
//             DISP=(      ,CATLG),          X
                  MOD
//             VOLUME=(subparms),             X
//             LABEL=(subparms),              X
                     SYSDA
//             UNIT=       ,                  X
                     SYSSQ

               SPACE
//             SPLIT     =(subparms)
               SUBALLOC
```

- B. Keeping

```
//SYSLIN   DD  DSNAME=dsname,                X
                  NEW
//             DISP=(      ,KEEP),           X
                  MOD
//             VOLUME=(subparms),             X
//             LABEL=(subparms),              X
                     SYSDA
//             UNIT=       ,                  X
                     SYSSQ


               SPACE
//             SPLIT     =(subparms)
               SUBALLOC
```

- C. Adding a Member to an Existing Cataloged Library

```
//SYSLIN   DD  DSNAME=dsname(member),        X
//             DISP=OLD
```

Case 6: COPY Statement in COBOL Source Module or a BASIS Card in the Input Stream

Add the SYSLIB (or equivalent) DD card(s), as shown in examples A, B, or C.

A. COPY

```
//SYSLIB   DD  DSNAME=copylibname,DISP=SHR
```

B. BASIS Card

```
//SYSLIB   DD  DSNAME=basislibname,DISP=SHR
```

C. Both BASIS and COPY

```
//SYSLIB   DD  DSNAME=basislibname,DISP=SHR
//            DD  DSNAME=copylibname,DISP=SHR
```

(DD statements for additional copylibs may follow.)

LINKAGE EDITOR

Figure 186 shows a general job control procedure for a linkage editor job step. The following cases show how to add to or modify the procedure to obtain various processing options.

Case 1: Input from Previous Compilation in Same Job

Change the SYSLIN statement to

```
//SYSLIN   DD  DSNAME=*.stepname.SYSLIN,     X
//             DISP=(OLD,DELETE)
```

where stepname is the name of the previous compilation job step and ddname is SYSLIN. If the input is to be saved, specify KEEP rather than DELETE.

Case 2: Input from System Input Stream

Change SYSLIN statement and the end of the procedure as follows:

```
//SYSLIN   DD *
    (object module(s))
/*
```

If another job step follows the link-edit step, the EXEC statement for that job step follows the /* statement or the end of the object module.

Case 3: Input Not from Compilation in Same Job

Specify in the SYSLIN DD statement where the object modules to be used as input are stored. (Only one member of a library can be specified in the SYSLIN DD statement.)

```
//jobname   JOB   acctno,name,MSGLEVEL=1
                .
                .
                .
//stepname  EXEC  PGM=IEWL,PARM=(options)
//SYSPRINT  DD    SYSOUT=A
//SYSLMOD   DD    DSNAME=&&name(member),UNIT=SYSDA,DISP=(NEW,PASS),   X
//                SPACE=(subparms)
//SYSLIB    DD    DSNAME=SYS1.COBLIB,DISP=OLD
//SYSUT1    DD    UNIT=SYSDA,SPACE=(subparms)
//SYSLIN    DD    DSNAME=dsname,DISP=OLD
```

Figure 186. General Job Control Procedure for a Linkage Editor Job Step

Case 4: Output to Be Placed in Link Library

Change the SYSLMOD statement as follows:

```
//SYSLMOD  DD DSNAME=SYS1.LINKLIB(member),X
//            DISP=OLD
```

where member is the name of the load module that is to be added to the link library. No other information is needed in the statement.

Case 5: Output to Be Placed in Private Library

Change the SYSLMOD statement as follows:

```
//SYSLMOD  DD  DSNAME=dsname(member),     X
//             DISP=OLD
```

where member is the name of the load module to be added, and dsname is the name of an existing library. If the library is not cataloged, UNIT and VOLUME parameters must be specified.

Note: See "Using the DD Statement" in the Chapter "User Non-VSAM File Processing" for an example of creating a new library and storing the load module as its first member.

Case 6: Output to Be Used Only in this Job

The general procedure should be used. The load module is stored in a temporary library.

EXECUTION TIME

Figure 187 shows a general job control procedure for an execution-time job step.

The following cases show how to add to or modify the general procedure to obtain various processing options.

Case 1: Load Module to Be Executed Is in Link Library

Use the general procedure, where progname in the EXEC statement is the member name of the load module.

Case 2: Load Module to Be Executed Is a Member of Private Library

If a STEPLIB DD statement is not in the JCL for the step where the program is required, then the JOBLIB DD statement must follow the JOB statement, as in the following statements:

```
//JOB1     JOB
//JOBLIB   DD    DSNAME=MYLIB,              X
//                 DISP=(OLD,PASS)
//STEP1    EXEC  PGM=PAYROLL
               .
               .
               .

//STEP2    EXEC  PGM=ACCOUNT
               .
               .
               .
```

The JOBLIB statement defines the private library MYLIB. No volume or unit parameters are given since the library is cataloged. Since JOBLIB has the disposition PASS, both steps can execute members of the library named in the JOBLIB statement.

```
//stepname EXEC  PGM=progname
//ddname   DD    (parameters for user-specified data sets)
           .
           .
           .
```

Figure 187. General Job Control Procedure for an Execution-Time Job Step

Case 3: Load Module to Be Executed Is Created in Previous Linkage Editor Step in Same Job

Change the EXEC statement as follows:

```
//stepname  EXEC  PGM=*.stepname.SYSLMOD
```

where stepname following PGM is the name of the linkage editor job step that created the load module.

Case 4: Abnormal Termination Dump

Add the statement:

```
//SYSABEND  DD  SYSOUT=A
          or
//SYSUDUMP DD SYSOUT=A
```

This statement requests a full dump if abnormal termination occurs during execution.

Case 5: DISPLAY Is Included in Source Module

Add the statement:

```
//SYSOUT   DD  SYSOUT=A
```

Case 6: DISPLAY UPON SYSPUNCH Is Included in Source Module

Add the statement:

```
//SYSPUNCH  DD  SYSOUT=B
```

Case 7: ACCEPT Is Included in Source Module (Except for Format 2 or ACCEPT MESSAGE)

If the data is in the input stream, add the statement:

```
//SYSIN    DD  *
          (data)
/*
```

(See Case 2 under "General Job Control Procedures for a Compilation Job Step" for a description of the DD * convention.)

Case 8: Debug Statements EXHIBIT or TRACE Are Included in Source Module

Use the statement (unless it is already included):

```
//SYSOUT   DD  SYSOUT=A
```

Note: If the job step already includes a SYSOUT DD statement for some other use, another need not be inserted.

Case 9: Object Time Symbolic Debugging Options

Add the statements:

```
//SYSDBOUT DD SYSOUT=A required for all
                         options
//SYSDBG   DD *        required for SYMDMP
                         option
   (control cards)
/*
```

debug DDname card also needed

Case 10: COUNT Option

Add the statements:

```
//SYSCOUNT DD SYSOUT=A
//SYSDBOUT DD SYSOUT=A required for all
                        options
```

APPENDIX J: FIELDS OF THE GLOBAL TABLE

In this appendix, each field of the Task Global Table (Figure 188) and of the Program Global Table (Figure 189) is listed by its relative location in main storage. Each field is further described in the discussion associated with Figures 188 and 189.

TASK GLOBAL TABLE

The Task Global Table (TGT) is used to record and save information needed during execution of the object program. It begins with a series of fixed-length fields followed by a series of variable-length fields. These fields are illustrated in Figure 188 and are described in this section.

Relative Location	Field
0	SAVE AREA
72	SWITCH
76	TALLY
80	SORT SAVE
84	ENTRY SAVE
88	SORT CORE SIZE
92	RET CODE
94	SORT RET
96	WORKING CELLS
400	SORT FILE SIZE
404	SORT MODE SIZE
408	PGT-VN TABLE
412	TGT-VN TABLE
416	VCON PTR
420	LENGTH OF VN TBL
422	LABEL RET
423	CURRENT PRIORITY
424	DBG R14SAVE
428	COBOL INDICATOR
432	A(INIT1)
*436	DEBUG TABLE PTR
440	SUBCOM PTR
444	SORT-MESSAGE
452	SYSOUT DDNAME
453	Reserved
454	COBOL ID
456	A (WHEN-COMPILED) INFO
460	COUNT TABLE ADDRESS
472	DBG R11SAVE
476	COUNT CHAIN ADDRESS
480	PRB1 CELL PTR

Figure 188. Fields of the Task Global Table (Part 1 of 3)

Relative Location	Field
484	Unused
489	TA LENGTH
492	Unused
500	PCS LIT PTR
504	DEBUGGI NG
508	CD FOR INITIAL INPUT
512 beginning of variable-length portion	OVERFLOW
	BL
	DECBADR
	FIB
	DEBUG TRANSFER
	DEBUG CARD
	DEBUG BLL
	DEBUG VLC
	DEBUG MAX
	DEBUG PTR
	TEMP STORAGE
	TEMP STORAGE-2
	TEMP STORAGE-3
	TEMP STORAGE-4
	BLL
	VLC
	SBL
	IND
	SUBADR
	ONCTL
	PFMCTL
	PFMSAV
	VN
	SAVE AREA-2

Figure 188. Fields of the Task Global Table (Part 2 of 3)

Relative Location	Field
	SAVE AREA-3
	XSASW
	XSA
	PARAM
	RPTSAV AREA
	CHECKPT CTR
	VCON TBL
	DEBUG TABLE

Figure 188. Fields of the Task Global Table (Part 3 of 3)

The lengths of the variable-length fields are determined by the requirements of the program (if not required, a particular field may not exist in the object program).

SAVE AREA
the program's save area; used to provide standard subroutine linkage when this program is called (by the Operating System or by another program) and when this program calls other programs.

SWITCH
a fullword switch. Only the following bits are used:

Bit	Meaning
0	Indicates a size error in series addition or subtraction. If a SIZE ERROR clause was included in the source statement, and a size error occurs before all data items in the series have been added or subtracted, this bit is set to 1. It is tested after the entire addition or subtraction is complete. If the value is 1, the instructions generated for the ON SIZE ERROR clause are executed.
1	Used for TRACE. It is set to 1 by the execution of a READY statement, and reset to 0 by a RESET statement. If the program uses a TRACE statement, there are instructions to test this bit at the point of definition for every source program procedure-name (PN). If it is on, the DISPLAY subroutine (ILBODSP0) is called to print the card number of the procedure-name. (See "Appendix B: COBOL Library Subroutines" for a description of the DISPLAY subroutine.)
2	Indicates program initialization. Set to 1 by routine INIT3 to show that initialization has been performed. Tested by INIT3 so that if the module is re-entered, INIT3 can perform re-entry functions instead of initialization functions.
3	Main or subprogram switch. Set by INIT2 if this is a main program.
4	Used for SYMDMP. It is set to 1 if the symbolic debug option is in effect for the program. This bit is tested by the object-time COBOL library debugging control subroutine ILBODBG0.
5	Used for FLOW. It is set to 1 if the flow trace option is in effect for the program. This bit is tested by the object-time COBOL library debugging control subroutine ILBODBG0.
6	Used for STATE at program initialization time. If on, bit 10 is set on and this bit turned off. Thereafter in the object program, this bit is used to indicate that an ON OVERFLOW or ON SIZE ERROR condition has occurred for a statement.
7	Used for OPT. It is set to 1 if optimization has been requested for the program or if the SYMDMP or STATE and OPT, or FLOW and OPT options have been specified.
9	Used for CALL, CANCEL, or a recursive CALL. It is set to 1 by the generated code for the CALL or CANCEL verb. It is tested by INIT2 to determine whether a recursive CALL condition exists.
10	Set on in program initialization if STATE is requested.
12	Used for QUOTE IS APOST. It is set to 1 if the apostrophe is to be used to delineate literals and to be used in the generation of figurative constants.
13	Used for SYMDMP. It is set to 1 if SYMDMP is requested and the program contains a floating-point item.
14	Always set to 1.
15	Indicates maximum length for a variable-length field. Before the execution of a Q-Routine, this bit is set

to 1 if the VLC and SBL for the field are to be set to their maximum possible values, rather than a value depending on the current value of a data item. The maximum value is the value of X in the clause "OCCURS X TIMES DEPENDING ON...".

16 SRVBIT set on if ILBOLOM is link-edited with program. Set to 1 if COUNT is specified

24-31 DECIMAL-POINT IS COMMA clause byte. If this clause was specified, the byte contains a comma in EBCDIC. If not, it contains a decimal point.

TALLY
a fullword used for source program references to the special register TALLY.

SORT SAVE
a fullword used during the execution of a SORT/MERGE RETURN statement to contain the GN for the next sequential instruction following the RETURN.

ENTRY SAVE
a fullword used to save the entry point of the program during INIT2 and INIT3 execution.

SORT CORE SIZE
a fullword for the SORT-CORE-SIZE special register as used in the source program.

RET CODE
a halfword for the RETURN-CODE special register, which is used in the source program to provide a completion code on a STOP RUN, EXIT PROGRAM, or GOBACK statement, or to store the return code from a called program. It is the user's responsibility to set this code.

SORT RET
a halfword used to contain the return code from a SORT/MERGE operation.

WORKING CELLS
variable-length cells used by COBOL library subroutines called by the program. The total length of the field is 304 bytes.

SORT FILE SIZE
a fullword for the SORT-FILE-SIZE special register as used in the source program.

SORT MODE SIZE
a fullword for the SORT-MODE-SIZE special register as used in the source program.

PGT-VN TBL
a fullword pointer to that part of the VN field of the PGT containing VN's for independent segments.

TGT-VN TBL
a fullword pointer to that part of the VN field of the TGT containing VN's for independent segments.

VCON PTR
pointer to the VCON TBL field of the TGT. This is required because the VCON TBL field is variably located, and the VCON PTR is fixed within the TGT.

LENGTH OF IND VN TBL
a halfword containing the length of that part of the VN field (the length is the same for both the TGT and PGT) containing VN's for the independent segments.

LABEL RET
the LABEL-RETURN special register for nonstandard labels. If an error occurs in such a label, it is the user's responsibility to place a nonzero value into this 1-byte cell.

CURRENT PRIORITY
for a segmented program, the segmentation subroutine ILBOSGM0 inserts the priority of the segment currently in the transient area. This field is initialized to 0. The current priority cell is used in all segmented programs to store the priority of the segment currently loaded. Subroutine ILBOSGM0 uses it to determine whether to load and/or initialize the segment of destination in a branch.

DBG R14SAVE
indicates the contents of register 14. A routine of the debug control subroutine ILBODBG0 is called to save this information before the execution of any instruction that passes control outside the COBOL program.

COBOL INDICATOR
identifies the object program as an OS/VS COBOL program.

INIT1 ADCON
address of INIT1 used for GOBACK, STOP RUN, and EXIT PROGRAM instructions, and for segmentation coding.

TGTTAB PTR
: if the FLOW SYMDMP or STATE compiler options are specified, this field points to the TGTTAB.

SUBCOM PTR
: a pointer to the subroutine communications (SUBCOM) area in the COBOL subroutine library.

SORT-MESSAGE
: an 8-byte area for the SORT-MESSAGE special register, which is used in the source program to allow the user to specify to the Sort/Merge program where to place the messages it issues.

SYSOUT DDNAME
: a 1-byte area with the SYSx character.

COBOL ID
: contains the identifying number of the compiler.

WHEN-COMPILED ADDRESS
: Address of WHEN-COMPILED information in INIT1.

COUNT TABLE ADDRESS
: Relative address of the COUNT table from the beginning of the TGT. The COUNT table is located between the Q-routines, if any, and the INIT2 routine. The count table is used only when the program terminates.

DBG R11SAVE
: indicates the contents of register 11. When the dynamic dumping routine of the debug control subroutine ILBODBG0 receives control, it places the return address to the in-line code of the calling program in register 11. Therefore, the contents of register 11 must be saved.

COUNT CHAIN ADDRESS
: Address of the COUNT CHAIN for this program. The address is initialized to zero if count is specified; the address is filled in at execution time.

PRBL1 CELL PTR
: a fullword cell containing the address of the first PROCEDURE BLOCK cell in the PGT.

TA LENGTH
: a halfword initialized to the length of the largest segment with a nonzero priority.

PCS LIT PTR
: a fullword cell containing the address of the PCS (Program Collating Sequence) alphabet.

DEBUGGING
: a fullword cell containing the address of the beginning of the debugging cells in the variable portion of this table.

CD FOR INITIAL INPUT
: a fullword cell containing the address of the CD area with INITIAL INPUT clause.

OVERFLOW
: if the TGT is longer than 4096 bytes, this field contains one fullword cell pointing to each 4096-byte area after the first. The cell is loaded into a register when a base is required for the overflow area.

BL
: base locators. Each BL cell is a fullword containing an address in the data area. There is one BL pointing to the beginning of the Working-Storage Section and one for each file in the File Section. More than one BL is assigned if an area is larger than 4096 bytes.

DECBADR
: DECB addresses. There is one fullword cell pointing to the address of the DECB for each basic file.

FIB
: File Information Block addresses. There is one fullword cell pointing to the address of the FIB for each VSAM file.

DEBUG TRANSFER
: a 1-byte cell that indicates the type of invocation for a PN.

DEBUG CARD
: a 2-byte cell containing the card number.

DEBUG BLL
: a 2-byte cell containing the displacement to the BLL cell.

DEBUG VLC
: a 2-byte cell containing the displacement to the VLC cell.

DEBUG MAX
: a 2-byte cell containing the maximum size of DEBUG-ITEM.

DEBUG PTR
: a fullword cell containing a pointer used by ILBOBUG to reference the debug subscript table.

TEMP STORAGE
: temporary storage for arithmetic

operations. TS space is allocated in doubleword blocks.

TEMP STORAGE-2
temporary storage for nonarithmetic instructions. These cells are variable in length.

TEMP STORAGE-3
temporary storage used to align fields of data described by the SYNCHRONIZED option. The field begins on a doubleword boundary.

TEMP STORAGE-4
temporary storage cells used for the SEARCH ALL table-handling verb. The field starts on a doubleword boundary.

BLL
base locators for the Linkage Section. Each BLL cell is a fullword containing the address of an area passed as a result of an ENTRY statement, a label record, a totaled area, a sort description entry, or a GIVING option in a USE...ERROR statement.

VLC
variable-length cells. Each VLC is a halfword whose value is set by the execution of a Q-Routine. It contains the current length of a variable-length field. There is one VLC for each OCCURS...DEPENDING ON clause and all items to which it is subordinate.

SBL
secondary base locators. Each SBL cell is a fullword set by the execution of a Q-Routine. It contains the current address of a field which is variably located because it follows a variable-length field.

IND
fullword cells, each containing the current value of an INDEX-NAME. There is one IND cell for each implicitly-defined INDEX-NAME. (Explicitly-defined INDEX-NAMEs are not listed in these cells.)

SUBADR
subscript addresses. Each SUBADR cell is a fullword containing the address for a subscripted reference.

ONCTL
control counters for ON statements. Each is a fullword initialized to 0.

PFMCTL
PERFORM control counters and DEBUG saved location. Each PFMCTL cell is a fullword used for a PERFORM n TIMES statement to count the number of times the procedure has been performed. For DEBUG, a PFMCTL cell is used to save the contents of register 14 when the DEBUG packet is entered. DEBUG packets are called by BALR 14,15.

PFMSAV
PERFORM saved locations. Each is a fullword used to contain an address. For PERFORM, the cell is used to store the address of the next sequential instruction after the performed procedure, when that procedure is being executed because of a PERFORM. This is to enable the procedure to be executed in-line.

VN
variable procedure-names. Each VN cell is a doubleword containing the current address of a branch point which may change during program execution because of an ALTER or PERFORM statement.

SAVE AREA-2
pointer to the save area for label- and error-processing declaratives.

SAVE AREA-3
variable number of fullwords used for OPEN parameters.

XSASW
1-byte EXHIBIT switches. These are used as first-time switches for the coding generated for the EXHIBIT CHANGED statement. They are also used in certain types of SORT statements and ON statements.

XSA
EXHIBIT saved area cells. These are variable in length and are referred to in the coding generated for an EXHIBIT CHANGED statement. There is one XSA for each operand to be exhibited with a CHANGED option. These cells are also used for SORT and RELEASE verbs.

PARAM
parameter area of fullwords, containing parameter lists for macro instruction expansions of certain source statements. The size of the parameter area equals the largest number of words required for any one expansion.

RPTSAV
six words used to save branch addresses during the execution of Report Writer routines, if the Report Writer is used.

CHECKPT CTR
fullword cells used to count the number of file records processed for a file for which checkpoints are to be taken.

VCON TBL
8-byte V-type address constants for nonresident segments. The format of each entry is:

Byte	Contents
0	Priority number
1-3	0
4-7	VCON to independent segment

DEBUG TABLE
table used by the flow trace and statement number and symbolic debug COBOL library subroutines. The format depends on the options specified.

- If the FLOW compiler option is specified:

Byte(s)	Contents
0	Number of traces requested
1-3	Unused

- If the STATE option is specified:

Byte(s)	Contents
0-3	Start of Q-Routines, or if none, start of INIT2.
4-7	Size of Declaratives (not including Report Writer) Section.
8-11	Starting address of PROCTAB in object module.
12-15	Starting address of SEGINDX in object module.
16-19	Ending address of SEGINDX in object module.

- If both the FLOW and STATE compiler options are specified:

Byte(s)	Contents
0	Number of traces requested
1-19	The same as shown above for the STATE option.

- If the SYMDMP option is specified:

Byte(s)	Contents
0-3	Start of Q-Routines, or if none, start of INIT2.
4-5	Hashed compilation indicator.

- If both the SYMDMP and FLOW options are specified:

Byte(s)	Contents
0	Number of traces requested.
1-5	The same as shown above for the SYMDMP option.

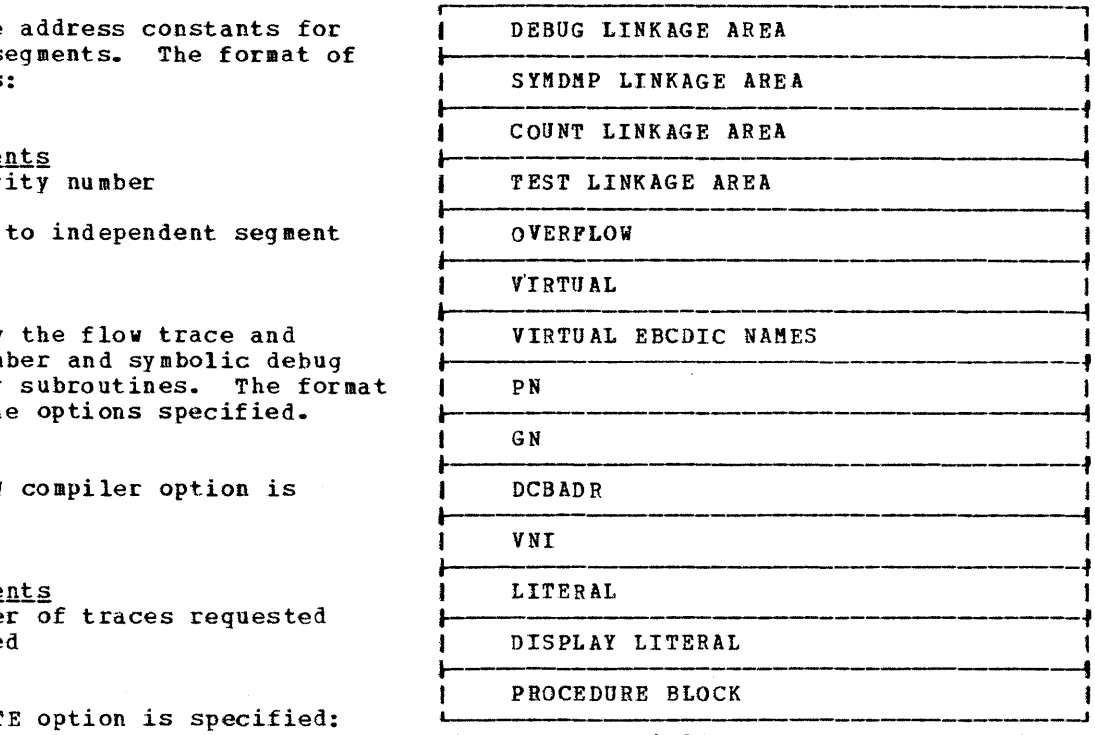


Figure 189. Fields of the Program Global Table

PROGRAM GLOBAL TABLE

The Program Global Table (PGT) contains data referenced by procedure instructions. All the fields in the PGT are variable in length. PGT data is never modified by procedure instructions; rather, it remains constant throughout program execution.

The fields in the PGT are illustrated in Figure 189 and described in the text below.

DEBUG LINKAGE AREA
a 12-byte area that contains the linkage for dynamic dumps. If the SYMDMP option is not specified, this area does not exist.

SYMDMP LINKAGE AREA
a 12-byte area that contains the linkage to the SYMDMP routine for dynamic dump requests.

COUNT LINKAGE AREA
8-byte area that contains the linkage to the COUNT routine. If the COUNT option is not specified, this 8-byte area does not exist.

TEST LINKAGE AREA
16-byte field that contains the linkage to the IBM OS COBOL Interactive Debug Program Product (Program No. 5734-CB4) when TEST was specified for compilation.

OVERFLOW
if the entire PGT exceeds 4096 bytes in length, there is one fullword OVERFLOW cell pointing to each 4096-byte section after the first. The cell is loaded into a register when a base is needed to refer to the section of the PGT.

VIRTUAL
each virtual is a fullword containing the address of an external procedure (the result of an ESD and RLD in the object module) unless either the DYNAM or the RESIDENT option is in effect. If either of these options is in effect, the virtuals corresponding to library subroutines are written as EBCDIC'/ 00 00 00';/ in addition, if the DYNAM option is in effect, the virtuals corresponding to user subprograms contain the relative displacement of the subprogram name from the beginning of the PGT. It is required because of a CALL statement in the source program or a branch to a COBOL library object-time subroutine.

VIRTUAL EBCDIC NAMES
indicates the EBCDIC names of library subroutines and user subprograms. If either the DYNAM or the RESIDENT option is in effect, the EBCDIC names of all library subroutines that are to be dynamically loaded are listed; in addition, if DYNAM is in effect, the EBCDIC names of all user subprograms that are to be dynamically called are listed. Each VIRTUAL EBCDIC NAME cell is a doubleword containing the name of the subroutine or subprogram, left justified and padded with blanks if necessary. If neither DYNAM nor RESIDENT is in effect, this field does not exist.

PN
source program procedure-names. When the OPT option is in effect, only those PN's associated with ALTER and declaratives references receive PN cells. Each PN cell is a fullword containing the address of the first instruction in a block of coding. The addresses of the PN's are in the same order as their definition in the source program. The program branches by loading an address from the PGT and then branching to it.

GN
compiler-generated procedure-names. When the OPT is in effect, only those GN's associated with AT END and INVALID KEY references receive GN cells. Each GN is a fullword containing the address of the first instruction in a block of coding. GN's are used in the same way as PN's. They were generated to provide addresses for branches implied but not stated in the source program. They are stored in the PGT in the order in which they were generated.

DCBADR
DCB addresses. Each DCBADR cell is a fullword containing the address of a data control block in the data area of the program. There is one DCBADR cell for each DCB generated.

VNI
variable procedure-name initialization cells. There is one doubleword VN cell for each variable procedure-name in the program. It contains the initial value of the VN, and is used to initialize the VN values in the TGT. VN's are generated to contain branch addresses which vary because of PERFORM or ALTER statements.

LITERAL
literals referred to by procedure instructions. The literals are variable in length. There is no duplication in storage, since duplicate literals were eliminated.

DISPLAY LITERAL
literals used by calling sequences rather than instructions. They are variable in length; duplication was eliminated. each cell is a fullword containing the address of a procedure block. The compiler assigns these cells only when the OPT option is in effect.

PROCEDURE BLOCK
each cell is a fullword containing the address of a procedure block. The compiler assigns these cells only when the OPTIMIZE option is in effect.

APPENDIX K: DIAGNOSTIC MESSAGES

This appendix contains the COBOL object-time messages (including those messages issued by the queue-analyzer subroutine) and briefly describes how to generate a listing of COBOL compile-time messages.

COMPILE-TIME MESSAGES

The user can request a complete listing of the diagnostic messages issued by this compiler by compiling a program with a program-name of ERRMSG specified in the PROGRAM-ID paragraph. For a description of the formats of compiler diagnostics and information about generating this listing, see "Compiler Output" in the "Output" part of this publication.

If the compiler encounters a D-level ("disaster") error situation, and the user program being processed had specified the DUMP compile option, then the associated error message for that situation is in most cases not produced. (The additional processing needed to produce the message would alter the contents of internal storage and thus reduce the dump's value.) In place of the message, the compiler issues a four-digit U-type completion code along with the resultant abend dump. A list of these codes and an indication of their origin within the compiler phases can be found in the manual IBM OS/VS COBOL Compiler Program Logic, Order No. LY28-6486.

OBJECT-TIME MESSAGES

The COBOL library subroutines issue both informative messages and an occasional U-type abend completion code.

Completion Codes

U0187 An incorrect compiler-generated verb table has been discovered. The abend (initiated by ILBOTC3) follows message IKF187I.

U0203 An attempt to divide by zero (normally causing an ABEND OCB) was detected by ILBOXDI, and ON SIZE ERROR was not specified. Register 14 points to the location in the program that caused the error.

U0295 The Return Code (RC12) has been changed from positive to negative. The abend (initiated by ILBOSRV) may follow a terminal message.

U0303 The time stamp of the volume on which a VSAM data set is stored does not match the system time stamp in the data set's catalog record. ILBOVOC initiates the abend.

U0304 The time stamps of a VSAM data component and an index component do not match, indicating that either the data or the index has been updated separately from the other. ILBOVOC initiates the abend.

U0519 Execution has reached the bottom of the Procedure Division, but does not find a STOP RUN, GO BACK, or EXIT PROGRAM statement. An error exists in the program's logic flow. The abend is initiated by the compiler-generated object code. A warning about the situation may have been issued at compile time.

U1301 I/O error for a non-QSAM/VSAM file for which no error declarative was coded. ILBOSYN initiates the abend.

U3361 The PCONTROL name is not the same as the current COBOL program name and SYMDMP was not cancelled. ILBOD21 initiates the abend.

U3440 Insufficient main storage is available, or an invalid GETMAIN/FREEMAIN request has occurred. The abend (initiated by ILBOCMM) follows message IKF993I or IKF994I.

U3505 A flow-of-control error has been discovered. The abend (initiated by ILBODBG1) follows message IKF193I.

Informative Messages

The following messages are preceded by a system-generated 2-character numeric field, which is used to identify the program issuing the message and may be required in the operator response.

IKF000A- xxx

Explanation: This message is generated by the STOP statement with the 'literal' option. The message text is supplied by the object program and may indicate alternative action to be taken.

System Action: The object program enters wait state.

Programmer Response: Check message text supplied by the object program on alternative action to be taken.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

Operator Response: Follow instructions given by the programmer when program was submitted for execution. If the job step is to be resumed, enter

REPLY xx, 'y'

where y is any single character. Processing continues.

IKF111I Text as supplied by system SYNADAF routine.

Explanation: This information is provided when a permanent input/output error or some other exceptional input/output condition has occurred and no provisions were made to handle it within the COBOL program. The data set is not closed and control is returned to the next higher level program.

Programmer Response: Probable user error. Include an input/output error declarative for the appropriate file to process the error condition.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

Operator Response: Probable user error. Supply the programmer with the console message.

IKF115I QSAM ERROR AT DISPLACEMENT nnnn IN PROGRAM xxxx--FILE STATUS IS yy

Explanation: An error which would result in a FILE STATUS value of 90 or higher (identified by yy in the message) has been encountered at the relative displacement identified by nnnn in the COBOL program whose PROGRAM-ID is xxxx; however, a FILE STATUS clause was not specified and no error declarative was active for the QSAM file.

System Action: COBOL terminates execution of the program and returns to its caller with a return code of 12.

Programmer Response: Correct the indicated error. Consider adding a FILE STATUS clause to intercept such errors for in-program handling.

IKF120I TABLE OVERFLOW. TOO MANY DYNAMIC CALLS.

Explanation: This message is issued when the number of dynamic calls exceeds that which the compiler can process.

System Action: The run is terminated.

Programmer Response: Probable user error. Reduce the number of dynamic calls and rerun the job.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF129I UNEXPECTED DATA FOLLOWING LAST SLASH IN PARM FIELD, LAST SLASH AND UNEXPECTED DATA TRUNCATED.

Explanation: A slash is required to separate user-defined parameters from COBOL-defined parameters in the PARM field of the EXEC statement at execution time.

System Action: The last slash and all data following it in the PARM field are truncated and not passed to the object program.

Programmer Response: Correct the data following the last slash to contain only those items expected by COBOL for execution time (see the section "Options for Execution" in the chapter "Job Control Procedures").

IKF140I NO STORAGE AVAILABLE FOR STAE. ALL DEBUGGING OPTIONS CANCELLED.

System Action: All debugging options are canceled.

Programmer Response: Probable user error. Increase REGION SIZE and re-execute the program.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF141I- INSUFFICIENT INFORMATION PASSED BY STAE ON ABEND. RERUN JOB.

System Action: All debugging options are canceled.

Programmer Response: Probable user error. Increase REGION SIZE and re-execute the program.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF143I-* INSUFFICIENT FLOW TRACE TABLE SPACE.

Explanation: The Operating System GETMAIN macro instruction returned a nonzero code, indicating that sufficient space in the region is not available for the Flow Trace table.

System Action: Flow output is canceled for the program.

Programmer Response: Probable user error. Increase REGION SIZE and re-execute the program.

If the problem recurs, have the following available before calling IBM for programming

support: source deck, control cards, and compiler output.

IKF155I- STATEMENT NUMBER ERROR.

Explanation: A compiler or logic error has occurred during STATE option processing. Under certain conditions, this error may result from other user errors. For example, a loop might destroy some of the information required by the STATE subroutines; an invalid branch might cause a non-existent priority-number to be stored in the TGT, etc.

System Action: STATE option canceled.

Programmer Response: If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF158I-* TOO MANY PROGRAMS TRACED.

Explanation: The FLOW option is effective in a subprogram structure of more than 10 programs compiled with the FLOW option.

System Action: Programs 11 and higher are identified by asterisks in the PROGRAM-ID print field. Tracing, however, continues.

Programmer Response: None.

IKF159I-* NO PROCEDURES TRACED.

Explanation: Abnormal termination has taken place before any COBOL statement with a procedure-name could be traced.

System Action: No tracing is done.

Programmer Response: Probable user error. If a trace is desired, recompile the program after inserting additional procedure-names.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF160I- IDENTIFIER NOT FOUND - ****.

Explanation: The **** identifier specified on a line-control card was not defined in the Data Division of the COBOL program.

System Action: The dump request on the line-control card for this identifier is ignored.

Programmer Response: Probable user error. Rewrite the line-control card indicated to include the required identifier.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF161I- CARD NUMBER NOT FOUND.

Explanation: The line-num parameter of the line-control card must correspond to the generated card number directly preceding that for the data card at which the formatted dump is to begin.

System Action: The line-control card with non-existent card number is skipped.

Programmer Response: Probable user error. Substitute the correct card number for the incorrect specification indicated.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF162I- VERB NUMBER NOT FOUND.

Explanation: The verb-num parameter of the line-control card specifies a verb number that does not exist in the line specified by the corresponding line-num parameter.

---

*Only the message number, and not the text of the message, is printed on the system output listing.

System Action: The line-control card with the non-existent verb number is skipped.

Programmer Response: Probable user error. Substitute the verb number desired.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF163I- NO ROOM TO DUMP.

Explanation: There is insufficient space for a symbolic dump to be generated.

System Action: A Data Division dump (and sometimes COBOL statement number message) is not given.

Programmer Response: Probable user error. Include an additional 22K in the REGION parameter of the EXEC statement when the SYMDMP option is specified.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF164I- I/O ERROR ON DEBUG FILE.

Explanation: The SYSUT5 file must be specified when symbolic dumping is requested.

System Action: The SYMDMP output is canceled for the program.

Programmer Response: Probable user error. Include an additional DD card for the SYSUT5 file.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF165I- WRONG DEBUG FILE FOR PROGRAM.

Explanation: An additional data set, SYSUT5, is required when symbolic dumping is requested.

System Action: SYMDMP output is canceled for the program.

Programmer Response: Probable user error. Include an additional DD card for the SYSUT5 file.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF166I- NO ROOM FOR DYNAMIC DUMPS.

Explanation: There is insufficient space for a symbolic dump to be generated.

System Action: Dynamic dumping, but not abnormal termination dumping, is canceled for the program.

Programmer Response: Probable user error. Include an additional 22K in the REGION parameter of the EXEC statement when the SYMDMP option is requested.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF167I- INVALID FILE-NAME.

Explanation: An additional data set, SYSUT5, is required when symbolic dumping is requested.

System Action: SYMDMP output is canceled for the program.

Programmer Response: Probable user error. Include an additional DD card for the SYSUT5 file.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF168I- UNSUCCESSFUL OPEN OF DEBUG FILE.

Explanation: A GO.SYSDBG DD card is required when SYMDMP is specified.

System Action: SYMDMP output is canceled for the program.

Programmer Response: Probable user error. Check for a missing or incorrectly punched DD statement.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF169I- MISSING PARAMETERS.

Explanation: A program requesting symbolic dumping must include both program-control cards and line-control cards with their several parameters.

System Action: The option with the missing parameter is ignored.

Programmer Response: Probable user error. Include the parameter(s) missing from the program-control/line-control cards.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF170I- INVALID OPTION.

Explanation: The program control card can include only the IBM-designated options.

System Action: Execution continues.

Programmer Response: Probable user error. Check for misspelled option names on the program control card before rerunning the job.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF171I- SUBSCRIPTING ILLEGAL

Explanation: The line-control card contains subscripted namesin the 'NAME1[THRU NAME2]' option. Subscripting is not permitted.

System Action: Subscripting is ignored. Every occurrence of the name is printed.

Programmer Response: Probable user error. Remove subscript from 'NAME1[THRU NAME2]' option.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF172I- ON PARAMETER TOO BIG.

Explanation: None of the parameters specified in the ON option of a line-control card can exceed 32767.

System Action: The number is reduced to 32767.

Programmer Response: Probable user error. Respecify the parameter indicated.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

Note: Messages IKF160I through IKF172I (except IKF170I) may appear interspersed among the SYMDMP control cards at the point at which the error is recognized. PROGRAM-ID is specified for messages IKF163I through IKF172I (except IKF170I). For messages IKF160I through IKF162I, the PROGRAM-ID is that of the nearest preceding program-control card, and the card/verb number of the corresponding line-control card is given instead. Messages IKF163I through IKF165I may also appear in the midst of the dump output if the error condition is not recognized until dumping begins.

IKF173I- SYMDMP/STATE/FLOW INTERNAL ERROR. EXECUTION CANCELLED.

Explanation: This message is issued when an exceptional input/output condition has occurred and no provisions were made to handle it within the COBOL program.

System Action: The job is canceled.

Programmer Response: Probable user error. Rereun the job. If the problem recurs, have the following available before calling IBM for programming support: source deck, control

cards, and execution-time output.

IKF174I- SYMDMP CANCELLED. NO CONTROL CARDS FOUND.

Explanation: Programs run with the SYMDMP option must include both object-time control cards and program control cards.

System Action: The SYMDMP option is canceled.

Programmer Response: Probable user error. Provide the necessary control cards before rerunning the job.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF175I- COBOL PROGRAM WITH DEBUGGING NOT FOUND.

Explanation: Either standard linkage conventions were not followed or the program in which the abnormal termination took place was a COBOL program with no debugging options or a non-COBOL program.

System Action: No debugging information is generated.

Programmer Response: Probable user error. Verify that standard linkage conventions are followed and then check for the possibility that the abnormal termination is in a COBOL program with no debugging options or a non-COBOL program.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and execution-time output.

IKF176I- NO SAVE AREA CHAIN. SYMDMP CANCELLED.

Explanation: Standard linkage conventions must be followed in a calling-called sequence.

System Action: SYMDMP output is canceled.

Programmer Response: Probable user error. Provide the necessary linkage code before rerunning.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and execution-time output.

IKF177I- USER ERROR FOUND. DEBUG OUTPUT FOLLOWS.

Explanation: This message is issued when an exceptional input/output condition has occurred and no provisions were made to handle it within the COBOL program.

System Action: Debugging information is generated and the job step is canceled.

Programmer Response: Probable user error. Rerun the job. If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and execution-time output.

IKF180I- DEBUG CANCELLED. UNABLE TO OPEN OUTPUT DATA SET.

Explanation: When the SYMDMP option is in effect, the SYSDBOUT DD data set must be requested.

System Action: All debugging options are canceled. The following message is issued at the console: DEBUG CANCELLED. UNABLE TO OPEN OUTPUT DATA SET.

Programmer Response: Probable user error. Provide a SYSDBOUT DD card before rerunning the job.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and execution-time output.

IKF181I- NO PCONTROL ENTRY FOUND FOR PROGRAM-ID NAMED. SYMDMP CANCELLED.

Explanation: PROGRAM-ID has not been found on the PCONTROL card. Possibly mispelled.

System Action: SYMDMP output is canceled for the program.

Programmer Response: Probable user error. Correct the spelling of the PROGRAM-ID on the PCONTROL card.

IKF182I- UNINITIALIZED OR INVALID BASE ADDRESS FOR DATA ITEM ABOVE

Explanation: Base locator for identifier has not been initialized (for example, an identifier in a record associated with an unopened FD).

System Action: Value of identifier is not printed.

Programmer Response: None.

IKF183I- SPACE NOT FOUND FOR THE COUNT CHAIN. CONTINUING.

Explanation: There is not enough space for the count chain.

System Action: The count output for the program is canceled for this entry into the program unit.

Programmer Response: Increase the size of the region and re-execute the program.

IKF184I- SPACE NOT FOUND FOR THE VERBSUM TABLE. CONTINUING.

Explanation: There is not enough space for the VERBSUM table.

System Action: Verb statistics are not printed.

Programmer Response: Increase the size of the region and re-execute the program.

IKF185I- COUNT OPTION CANCELED. NO CORE.

Explanation: There is not enough storage for the COUNT option.

System Action: The COUNT option is canceled.

Programmer Response: Increase the size of the region and re-execute the program.

IKF186I- COUNT OPTION CANCELED. UNABLE TO OPEN SYSCOUNT DATA SET.

Explanation: When the COUNT option is in effect, the SYSCOUNT data set must be available.

System Action: The COUNT option is canceled.

Programmer Response: Provide a SYSCOUNT DD card before rerunning the job.

IKF187I- INVALID COUNT TABLE ENTRY. NO STATISTICS.

Explanation: An entry in the count table is invalid.

System Action: User ABEND 187 will occur.

Programmer Response: This message should not occur. Have the following available before calling IBM for programming support: source deck, control cards, and compiler- and execution-time output.

IKF191I- MAXIMUM CARD NUMBER EXCEEDED. NUM OPTION CANCELLED.

Explanation: The NUM option is in effect and the maximum card number (999999) is given for a card that is not the last in the program.

System Action: The NUM option is canceled.

Programmer Response: Probable user error. Reassign card numbers so that only the last card in the program is given the number 999999.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and execution-time output.

IKF192I- SYMDMP OR TEST OPTION CANCELLED DUE TO NUM SEQUENCE ERROR.

Explanation: Either SYMDMP or TEST is in effect and the NUM option has been canceled because of other error conditions.

Programmer Response: Probable user error. Correct the errors indicated and rerun the job.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and execution-time output.

IKF193I- ERROR IN FLOW OF CONTROL--JOB CANCELLED

Explanation: In a multi-language environment in which COBOL is not the highest-level program in the run-unit, ILBOSTP0 was not called prior to calling the first COBOL subprogram.

System Action: Abend U3505 will occur (issued by ILBODBG1).

Programmer Response: Probable user error. Ensure that ILBOSTP0 is called prior to calling the first COBOL subprogram.

IKF400 through IKF411 These messages are produced by the Communication Job Scheduler (CJS) utility. Since this utility (written in COBOL) is subject to user modification, the text and meanings of these messages are also subject to change. Consult the current CJS program listing.

IKF430I-E nnc ENABLE TO WRITE STARTED MESSAGE TO CJS. RUN TERMINATED.

Explanation: The job-started messages required by the Communication Job Scheduler (CJS) from the job it has scheduled cannot be sent. The value nn is the CD status key associated with the error. The value c is the CD type: I for input, O for output.

System Action: The run unit is terminated.

Programmer Response: If nnc = 20I: the queue structure specified in the PARM field for the scheduled program cannot be accessed; ensure that the COBTPQD DD card is present (if required), and that a DD card for the queue responsible for the scheduling is also present.

If nnc = 220: ensure that the required DD card COBTPOUT is present.

If nnc = other: an I/O error has occurred.

Reschedule the job by reinitializing the CJS.

IKF431I-E nnc UNABLE TO WRITE ENDED MESSAGE TO CJS. RUN TERMINATED.

Explanation: Same as for IKF430.

System Action: The run unit is terminated.

Programmer Response: The Communication Job Scheduler (CJS) must be reinitialized in order to resume polling for the queues associated with the ended job.

IKF440I-E RECEIVE ISSUED WITHOUT NO DATA CLAUSE AFTER EGI ON BSAM-MODE INPUT QUEUE

Explanation: An implicit request for RECEIVE to wait for the next message to enter the input queue cannot be satisfied for an empty BSAM-mode input queue (no additional messages can be placed in the queue).

System Action: The run unit is terminated.

Programmer Response: Provide a no data clause for the RECEIVE statement, or do not attempt to read beyond EGI FOR BSAM-mode input queues.

IKF555I- INVALID SEPARATE SIGN CONFIGURATION

Explanation: The SEPARATE SIGN configuration is invalid on the data item.

Programmer Response: Probable user error. Correct the indicated statement and rerun the job.

If the problem recurs, have the following items available before calling IBM programming support: source deck, control cards, and compiler output.

IKF888I- UNSUCCESSFUL SORT FOR SORT-FILE-DDNAME.

Explanation: The Operating System Sort/Merge program has returned a nonzero code to the COBOL program and the user has not specified the special register SORT-RETURN.

Programmer Response: Probable user error. Specify the special register SORT-RETURN and rerun the job; do not assume that any portion of the sort (input or output procedure) has been performed.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

Operator Response: Probable user error. User should have indicated whether or not the job should be canceled. The user should not assume that any portion of the sort (e.g., Input or Output Procedure) has been performed.

IKF990D- AWAITING REPLY

Explanation: This message is generated by an ACCEPT statement with the FROM CONSOLE option.

Programmer Response: Not applicable.

Operator Response: If message is unexpected, then probable user error. Issue a REPLY command. The contents of the text field should be supplied by the programmer.

IKF991I- NO STORAGE AVAILABLE FOR WORKING STORAGE FOR ****

Explanation: The **** library subroutine has insufficient work space.

System Action: The job is canceled.

Programmer Response: Probable user error. Increase the region size and re-execute the program.

IKF992I- RECURSIVE CALL TO MODULE FROM MODULE ****.

Explanation: The module whose name is substituted for the asterisks in the message text has been called recursively.

System Action: COBOL does STOP RUN processing, returning to the original caller of its main program.

Programmer Response: Probable user error. Correct the recursive call situation and rerun the job.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF993I NO STORAGE AVILABLE FOR GETMAIN. INCREASE REGION SIZE AND RE-EXECUTE PROGRAM.

Explanation: A library subroutine has insufficient work space.

System Action: User ABEND 3440 will occur.

Programmer Response: Probable user error. Increase the region size and re-execute the program.

IKF994I GETMAIN/FREEMAIN REQUEST INVALID. PROGRAM EXECUTION TERMINATED.

Explanation: Library subroutine ILBOCMM0 has received a GETMAIN/FREEMAIN request from another library subroutine that it cannot recognize.

System Action: User ABEND 3440 will occur.

Programmer Response: This message should not occur. Have the following available before calling IBM for programming support: source deck, control cards, and compiler and execution-time output.

IKF999I UNSUCCESSFUL OPEN FOR ddname. PROGRAM EXECUTION TERMINATED.

Explanation: The data set identified by ddname could not be opened to implement an ACCEPT statement.

System Action: The run unit is terminated.

Programmer Response: Probable user error. Check for a missing or incorrectly-specified DD statement.

Diagnostic Messages -- MCS Considerations

All console messages issued by this compiler or its object code include parameters for multiple console support. A description of these parameters follows:

1. DISPLAY statement with the ON CONSOLE option (unnumbered) and all object time write-to-operator (IKF111I, IKF888I, IKF999I) messages are assigned:
   - a routing code of 2,11 (chief operator information/write-to-programmer)
   - a descriptor code of 7 (job status message).
2. STOP 'literal' (IKF000A) and ACCEPT statement with the FROM CONSOLE option (IKF990D) messages are assigned:
   - a routing code of 2,11 (chief operator information/write-to-programmer)
   - a descriptor code of 2 (immediate action required).
3. Compiler console messages (IKF0003I) are assigned:
   - a routing code of 2,11 (chief operator information/write-to-programmer)
   - a descriptor code of 7 (job status message).

Each message includes the write-to-programmer parameter and uses a system message block which then becomes unavailable until after the message is printed. Since a maximum number of these system message blocks must be specified by the installation's system programmer at the time of system generation, it is possible for the number of messages requiring system message blocks to exceed the number of blocks available. If this occurs, the programmer is warned of the condition, but all succeeding messages are ignored.

COBOL Object Program Unnumbered Messages

xxx...

Explanation: This message is written on the console and is recognizable because it is not preceded by a message code and action indicator. It is generated by a DISPLAY statement with the ON CONSOLE option. The message text is supplied by the object program and may indicate alternative action to be taken.

System Action: The job continues.

Operator Response: Operator response, if any is needed, is determined by the message text.

QUEUE ANALYZER MESSAGES

The COBOL queue-analyzer subroutine generates the messages that follow for the error conditions it diagnoses. Like the COBOL compiler, the queue analyzer sometimes issues warning messages and takes corrective action. Other conditions that indicate more serious errors result in termination of the queue structure description program.

For descriptions of the queue-analyzer subroutine and the queue structure description program, see the "Programming Techniques" part of this publication.

IKF700I-E MEMBER NOT WRITTEN FOR PRECEDING QUEUE STRUCTURE DUE TO ONE OR MORE ERRORS NOTED ABOVE.

Explanation: This message is always accompanied by other diagnostics. A partitioned data set cannot be created for the queue structure described because of serious errors.

System Action: The member is not written.

Programmer Response: Probable user error. Correct the syntax and other errors in the program before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF701I-E PDS ALREADY CONTAINS MEMBER WITH SAME NAME AS THE PREVIOUS QUEUE STRUCTURE. QUEUE NOT ADDED.

Explanation: Each member of a partitioned data set must have a unique name.

System Action: The queue is not added.

Programmer Response: Probable user error. Either assign a new name to the member being added or run the IEHPROGM utility program to scratch the dd member from the PDS; then rerun ILBOQSU0.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF702I-E RAN OUT OF SPACE IN PDS DIRECTORY WHILE ATTEMPTING TO ADD PRECEDING QUEUE STRUCTURE.

Explanation: There is not sufficient space in the partitioned data set to add the queue structure.

System Action: Compilation is continuing.

Programmer Response: Probable user error. Specify a larger value in the SPACE parameter.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF703I-W WARNING: QUEUE OR SUB-QUEUE NAME EXCEEDS 12 CHARACTERS IN LENGTH. FIRST 12 CHARACTERS USED AS NAME.

Explanation: The name of a queue or sub-queue name may contain a maximum of 12 characters.

System Action: The name is truncated to 12 characters.

Programmer Response: Probable user error. Substitute for the name indicated a queue name or sub-queue name that does not exceed 12 characters. Then recompile the program.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF704I-E UNRECOGNIZABLE KEYWORD IN ABOVE STATEMENT. STATEMENT DISREGARDED.

Explanation: The source statement immediately preceeding this error message contains at least one unrecognizable word.

System Action: The statement is discarded.

Programmer Response: Probable user error. Recode the

statement indicated and rerun the job.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF705I-E ONE OR MORE KEYWORDS OMITTED IN ABOVE STATEMENT. STATEMENT DISCARDED.

Explanation: Each sub-queue level must contain a reference to the sub-queue to be defined, the verb "IS", and the name to be assigned to the sub-queue.

System Action: The statement is discarded.

Programmer Response: Probable user error. Correct the source statement indicated before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF706I-W WARNING: QUEUE OR SUB-QUEUE NAME IS LESS THAN 12 CHARACTERS IN LENGTH AND ENDS IN COLUMN 80. ACCEPTED AS WRITTEN.

Explanation: The queue or sub-queue name ends in the 80th position in the record.

System Action: The statement is accepted as written.

Programmer Response: Verify that the queue name or sub-queue name is complete before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF707I-E SUB-QUEUE LEVEL NUMBER IN ABOVE STATEMENT IS NOT 1, 2, OR 3. STATEMENT DISCARDED.

Explanation: A queue structure must contain one, two, or three levels of sub-queues, written SUB-QUEUE-1, -2, or -3.

System Action: The statement is discarded.

Programmer Response: Probable user error. Substitute for the sub-queue level number indicated the numeral '1', '2', or '3', as appropriate.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF708I-E MORE THAN 200 STATEMENTS IN PRECEDING QUEUE STRUCTURE. SKIPPING TO NEXT QUEUE STRUCTURE OR END OF INPUT.

Explanation: A queue structure may not contain more than 200 statements.

System Action: The queue structure was not created.

Programmer Response: Probable user error. Recode the queue and sub-queue definition statements not to exceed a maximum of 200.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF709I-E ONE OR MORE LEVELS MISSING BETWEEN LEVEL IN ABOVE STATEMENT AND LEVEL DEFINED IN PRECEDING STATEMENT.

Explanation: When a lower level of the sub-queue hierarchy is described, all higher levels in that leg of the structure must be specified. For example, a SUB-QUEUE-3 statement cannot immediately follow a SUB-QUEUE-1 statement.

System Action: The statement is discarded.

Programmer Response: Probable user error. Supply the additional required sub-queues before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF710I-E FIRST CONTROL STATEMENT DOES NOT DEFINE A QUEUE-LEVEL NAME. SKIPPING TO FIRST "QUEUE IS" STATEMENT.

Explanation: The first statement in the queue structure description must specify a queue definition.

System Action: The compiler is skipping to the first queue definition.

Programmer Response: Probable user error. Rewrite the source statement(s) indicated so that the first control statement defines a queue-level name. Then recompile the program.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF711I-E DDNAME IN ABOVE STATEMENT NOT COMPLETED AT END OF 80 CHARACTERS. STATEMENT DISCARDED.

Explanation: A ddname in the statement indicated cannot contain more than 80 characters.

System Action: The statement is discarded.

Programmer Response: Probable user error. Replace the ddname indicated with one of the required length. Then recompile the program.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF712I-E DDNAME IN ABOVE STATEMENT EXCEEDS 8 CHARACTERS IN LENGTH. STATEMENT DISCARDED.

Explanation: A ddname cannot exceed 8 characters in length.

System Action: The statement is discarded.

Programmer Response: Probable user error. Substitute for the ddname indicated one that does not exceed 8 characters.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF713I-E A DDNAME IS SPECIFIED IN THE SECOND STATEMENT ABOVE, BUT THIS IS NOT A LOWEST-LEVEL SUB-QUEUE IN THE STRUCTURE.

Explanation: At execution time, the partitioned data set is described on a DD card, and the message control program table entries and the lowest-level sub-queue names are linked by DD cards.

System Action: The statement is discarded.

Programmer Response: Probable user error. Substitute for the invalid ddname indicated a name that matches a lowest-level sub-queue.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF714I-E DDNAME IN ABOVE STATEMENT BEGINS WITH A NUMBER. INVALID IN OS.

Explanation: In OS a ddname must begin with an alphabetic character.

System Action: The statement is discarded.

Programmer Response: Probable user error. Substitute for the invalid ddname one whose first character is alphabetic.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF715I-D UNABLE TO RUN COBOL QUEUE UTILITY DUE TO LACK OF SUFFICIENT CORE AVAILABLE. RERUN WITH LARGER REGION SIZE.

Explanation: There is not sufficient space to run the queue utility program.

System Action: The run is terminated.

Programmer Response: Probable user error. Specify a larger size in the REGION parameter.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF716I-D ERROR OCCURRED DURING OPENING QUEUE-STRUCTURE DATA SET. RUN TERMINATED.

Explanation: The DD card for the COBTPQD data set is either missing or incorrect.

System Action: The run is terminated.

Programmer Response: Probable user error. Check for a valid COBTPQD DD card before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF717I-D ERROR OCCURRED OPENING SYSIN DATA SET. RUN TERMINATED.

Explanation: Because the DD card for the SYSIN data set is either missing or incorrect, the SYSIN data set cannot be opened. The program cannot be compiled.

System Action: The run is terminated.

Programmer Response: Probable user error. Check for a valid SYSIN DD card before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF718I-E THE NAME ***** APPEARS MORE THAN ONCE AT THE SAME LEVEL IN A SINGLE LEG OF THE CURRENT STRUCTURE.

Explanation: Each queue or sub-queue defined for a queue structure must be unique.

System Action: The statement is discarded.

Programmer Response: Probable user error. Rewrite the queue or sub-queue name indicated so that it is unique.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF719I-E THE NAME **** AT SUB-QUEUE LEVEL **** IS SUPERFLUOUS SINCE THIS IS THE ONLY NAME AT THIS LEVEL.

Explanation: The sub-queue name at level **** is the only name at this level.

System Action: The statement is discarded.

Programmer Response: Probable user error. Check the queue structure for a one-legged path before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

APPENDIX L: RESOLVING COBOL COMPILER PROBLEMS

When a user encounters problems with the COBOL compiler or LCP, he should gather as much as possible of the following information before presenting the problem to the FE to get the problem resolved in the fastest possible way.

- Source deck--it is imperative that a source deck (or tape containing the source program) be supplied with any reported problem.
- COPY and BASIS libraries (if needed)
- Compilation listing (PMAP, DMAP, SXREF preferably)
- Abend listing (if necessary)
- Compiler options
- SIZE, BUF
- Compiler work file allocations
- Block sizes for compiler files
- Region/partition size
- System level and environment
- Type of hardware used
- PTFs applied to system
- PTFs applied to product
- ZAPs applied to product
- Any local fixes/changes made to product
- Any bypasses
- Does problem always occur or is it intermittent?
- If compiler failure--number of phase in which failure occurred (this can be obtained from first page of SYSUDUMP listing)
- If execution problem - any files used by the object program that would be needed to re-create the problem should be supplied when the problem is reported.

APPENDIX M: 3886 OPTICAL CHARACTER READER PROCESSING

The 3886 Optical Character Reader, Model 1* (herein referred to as "OCR"), is a general purpose online device that satisfies a broad range of data entry requirements. The OCR accepts documents from 3x3 inches to 9x12 inches in size. It can read machine-printed alphabetic, numeric and certain special characters in a wide variety of fonts, as well as hand-printed numeric characters.

The OCR reads documents one line at a time, under program control. Additional facilities, all under program control, include: document marking, line marking, document eject (with stacker selection, and line reread (for the current line, and with a different format description, if desired). (It is important in designing documents to remember that the OCR cannot reread previous lines; reading can only proceed from top to bottom on the document.)

The use of this appendix requires familiarity with the publications:

- IBM 3886 Optical Character Reader General Information Manual, Order No. GA21-9146.
- IBM 3886 Optical Character Reader Input Document Design Guide and Specifications Manual, Order No. GA21-9148.

In addition, the relevant portions of the following manuals should be referenced:

- OS/VS1 System Generation Reference, Order No. GC26-3791.
- OS/VS2 System Generation Reference, Order No. GC26-3792.
- OS/VS Program Planning Guide for IBM 3886 Optical Character Reader Model 1, Order No. GC21-5069.
- OS/VS Data Management Services Guide, Order No. GC26-3733.

---

*This device should not be confused with the 3336, Model 2, which is an offline Optical Character Reader, with output to tape. Information is included in this appendix, however, to help in processing tapes produced by the Model 2

OCR COBOL CAPABILITIES

The COBOL user can request nine different I/O operations with the OCR as follows:

- OPEN - Open a 3886 data set.
- CLOSE - Close a 3886 data set.
- READ - Read a line on a 3886 document.
- READO - Read a line on a 3886 document. READO must be followed by a WAIT.
- WAIT - Wait for completion of READO.
- SETDEV - Load the format record.
- MARKL - Mark a line.
- MARKD - Mark the current document.
- EJECT - Eject the current document.

OCR I/O REQUESTS

The I/O requests listed above cannot be issued directly; instead, the COBOL programmer must place the desired request in a COBOL data area of a specified format and then pass this information to a subroutine by a CALL statement with the USING option. The called subroutine handles the request and returns any requested information as well as certain additional information in case of an error. The CALL can have only the one parameter just described following the USING; if more than one is specified, the subroutine will return control immediately to the user with a value of 8 in the RETURN-CODE special register.

Figure 190 describes the necessary format of the COBOL data area that is passed to the subroutine. The data-name symbols used in Figure 190 are used for illustrative purposes only; any valid COBOL data-name symbols may be used. The data-name symbols used in Figure 186 describe the use of the fields in the parameter. To assist the user, an IBM-supplied source member, ILBOOCRD, can be included in the Data Division of the user program via a COPY statement. The contents of ILBOOCRD are listed in Figure 191. Another IBM-supplied source member,

ILBOOCRP, can be included in the Procedure Division via a COPY statement. ILBOOCRP, the contents of which are listed in Figure 192, can be used for doing OCR operations.

OCR STATUS KEY

After each operation, a status indicator is passed back to the COBOL program to indicate whether or not the operation was successful. This indicator is placed in bytes 22-23 of the data area passed to the subroutine. The possible cause, meanings, and programmer response to each value are listed in Figure 193.

DATA DIVISION Entry	Comments
01 OCR-FILE.	
02 OCR-FILE-ID PIC X(8)VALUE 'ddname'.	
02 OCR-FORMAT-RECORD-ID PIC X(8) VALUE 'DFR phase name'.	Format record name used for OPEN or SETDEV. Phase name is FRLGxxxx, where only the last 4 characters are used; if blank during OPEN, FRID on dd statement is used.
02 OCR-OPERATION PIC X(5).	Can be set to OPEN, CLOSE, READ, READO, WAIT, SETDV, MARKL, MARKD, or EJECT.
02 OCR-STATUS-KEY PIC 99.	Also referred to as exception code.
02 OCR-LINE-NUMBER PIC 99.	Line no. (0-33) passed to MARKL, READ, or EJECT.
02 OCR-LINE-FORMAT PIC 99.	Line format no. (0-63) passed to READ.
02 OCR-MARK PIC 99.	Mark option (1-15) passed to MARKL or MARKD.
02 OCR-STACKER PIC 9.	Pocket no. (1-2) passed to EJECT.
02 OCR-HEADER-RECORD PIC X(20)	Header information returned from READ or WAIT.
02 OCR-DATA-RECORD PIC X(130).	Data record returned from READ or WAIT.

Figure 190. Format of COBOL Parameter Data Area

```
************************* ILBOOCRD - OCR DATA DESCRIPTION ****************************
*****************************************************************************************
************************ O C R   3886   F I L E   F O R M A T ************************
*****************************************************************************************
01  OCR-FILE.
    05  OCR-FILE-CONTROL-AREA.
        10  OCR-FILE-ID                 PIC X(8)     VALUE 'DDNN3886'.
        10  OCR-FORMAT-RECORD-ID        PIC X(8)     VALUE 'FRLGDFR1'.
        10  OCR-OPERATION               PIC X(5)     VALUE 'OPEN '.
            88  OCRO-OPEN                            VALUE 'OPEN '.

            88  OCRO-CLOSE                           VALUE 'CLOSE'.
            88  OCRO-READ                            VALUE 'READ '.
            88  OCRO-READ-OVERLAPPED                 VALUE 'READO'.
            88  OCRO-WAIT                            VALUE 'WAIT'.
            88  OCRO-MARK-LINE                       VALUE 'MARKL'.
            88  OCRO-MARK-DOCUMENT                   VALUE 'MARKD'.
            88  OCRO-EJECT                           VALUE 'EJECT '.
            88  OCRO-SETDEV                          VALUE 'SETDV'.
        10  OCR-STATUS-KEY              PIC 99       VALUE 0.
*       (STATUS KEY CODES AND NAMES TO BE CHANGED)
            88  OCRS-SUCCESSFUL                      VALUE 00.
            88  OCRS-END-OF-FILE                     VALUE 10.
            88  OCRS-IO-ERRORS                       VALUE 30 THRU 39.
            88  OCRS-MISC-ERROR                      VALUE 30.

            88  OCRS-MARK-CHECK                      VALUE 31.
            88  OCRS-NONRECOVERY-ERROR               VALUE 32.
            88  OCRS-INCOMPLETE-SCAN                 VALUE 33.
            88  OCRS-MARK-AND-EQUIP-CHECK            VALUE 34.
            88  OCRS-PERMANENT-ERROR                 VALUE 39.
            88  OCRS-SPECIAL-ERRORS                  VALUE 90 THRU 99.
            88  OCRS-LOGIC-ERROR                     VALUE 92.
            88  OCRS-RESOURCE-UNAVAILABLE            VALUE 93.
            88  OCRS-INVALID-PARAMETER               VALUE 95.
            88  OCRS-INVALID-OPERATION               VALUE 99.
        10  OCR-LINE.
            15  OCR-LINE-NUMBER         PIC 99       VALUE 1.
            15  OCR-LINE-FORMAT         PIC 99       VALUE 1.
        10  OCR-MARK                    PIC 99       VALUE 0.
        10  OCR-STACKER                 PIC 9        VALUE 1.
```

Figure 191. IBM-supplied Data Division COPY Member (Part 1 of 2)

```
*
*    ******* HEADER AND DATA RECORD AREAS*******
*    FILLED IN BY SUCCESSFUL 'READ' AND/OR 'WAIT'.
*    (NOTE - 'READO' DOES NOT ALTER THESE AREAS)
*
05   OCR-HEADER-RECORD                                  VALUE ZEROS.
     10   ORCH-LINE-NUMBER              PIC 99.
     10   ORCH-LINE-FORMAT              PIC 99.
     10   ORCH-LINE-SCAN-COUNT          PIC 9.
     10   ORCH-LINE-STATUS              PIC 9.
          88   ORCH-LINE-GOOD                          VALUE 0.
          88   ORCH-LINE-BLANK                         VALUE 1.
          88   ORCH-LINE-GROUP-ERASE                   VALUE 3.
          88   ORCH-LINE-CRITICAL-ERR                  VALUE 2.
          88   ORCH-LINE-NON-CRITICAL-ERR              VALUE 4.
          88   ORCH-LINE-COMBINED-ERR                  VALUE 6.
          88   ORCH-LINE-INVALID                       VALUE 7.
          88   ORCH-END-OF-PAGE                        VALUE 5.
     10   ORCH-FIELD-INFO.
          15   ORCH-FILED-STATUS          PIC  9.      OCCURS 14.
               88   ORCH-FIELD-GOOD                    VALUE 0.
               88   ORCH-FIELD-REJECT-CHARS             VALUE 2.
               88   ORCH-FIELD-WRONG-LENGTH             VALUE 4.
               88   ORCH-FIELD-COMBINED-ERR             VALUE 6.
               88   ORCH-FIELD-BLANK                    VALUE 8.
               88   ORCH-FIELD-BLANK-SUP                VALUE 4.
05   OCR-DATA-RECORD.
     10   OCR-STANDARD-MODE-RECORD.
          15   OCR-STANDARD-FIELD-CHAR PIC X           OCCURS 130.
     10   OCR-IMAGE-MODE-RECORD
            REDEFINES OCR-STANDARD-MODE-RECORD.
          15   OCR-IMAGE-FIELD-LENGTH    PIC 99        OCCURS 14.
          15   OCR-IMAGE-FIELD-CHAR      PIC X         OCCURS 102.
************END OF 3886 DATA DIVISION COPY MEMBER************
```

Figure 191. IBM-supplied Data Division COPY Member (Part 2 of 2)

```
*******  ILBOOCRP - OCR 3886 PROCEDURES
*****************************************************************************
*******        O C R  3 8 8 6  P R O C E D U R E S        *******
*****************************************************************************
*     THE 3886 OCR SUBROUTINE USES OCR-FILE FIELDS AS FOLLOWS
*
*     ALL OPERATIONS REQUIRE
*         OCR-FILE-ID = THE UNIQUE NAME USED TO IDENTIFY THE FILE
*                   TO THE SUBROUTINE AND TO THE SYSTEM
*         OCR-OPERATION = THE CODE FOR THE REQUESTED OPERATION
*     ALL OPERATIONS RETURN
*         OCR-STATUS-KEY = RETURN CODE FOR VARIOUS OCCURRENCES
*
*     OCR-OPEN ('OPEN ') ALSO REQUIRES
*         OCR-FORMAT-RECORD-ID = LIBRARY NAME OF DFR TO LOAD
*     OCR-READ ('READ ') ALSO REQUIRES
*         OCR-LINE-NUMBER (1-33) = LINE TO READ (ON DOCUMENT)
*         OCR-LINE-FORMAT (1-63) = DLINT NUMBER (IN CURRENT DFR)
*         AND RETURNS (IF OCRS-SUCCESSFUL)
          OCR-HEADER-RECORD = HEADER RECORD, AS RETURNED BY THE 3886
*         OCR-RECOGNITION-RECORD = DATA FROM DOCUMENT, FROM 3886
*     OCR-READ-OVERLAPPED ('READO') HAS SAME REQUIREMENTS AS OCR-READ
*     OCR-WAIT('WAIT ') RETURNS SAME PARAMETERS AS OCR-READ
*     OCR-MARK-LINE ('MARKL') ALSO REQUIRES
*         OCR-LINE-NUMBER (1-33) = LINE TO MARK (ON DOCUMENT)
*         OCR-MARK (1-15) = SUM OF DESIRED MARK CODES (8421)
*
*     OCR-MARK-DOCUMENT ('MARKD') ALSO REQUIRES
*         OCR-MARK (1-15) = SUM OF DESIRED MARK CODES (8421)
*     OCR-EJECT ('EJECT') ALSO REQUIRES
*         OCR-POCKET (1-2) = STACKER TO SELECT (A OR B)
*         OCR-LINE-NUMBER (0-33) = NUMBER OF LINES ON DOCUMENT
*                   FOR VALIDATION (IF 0, NO VALIDATION WILL OCCUR)
*     OCR-SET-DEVICE ('SETDV') ALSO REQUIRES
*         OCR-FORMAT-RECORD-ID = LIBRARY NAME OF OFR TO LOAD
*
*NOTES
*     1.  THE TERMS DFR AND DLINT ARE USED TO REFER TO THE EXPANDED
*         CODE, IN LOADABLE FORM, OF THE RESPECTIVE SYSTEM MACROS.
*     2.  OCR-WAIT MAY BE REQUESTED AFTER, AND ONLY AFTER, A
*         SUCCESSFUL OCR-READ-OVERLAPPED REQUEST.  NO INTERVENING
*         I/O COMMANDS WILL BE ALLOWED ON THAT SAME FILE.
*     3.  THE PROCEDURES PROVIDED BELOW AUTOMATICALLY FILL IN
*         THE OCR-OPERATION FIELD, CALL THE SUBROUTINE, AND TEST
*         THE OCR-STATUS-KEY AFTER RETURN.  IF ANY EXCEPTIONAL
*         CONDITIONS OCCUR, THEY PASS CONTROL TO THE ROUTINE
*         OCR-EXCEPTION-ROUTINE, WHICH THE PROGRAMMER MUST PROVIDE.
*         THE PROGRAMMER MAY AVOID EXCEPTION ROUTINE INVOCATION BY
*         ADDING THE FOLLOWING PHRASE TO THE COPY STATEMENT:
*            REPLACING OCR-EXCEPTION-ROUTINE BY OCR-CALL-EXIT
*     4.  ALTHOUGH OCR-STATUS-KEY MAY INDICATE THAT THE DESIRED OPERATION
*         WAS UNSUCCESSFUL, THE VALIDITY OF THE DATA OBTAINED SHOULD
*         BE DETERMINED BY TESTING OCRH-LINE-STATUS
*****************************************************************************
```

Figure 192. IBM-supplied Procedure Division COPY Member (Part 1 of 2)

```
OCR-3886-PROCEDURES.
OCR-OPEN.
     MOVE 'OPEN ' TO OCR-OPERATION OF OCR-FILE.
     PERFORM OCR-CALL THRU OCR-CALL-EXIT.
OCR-CLOSE.
     MOVE 'CLOSE' TO OCR-OPERATION OF OCR-FILE.
     PERFORM OCR-CALL THRU OCR-CALL-EXIT.
OCR-READ.
     MOVE 'READ ' TO OCR-OPERATION OF OCR-FILE.
     PERFORM OCR-CALL THRU OCR-CALL-EXIT.
OCR-READ-OVERLAPPED.
     MOVE 'READO' TO OCR-OPERATION OF OCR-FILE.
     PERFORM OCR-CALL THRU OCR-CALL-EXIT.
OCR-WAIT.
     MOVE 'WAIT' TO OCR-OPERATION OF OCR-FILE.
     PERFORM OCR-CALL THRU OCR-CALL-EXIT.
OCR-MARK-LINE.
     MOVE 'MARKL' TO OCR-OPERATION OF OCR-FILE.
     PERFORM OCR-CALL THRU OCR-CALL-EXIT.
OCR-MARK-DOCUMENT.
     MOVE 'MARKD' TO OCR-OPERATION OF OCR-FILE.
     PERFORM OCR-CALL THRU OCR-CALL-EXIT.
OCR-EJECT.
     MOVE 'EJECT' TO OCR-OPERATION OF OCR-FILE.
     PERFORM OCR-CALL THRU OCR-CALL-EXIT.
OCR-SET-DEVICE.
     MOVE 'SETDV' TO OCR-OPERATION OF OCR-FILE.
     PERFORM OCR-CALL THRU OCR-CALL-EXIT.
OCR-CALL.
     CALL 'ILBDOCR0' USING OCR-FILE.
     IF NOT OCRS-SUCCESSFUL OF OCR-FILE.
       GO TO OCR-EXCEPTION-ROUTINE.
OCR-CALL-EXIT. EXIT.
**********END OF 3886 PROCEDURE DIVISION COPY MEMBER**********
```

Figure 192. IBM-supplied Procedure Division COPY Member (Part 2 of 2)

IMPLEMENTING AN OCR APPLICATION

Design and coding of the OCR aspects of an application may be accomplished in COBOL as follows:

1. Document design - prepare the OCR form that will be used for input, independently of the COBOL program.
2. Document description - code the DFR and DLINT macros to be used in reading the document(s), independently of the COBOL program.
3. COBOL file and record descriptions - code the COBOL data structures that correspond to the DLINT macros defined earlier. They should be defined as subordinate to the OCR-FILE area, which the programmer may COPY into the source program.
4. COBOL procedural code - code the COBOL source statements required to control the file, read lines, and recover from errors. COBOL provides a COPY member to simplify this file handling.

DOCUMENT DESIGN

Document design criteria are given in detail in the IBM 3886 Optical Character Reader, Input Document Design Guide and Specifications. The most important aspects of document design are:

1. The location of lines which can be read. These are identified by "timing marks"; lines not associated with timing marks are always ignored by the OCR. Note that lines may be almost anywhere on the document, and need not be at regular intervals.
2. The location of fields to be read. "Fields," strings of related characters, should be identified in document design. They will eventually required description, using the DFR and DLINT macros (see "Document Description").
3. The form identifier. This field should be a pre-printed code, useful for identifying one of many different forms. It should be at a common

location on the first (readable) line of each form. (This field can, of course, be ignored by programming or DLINT specification if desired; it should, however, be included in the form design, so as to allow for later form changes or additional batched forms without disruption of operations.)

DOCUMENT DESCRIPTION

Documents are described in the system with the DFR (Define Format Records) and DLINT (Define Line Type) macros.

The DFR macro identifies, by name, a collection of DLINT macros, and establishes various default field scanning options for them. As such, it is intended that each different DFR grouping will identify a different document, or a largely different way of scanning the same document (e.g., when typed entirely in a different font).

DFR and DLINT macros, after assembly and linkage editing, are preserved in loadable form until called for by the application program.

Status Key Value	Operations Causing Value	Meaning	Programmer Response
00	Any OCR I/O	Successful	Continue processing normally.
10	READ, WAIT, MARKL, MARKD, EJECT, SETDEV	End-of-file Close file.	Do end of file processing. See Note 1.
30	OPEN	Miscellaneous error	See Note 1.
31	EJECT	Mark Check	Attempt to re-read the line, or eject document and prepare to process the next document.
32	Any OCR I/O except OPEN and CLOSE	Non-recovery error	Eject the document and prepare to process the next document.
33	READ, WAIT	Incomplete scan	Re-read the line using either a different DLINT or an image-mode DFR.
34	EJECT	Mark Check and Equipment Check	See Note 2.

Notes

1. End-of-file occurs on the listed I/O commands when the operator has pressed the END-OF-FILE button, no documents remain in the read station, and no errors are outstanding. (EOF might also occur on OPEN but only following some unusual operator actions.) If the file is DD DUMMY, EOF is given only on READ and WAIT commands. Commands are checked for validity but no physical I/O requests are issued.

2. The noted errors represent serious I/O error conditions. No more I/O should be performed on the device after any of these errors are encountered. The program should, in general, indicate the error, do clean-up, and issue a STOP RUN.

3. The noted errors represent a serious programming error or a problem in the program environment. The program should indicate the error, do clean-up, and issue a STOP RUN.

Additional Note: WAIT and READ commands return data and header records only for the following codes: 00 (successful), 10 (EOF - if not a dummy file), 31 (mark check), and 33 (incomplete scan). For any other copies, the contents of the header and data record areas are unpredictable.

Figure 193. OCR STATUS KEY Values (Part 1 of 2)

Status Key Value	Operations Causing Value	Meaning	Programmer Response
39	Any OCR I/O except OPEN and CLOSE	Permanent error	See Note 2. This code indicates one of the following errors: Command Reject, Bus out check, Equipment Check, Non-initialized, RCP error, or Invalid Format.
92	Any OCR I/O	Logic error	See Note 3. This code represents an error in operation order: • OPEN issued on file already open. • File not open (all operations except OPEN). • WAIT issued but no READO in progress. • READO followed by an operation other than WAIT.
93	OPEN	Insufficient storage	See Note 3. This code indicates that a GETMAIN issued by the COBOL subroutine failed. The programmer should make certain that the REGION parameter specifies enough storage.
95	READ, READO, MARKL, MARKD, EJECT	Invalid Parameter	See Note 3. A parameter (except OCR-OPERATION) required by the last operation was invalid: too large or small, or contained invalid characters.
99	No OCR I/O	Unrecognizable operation	See Note 3. The OCR-OPERATION parameter contained an illegal operation code.

Notes:

1. End-of-file occurs on the listed I/O commands when the operator has pressed the END-OF-FILE button, no documents remain in the read station, and no errors are outstanding. (EOF might also occur on OPEN but only following some unusual operator actions.) If the file is DD DUMMY, EOF is given only on READ and WAIT commands. Commands are checked for validity but no physical I/O requests are issued.

2. The noted errors represent serious I/O error conditions. No more I/O should be performed on the device after any of these errors are encountered. The program should, in general, indicate the error, do clean-up, and issue a STOP RUN.

3. The noted errors represent a serious programming error or a problem in the program environment. The program should indicate the error, do clean-up, and issue a STOP RUN.

Additional Note: WAIT and READ commands return data and header records only for the following codes: 00 (successful), 10 (EOF - if not a dummy file), 31 (mark check), and 33 (incomplete scan). For any other copies, the contents of the header and data record areas are unpredictable.

Figure 193. OCR STATUS KEY Values (Part 2 of 2)

Each DLINT macro describes the scanning of a line, by field, in terms of: the starting and ending points of fields on a line (in tenths of an inch); the field lengths (in characters); the font code to be used (OCR-A, OCR-B, Gothic, or hand-printed numerics, all with various add special character suppression); field character delimiters (a character to end a field scan); and various additional options.

Note that the DLINT macro may specify either "standard mode" or "image mode."

Instandard mode, all DLINT options are valid, and the data record is of a fixed format, according to the field lengths in characters. In image mode, the field length and all EDIT keywords are invalid; the data record begins with 14 2-byte length parameters, indicating the length of the fields that follow. Because of this variable format in the data record, image mode should be used only in applications for which standard mode is unsuitable.

COBOL FILE AND RECORD DESCRIPTIONS

The file to be processed must be described in the Data Division according to the format in Figure 190 or the programmer may conveniently use the IBM-supplied Data Division COPY member (see Figure 191). In the IBM-supplied COPY member, all fields and codes are included, along with descriptive names and default values. The programmer need modify only those fields that are not appropriate for the particular application.

The file description ("OCR-FILE" in the COPY member) includes all fields that the programmer must provide to the subprogram, the OCR-STATUS-KEY returned by the subprogram, and fields that describe the header and data records returned directly by the device. (Note that the header and data records are not constructed under program control; they are not altered after reading, and thus their contents are fully described in the General Information manual.)

The COBOL record descriptions are based on the DLINT formats, either in image mode or standard mode. If the macro specifies standard mode scanning, the data record is returned in a fixed format according to the DLINT: fields contiguous, from left to right in the same order, each with a specified length in bytes. If the macro specifies image mode scanning, however, the field lengths are returned at the beginning of the data record, and fields vary in location within the data record. Because of this, image mode should be used only in cases for which standard mode is unsuitable.

The programmer may describe the data records to be read by the application program by following the Data Division COPY request with the statement: "05 dataname REDEFINES OCR-DATA-RECORD." and starting the structure of each record description with a level number greater than 5 (see Figure 198 for an example).

PROCEDURAL CODE

The 3886 file is processed by using CALL statements to the IBM-supplied routine ILBOOCR0 or by including the IBM-supplied Procedure Division COPY member ILBOOCRP.

ILBOOCRP (see Figure 192 for contents) provides paragraphs to perform, which set the appropriate operation code, CALL the subroutine ILBOOCR0, and passes control to a programmer supplied OCR-EXCEPTION routine if an exception occurs.

In general, the programmer must move parameter information to the file area (OCR-FILE), and then issue a PERFORM for the appropriate procedure. Figure 194 lists the permissible I/O requests and the fields that must be set before issuing the CALL; also included are the fields that receive information back from the subroutine upon completion of the request.

Exception Handling with ILBOOCRP

If an exception occurs, the COPY member passes control to the procedure-name OCR-EXCEPTION-ROUTINE. If operations are to be retained in this routine, the programmer should do so by using the CALL statement directly, and testing the OCR-STATUS-KEY value afterwards. Return from the OCR-EXCEPTION-ROUTINE would normally be to OCR-CALL-EXIT (after a successful retry or recovery). This will return control to the invoking PERFORM.

Function	Using Identifier	Set By User	Subroutine Returns
OPEN	OCR-FILE	OCR-FILE OCR-OPERATION OCR-FORMAT-RECORD-ID	OCR-STATUS-KEY
CLOSE	OCR-FILE	OCR-FILE-ID OCR-OPERATION	OCR-STATUS-KEY
READ	OCR-FILE	OCR-FILE-ID OCR-OPERATION OCR-LINE-NUMBER OCR-LINE-FORMAT	OCR-STATUS-KEY OCR-HEADER-RECORD OCR-DATA-RECORD
READO	OCR-FILE	OCR-FILE-ID OCR-OPERATION OCR-LINE-NUMBER OCR-LINE-FORMAT	OCR-STATUS-KEY
WAIT	OCR-FILE	OCR-FILE-ID OCR-OPERATION	OCR-STATUS-KEY OCR-HEADER-RECORD OCR-DATA-RECORD
MARKL MARKD REJECT	OCR-FILE	OCR-FILE-ID OCR-OPERATION OCR-LINE-NUMBER (MARKL or EJECT) OCR-MARK (MARKL or MARKD) OCR-STACKER (EJECT)	OCR-STATUS-KEY
SETDEV	OCR-FILE	OCR-FILE-ID OCR-FORMAT-RECORD OCR-FORMAT-RECORD-ID	OCR-STATUS-KEY

1. READ combines the functions of READO and WAIT. I/O overlap is not permitted within the issuing task.
2. A successful READO must be followed by a WAIT request for that same OCR-FILE area. Intervening I/O operations for that file are not permitted.
3. The WAIT function causes the active task to be placed in the wait condition, if necessary, until the preceding READO request is completed. The WAIT must be issued immediately following READO.

Figure 194. Requesting OCR Functions and Information Returned

SAMPLE PROGRAM

The sample program that follows consists of the document to be processed, the JCL to process the DFR and DLINT macro code, the COBOL source program, and the JCL to execute the program. Note that the IBM-supplied COPY members are used in the program.

A typical application for an optical character reader is processing insurance premiums. Figure 195 shows an insurance premium notice for the Standard Acme Life Insurance Company. The document has three lines of data to be read. The first line contains one field, the name of the policy holder. The second line contains four fields: the second line of the policyholder's address, the policy number, the premium amount due, and a code to be hand printed if the amount paid is different from the amount due. The third line contains one field that contains the amount paid if different from the amount due.

Format Record Assembly Example

To process documents like that in Figure 195, one format record is used. The format record must be created in a separate assembly. The coding necessary to create the format record is shown in Figure 196. The numbers at the left of the coding form correspond to those in text. Figure 197 shows the data, from the document shown in Figure 195, as it is received by the program.

1. The job control language (JCL) statements indicate that the job is an assembly. The output from the assembly is linkage-edited into SYS1.IMAGELIB with a member name of FRLGIPN. The format record is identified by IPN.

2. The DFR macro instruction specified the characteristics common to all lines on the document. The following information is provided:

   FONT=ANA1: The alphameric OCR-A font is used for reading any fields that do not have another font specified in the DLINT macro instruction field entries.

   REJECT=@: The commercial at sign (@) is substituted for any reject characters encountered.

   EDCHAR=(',','.'): The comma and period are removed from one or more fields as indicated in DLINT entries (line 2, field 3).

3. The DLINT macro instruction describes one line type in a format record described by the DFR macro instruction. The following information is provided about the first line:

   LFR=1,LINBEG=4: The first line on the document has a line format record number of 1. The first field to read from the line begins four tenths of an inch from the left edge of the document. The data record is in the standard mode; editing is performed on both fields on the line.

   FLD1=(32,20,NCRIT),EDIT1=HLBLOF: The first and only field on the line ends 3.2 inches from the left edge of the document, the edited data is placed in a 20-character field; the field is not considered critical. All leading and trailing blanks are removed, the data is left-justified, and the field is padded to the right with blanks.

The second line on the document is described as follows:

LFR=2,LINBEG=4: The second line on the document has a line format record number of 2. The first field read begins four tenths of an inch from the left edge of the document. The data record is in standard mode; editing is performed on all fields on the line.

FLD1=(30,20,NCRIT),EDIT1=HLBLOF: The first field on the line ends 3.0 inches from the left edge of the document, the edited data is placed in a 20 byte field; the field is not considered critical. All leading and trailing blanks are removed, the data is left justified, and the field is padded to the right with blanks.

FLD2=(42,5,NUMA),EDIT2=ALBNOF: The second field ends 4.2 inches from the left edge of the document, the edited data is placed in an eight-byte field, the field is critical. All leading and trailing blanks are removed from the field. The resulting field must be eight digits in length or a wrong length field indicator is set.

FLD3=(54,6,NUMA),EDIT3=HLBHIF,EDCHAR: The third field ends 5.4 inches from the left edge of the document, the edited data is placed in six-byte field, the field is critical. All leading and trailing blanks are removed, the data is right-justified, and the field is padded to the left with zeros. A comma, if present, and the decimal point are removed from the edited field.

FLD4=(62,1,NHP1),EDIT4=ALBHIF: The fourth field ends 6.2 inches from the left edge of the document, the edited data is placed in a one-byte field, the field is critical and is read using the numeric handprinting normal mode. All blanks are removed, the data is right-justified, and the field is padded to the left with zeros.

The third line on the document is described as follows:

LFR=3,LINBEG=45: The third line on the document has a line format record number of 3. The field to read begins 4.5 inches from the left edge of the document. The data record is in standard mode; editing is performed.

FLD1=(63,7,NHP1),EDIT1=ALBHIF: The field on this line ends 6.3 inches from the left edge of the document, the edited data is placed in a seven-byte field, the field is critical, and is read using the numeric handprinting normal mode. All blanks are removed, the data is right-justified, and the field is padded to the left with zeros.

FREND=YES: This is the format record end. No DLINT macros follow this statement.

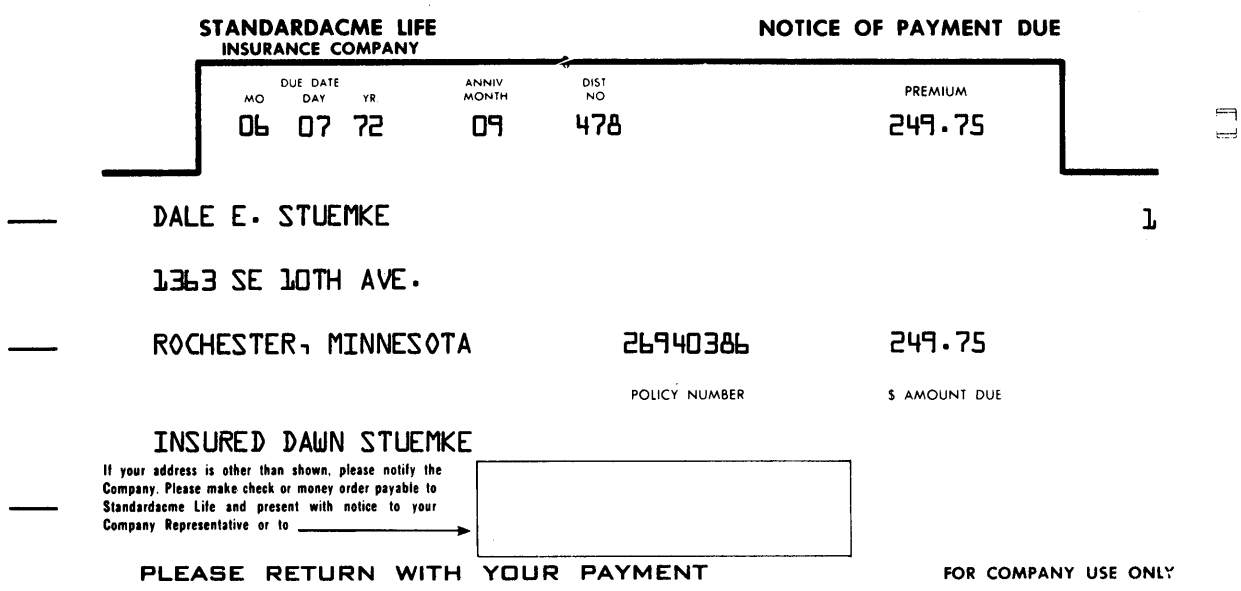


Figure 195. Sample Document

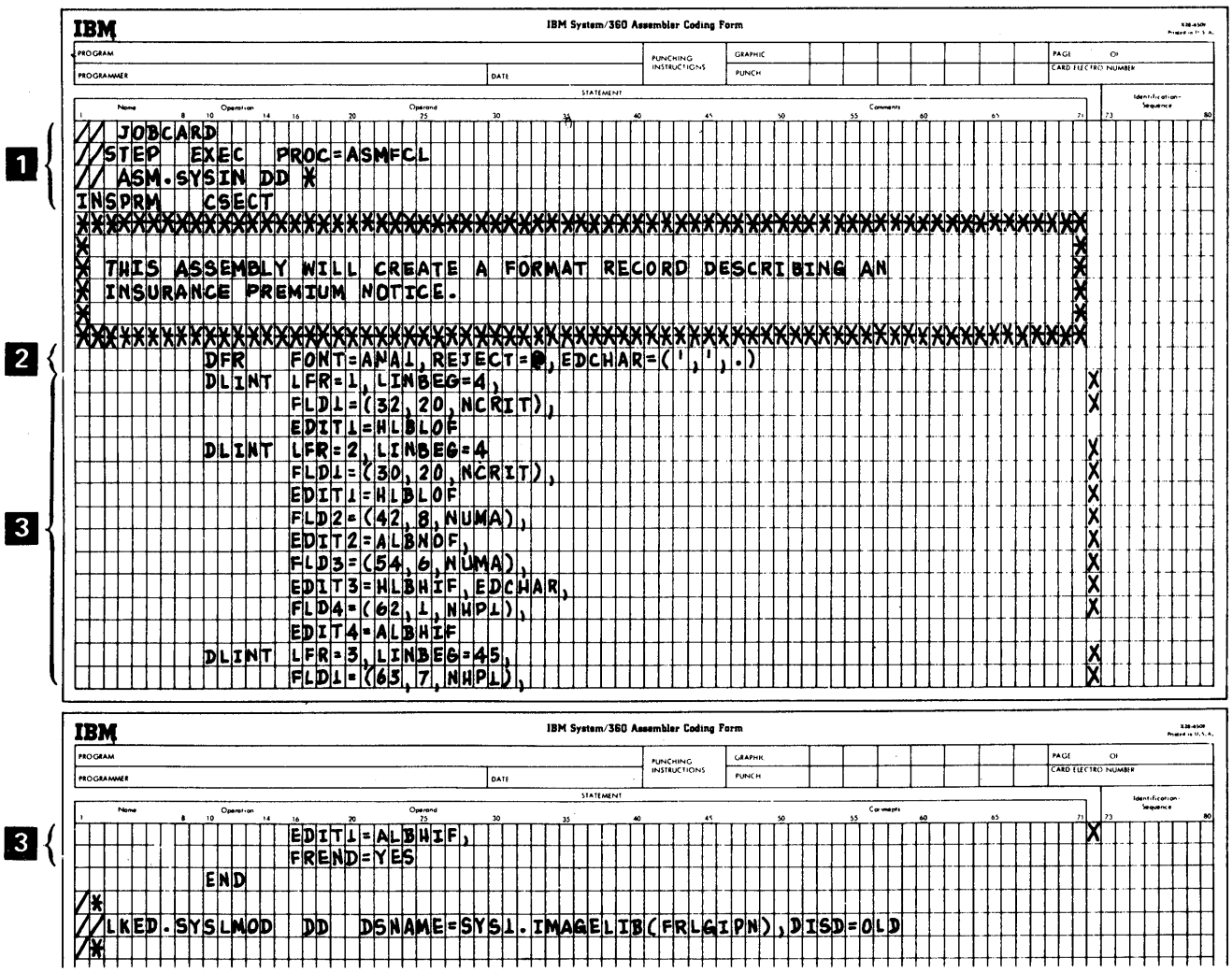
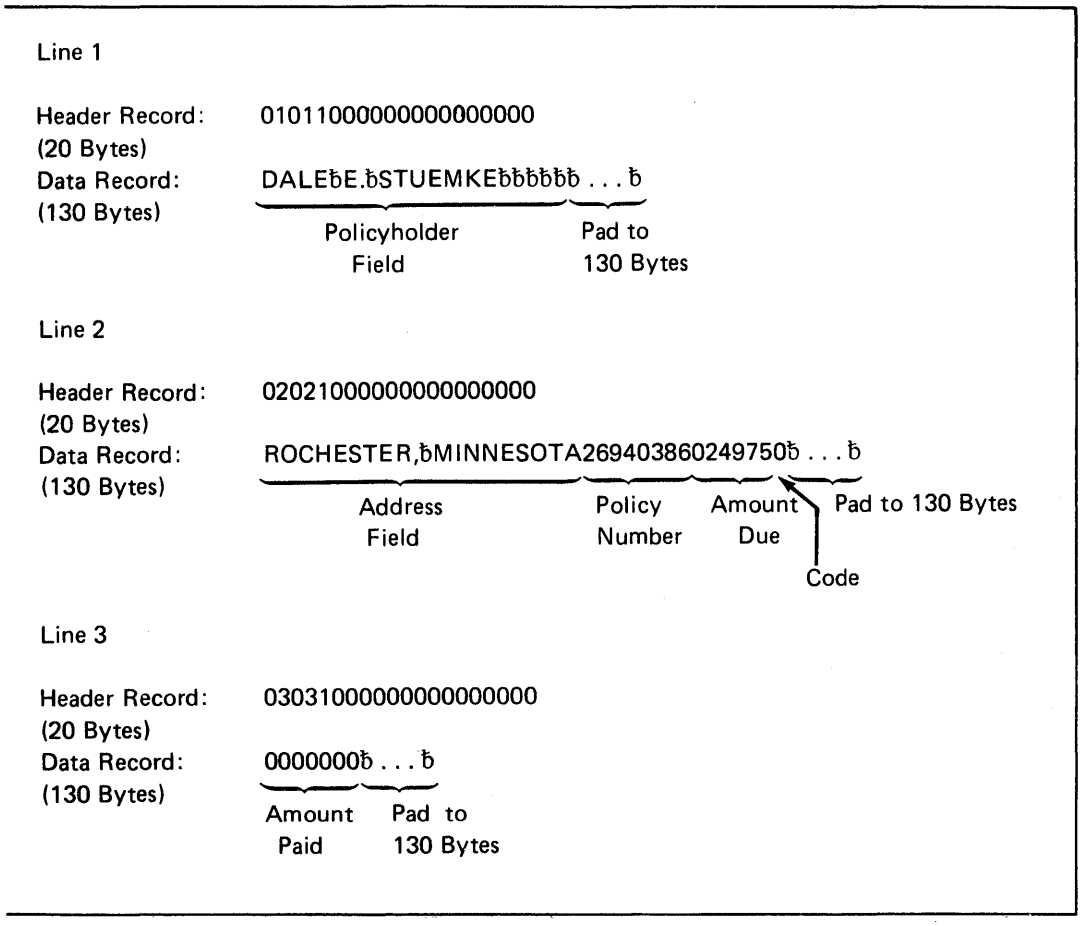


Figure 196. Format Record Assembly Coding Example



'igure 197. Sample Data

Figure 198 shows the COBOL source statements used to process the data in Figure 197. 'he JCL to run the program is also included in Figure 198.

```
*****************************************************************************
*********        S A M P L E   O C R   P R O G R A M         *********
*****************************************************************************
ID DIVISION.
  PROGRAM-ID. DOCLIST.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT PRINTER. ASSIGN TO SYS009-UR-1403-S.
DATA DIVISION.
FILE SECTION.
FD  PRINTER LABEL RECORDS ARE OMITTED.
01  PRINT-RECORD.
    05  FILLER                PIC X.
    05  PRINT-LINE            PIC X(130).
WORKING-STORAGE SECTION.
77  PRINT-CONTROL                PIC 9       VALUE 1.
77  MSG-PERMANENT-ERROR          PIC X(24)   VALUE
        'PERMANENT ERROR OCCURRED'.
77  MSG-MARK-CHECK               PIC X(19)   VALUE
        'MARK CHECK OCCURRED'.
77  MSG-MARK-AND-EQUIP-CHECK     PIC X(39)   VALUE
      'MARK CHECK AND EQUIPMENT CHECK OCCURRED'.
77  MSG-INCOMPLETE-SCAN          PIC X(24)   VALUE
        'INCOMPLETE SCAN OCCURRED'.
77  MSG-NONRECOVERY-ERROR        PIC X(26)   VALUE
        'NONRECOVERY ERROR OCCURRED'.
77  MSG-BAD-DATA                 PIC X(50)   VALUE
        'THE FOLLOWING LINE WAS MISREAD.  THE LINE HEADER ='.
01  MSG-TERMINATION.
    05  FILLER                   PIC X(44)   VALUE
        'TERMINAL ERROR OCCURRED - OCR-STATUS-KEY = '.
    05  MSG-TERM-STATUS-KEY      PIC XX.
```

Figure 198. Sample COBOL OCR Processing Program (Part 1 of 3)

```
01  OCR-FILE  COPY ILBDOCRD.
    05  NOTICE-OF-PAYMENT-DUE REDEFINES OCR-DATA-RECORD.
        10  LINE-1.
            15 L1-POLICYHOLDER-NAME            PIC X(20).
        10  LINE-2 REDEFINES LINE-1.
            15 L2-CITY-AND-STATE               PIC X(20).
            15 L2-POLICY-NUMBER                PIC X(8).
            15 L2-AMOUNT-DUE                   PIC 9(4)V99.
            15 L2-PAYMENT-VERIFY-CODE          PIC 9.
        10  LINE-3 REDEFINES LINE-1.
            15 L3-AMOUNT-PAID           PIC 9(5)V99.
PROCEDURE DIVISION.
    STOP RUN.
PIO-START.
    MOVE 'SYS010' TO OCR-FILE-ID.
    MOVE 'FORMAT' TO OCR-FORMAT-RECORD-ID.
    PERFORM OCR-OPEN.
    OPEN OUTPUT PRINTER.
PIO-HEAD.
    MOVE ALL '*' TO PRINT-LINE.
    PERFORM PRINT-ROUTINE.
    MOVE 1 TO OCR-STACKER.
PIO-READ.
    PERFORM OCR-READ.
    IF OCRS-NONRECOVERY-ERROR,          GO TO PIO-EOP-ERR.
    IF ORCH-LINE-GOOD,                  GO TO PIO-GOOD.
    IF OCRH-LINE-BLANK,                 GO TO PIO-GOOD.
    IF OCRH-LINE-NON-CRITICAL-ERR,      GO TO PIO-GOOD.
    IF ORCH-END-OF-PAGE,                GO TO PIO-EOP.
****IF ORCH HAS ANY OTHER CODE, CONSIDER THE DATA AS BAD ****
PIO-BAD.
    MOVE MSG-BAD-DATA TO PRINT-LINE.
    PERFORM PRINT-ROUTINE.
    MOVE 2 TO OCR-STACKER.
PIO-GOOD.
    MOVE OCR-DATA-RECORD TO PRINT-LINE.
    PERFORM PRINT-ROUTINE.
    MOVE 1 TO PRINT-CONTROL.
    ADD 1 TO OCR-LINE-NUMBER, OCR-LINE-FORMAT.
    IF ORCH-LINE-NUMBER IS LESS THAN 3, GO TO PIO-READ.
PIO-EOP.
    MOVE 3 TO OCR-LINE-NUMBER.
    PERFORM OCR-EJECT.
PIO-EOP-ERR.
    MOVE 1 TO OCR-LINE-NUMBER, OCR-LINE-FORMAT.
    MOVE 3 TO PRINT-CONTROL.
    GO TO PIO-HEAD.
```

Figure 198. Sample COBOL OCR Processing Program (Part 2 of 3)

```
******  EXCEPTION PROCESSING ROUTINE **************
OCR-EXCEPTION-ROUTINE.
    IF OCRS-END-OF-FILE,   GO TO P20-EOF.
    IF OCRS-MARK-CHECK,
        MOVE MSG-MARK-CHECK TO PRINT-LINE,
        GO TO P20-RETURN.
    IF OCRS-NONRECOVERY-ERROR,
        MOVE MSG-NONRECOVERY-ERROR TO PRINT-LINE,
        GO TO P20-RETURN.
    IF OCRS-INCOMPLETE-SCAN,
        MOVE MSG-INCOMPLETE-SCAN TO PRINT-LINE,
        GO TO P20-RETURN.

    IF OCRS-MARK-AND-EQUIPMENT-CHECK,
        MOVE MSG-MARK-AND-EQUIP-CHECK TO OCR-LINE,
        GO TO P20-PRINT-EOF.
    IF OCRS-PERMANENT-ERROR,
        MOVE MSG-PERMANENT-ERROR TO PRINT-LINE,
        GO TO P20-PRINT-EOF.
******  IF NONE OF THE ABOVE ERRORS, GIVE TERMINATION MESSAGE ******
    MOVE OCR-STATUS-KEY TO MSG-TERM-STATUS-KEY.
    MOVE MSG-TERMINATION TO PRINT-LINE.
    GO TO P20-PRINT-EOF.
P20-RETURN.
    PERFORM PRINT-ROUTINE.
    GO TO OCR-CALL-EXIT.
P20-PRINT-EOF.
    PERFORM PRINT-ROUTINE.
P20-EOF.
    PERFORM OCR-CLOSE.
    CLOSE PRINTER.
    STOP RUN.
PRINT-ROUTINE.
    WRITE PRINT-RECORD AFTER ADVANCING PRINT-CONTROL.
OCR-COPIED-PROCEDURES.  COPY ILBDOCRP.
/*
//LKED.SYSIN DD*
    ENTRY DOCLIST
/*
//GO.SYSPRINT DD SYSOUT=A
//GO.SYSUDUMP DD SYSOUT=A
//GO.SYS010  DD DUMMY
```

Figure 198. Sample COBOL OCR Processing Program (Part 3 of 3)

PROCESSING TAPES FROM THE 3886 OCR, MODEL 2

Tape records produced from the IBM 3886, Model 2, are almost identical in format to the header and data records returned by the Model 1. The main differences between the records are:

1. The Model 2 tape contains a document trailer record after the line output records for each document. The content of this trailer record differs from that of line output records.
2. The codes used in certain fields of the header record differ between the two models.

Because of the similarity, however, the Data Division COPY member defined for the Model 1 may be tailored to describe the Model 2 tape records. To do so, the programmer should punch out the COPY member, modify it according to the installation requirements, and recatalog it.

Specific information on the formats and contents of the Model 2 tape records is contained in the IBM 3886 Optical Character Reader, General Information Manual (GA21-9146).

As an aid to readers, this index contains not only the conventional references to material in this Programmer's Guide, but also references to material that can be found in the COBOL language manual. Entries having page numbers refer to pages in this manual. Entries having only asterisks point to IBM VS COBOL for OS/VS (GC26-3857). For example:

```
Able  347
Baker*
Charlie  89, *
Dog  23, 106
Easy
  additional information*
  sub-one  33
  sub-two  400
```

These entries would indicate that information on the subject "Able" is available on page 347 of this manual, and information on "Dog" is available in this manual on pages 23 and 106. Information on "Baker" does not exist in this manual; it can nevertheless be found in the language manual (by looking up "Baker" in that manual's index). The entry for "Charlie" indicates that some information is available in this Programmer's Guide (on page 89), and additional information can be found in the language manual (again, by looking up "Charlie" in its index). The entries for "Easy" indicate that its "sub-one" and "sub-two" aspects can be found in this manual (on pages 33 and 400, respectively); additional information on "Easy" is also available in the language manual.

*Consult VS COBOL for OS/VS (GC26-3857)

*Consult VS COBOL for OS/VS (GC26-3857)

*Consult VS COBOL for OS/VS (GC26-3857)

*Consult VS COBOL for OS/VS (GC26-3857)

*Consult VS COBOL for OS/VS (GC26-3857)

*Consult VS COBOL for OS/VS (GC26-3857)

*Consult VS COBOL for OS/VS (GC26-3857)

*Consult VS COBOL for OS/VS (GC26-3857)

*Consult VS COBOL for OS/VS (GC26-3857)

*Consult VS COBOL for OS/VS (GC26-3857)

*Consult VS COBOL for OS/VS (GC26-3857)

*Consult VS COBOL for OS/VS (GC26-3857)

*Consult VS COBOL for OS/VS (GC26-3857)

*Consult VS COBOL for OS/VS (GC26-3857)

*Consult VS COBOL for OS/VS (GC26-3857)

OS/VS COBOL Programmer's Guide (File No. S370-24) Printed in U.S.A. SC28-6483-2

SC28-6483-2