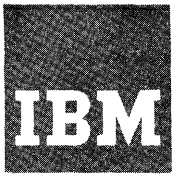
File No. S360-24
Order No. GC28-6399-2

OS

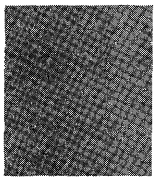
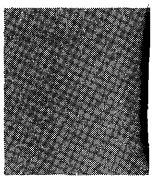
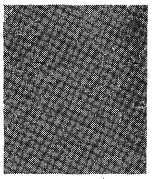
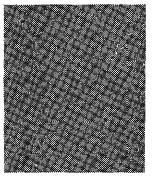
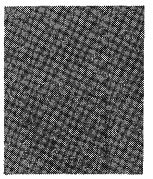


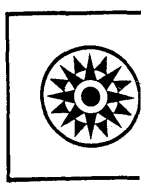
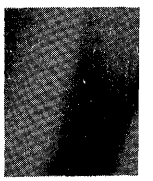
Systems Reference Library

IBM OS Full American National Standard COBOL Compiler and Library, Version 2 Programmer's Guide

Program Numbers 360S-CB-545
360S-LM-546

This publication describes how to compile an American National Standard COBOL X3.23-1968 program using Version 2 of the OS Full American National Standard COBOL compiler. It also discusses how to link-edit or load and execute the program under control of the Operating System. There is a description of the output of each of these steps, i.e., compile, link-edit, load, and execute. In addition, there is an explanation of the features of the compiler and available options of the operating system. Note that American National Standard COBOL was formerly known as USA Standard COBOL.

Third Edition (July 1972)

This is a major revision of, and makes obsolete, GC28-6399-1 and Technical Newsletters GN28-0422, GN28-0437, and GN28-0473.

This edition corresponds to Release 21.6 of the IBM Operating System.

Changes are continually made to the specifications herein; any such changes will be reported in subsequent revisions or Technical Newsletters. Before using this publication in connection with the operation of IBM systems, refer to the latest SRL Newsletter, Order No. GN20-0360, for editions that are applicable and current.

Requests for copies of IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form for readers' comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, Programming Publications, 1271 Avenue of the Americas, New York, New York 10020. Comments become the property of IBM.

The purpose of this publication is to enable programmers to compile, linkage edit, and execute, or compile and load American National Standard COBOL compiler Version 2 programs under control of the IBM System/360 Operating System. The COBOL language is described in the publication IBM OS Full American National Standard COBOL, Order No. GC28-6396, which is a corequisite to this publication.

Programmers who are familiar with the operating system and wish to know how to run COBOL programs should read "Job Control Statements" and "Data Set Requirements" under "Job Control Procedures," and "Output." These chapters provide information about the preparation of COBOL programs for processing by the operating system.

Programmers who are unfamiliar with the concepts of the Operating System should read "Introduction," "Job Control Procedures," "Checklist for Job Control Procedures," and "Using Cataloged Procedures" in addition to the sections listed above.

The chapters "Program Checkout" and "Programming Techniques" are of special interest, since they contain information about debugging and efficient programming. Other chapters discuss optional features of the language and the operating system. Some chapters include introductory information about features of the operating system that are described in detail in other publications.

The machine configuration required for system operations is described in the chapter "Machine Considerations."

Wider and more detailed discussions of the operating system are given in the following publications:

IBM System/360 Operating System: Concepts and Facilities, Order No. GC28-6535

IBM System/360 Operating System: Job Control Language Reference, Order No. GC28-6704

IBM System/360 Operating System: Job Control Language Charts, Order No. GC28-6632

IBM System/360 Operating System: System Programmer's Guide, Order No. GC28-6550

IBM System/360 Operating System: Linkage Editor and Loader, Order No. GC28-6538

IBM System/360 Operating System: Supervisor Services, Order No. GC28-6646

IBM System/360 Operating System: Data Management Services, Order No. GC26-3746

IBM System/360 Operating System: Supervisor and Data Management Macro Instructions, Order No. GC28-6647

IBM System/360 Operating System: Sort/Merge, Order No. GC28-6543

IBM System/360 Operating System: Utilities, Order No. GC28-6586

IBM System/360 Operating System: System Generation, Order No. GC28-6554

IBM System/360 Operating System: Messages and Codes, Order No. GC28-6631

IBM System/360 Operating System: Programmer's Guide to Debugging, Order No. GC28-6670

IBM System/360 Operating System: Storage Estimates, Order No. GC28-6551

ILLUSTRATIONS

FIGURES

TABLES

An American National Standard COBOL program can be processed by the IBM System/360 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 the system catalog of data sets. 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 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 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.

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 one or more object modules, to form one load module.

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

LOADING

The Loader is a service program that processes COBOL 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 IBM System/360 Operating System: Linkage Editor and Loader, where a discussion of invoking the Loader can be found in "Using the Cataloged Procedures."

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 Operating System offers two control programs. These are Multiprogramming with a Fixed Number of Tasks (MFT) and Multiprogramming with a Variable Number of Tasks (MVT).

MULTIPROGRAMMING WITH A FIXED NUMBER OF TASKS

The multiprogramming with a fixed number of tasks (MFT) 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 MFT 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 MFT control program, see the publication IBM System/360 Operating System: Storage Estimates.

MULTIPROGRAMMING WITH A VARIABLE NUMBER OF TASKS

The multiprogramming with a variable number of tasks (MVT) control program divides storage into areas called regions. Like MFT, the MVT control program uses a priority scheduling system and provides for concurrent execution of up to 15 jobs. In addition, the MVT control program provides for assignment of storage regions on a variable basis according to a region code.

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, and regulate the display of job scheduler messages. With priority schedulers, additional parameters are used to assign job priority, to request a specific class for job scheduler messages, to specify the amount of main storage to be allocated to the job, and to 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, and pass control information to a processing program. With priority schedulers, additional parameters 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 "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 certain jobs.

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 American National Standard 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, and SYSUT4 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, and SYSUT4 are &&UT1, &&UT2, &&UT3, and &&UT4, 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 three 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. 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. If SYSOUT=B is to be used, the unit name SYSCP must be specified at system generation.

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. Catalog 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. "Using Cataloged Procedures" provides information about using and modifying cataloged procedures.

"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 IBM System/360 Operating System: Job Control Language 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.

Table 1. Control Statements

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 must 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

CONTROL STATEMENTS

The COBOL programmer uses the control statements shown in Table 1 to compile, linkage edit, and execute programs.

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.

Priority schedulers process complete jobs according to their relative priority, and available system resources. Systems that provide multiprogramming (the MFT or MVT environments) use priority schedulers.

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 3.

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 3. 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 4, 5, and 7).

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 4, 5, and 7. 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 comments by following the interrupted field with at least one blank.
3. Optionally, code any nonblank character in column 72. If a character is not coded in column 72, the job scheduler treats the next statement as a continuation statement as long as the conventions outlined in items 4 and 5 are observed.
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 4, 5, 6, and 7 to improve the readability of control statement definitions. In actual statements, blanks would be interpreted as delimiters.

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=(nnnnnxK[,nnnnnyK])]
		[ROLL=(x,y)]
		[TYPRUN=HOLD]
		[RD=request]
		[RESTART=({*, stepname, stepname.procstepname} [,checkid])]

[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.

Figure 4. JOB Statement

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. For priority scheduling systems: job priority assignment, job scheduler message class, and for the MVT environment, main storage region size.

Figure 4 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.

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. As a system generation option with sequential schedulers, the account number can be established as a required subparameter. With priority schedulers, 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.

Reference:

- To write an accounting routine that processes job accounting information, see the section "Adding an Accounting Routine to the Control Program" of the publication IBM System/360 Operating System: System Programmer's Guide.

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 with sequential schedulers, the programmer's name can be established as a required parameter. With priority schedulers, the requirement can 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.

Reference:

- To write a routine that processes the programmer-name, see the section "Adding an Accounting Routine to the Control Program" of the publication IBM System/360 Operating System: System Programmer's Guide.

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

NR -- to request that the CHKPT macro instruction be allowed to establish a checkpoint, but not to permit automatic restart

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:

- When using a system with MVT or MFT, 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 IBM System/360 Operating System: Supervisor Services.

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.

```
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 IBM System/360 Operating System: Supervisor Services.

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
```

Replace the term "jobclass" with an alphabetic character A through O. The use of this parameter and the meaning of the character A through O are to be determined by each installation.

If the CLASS parameter is omitted, or CLASS=A is coded, the default job class of A is assigned to the job.

Note:

- If an installation provides time-slicing facilities in a system with MFT, 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 MVT, 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, or coded MSGCLASS=A, job scheduler messages are routed to the standard output class, A.

Reference:

- For a more detailed discussion of output classes, see the publication IBM System/360 Operating System: Operator's Reference, Form GC28-6691.

Specifying Main Storage Requirements for a Job (REGION)

(MVT only)

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 amount of main storage to be allocated to the job, and the storage hierarchy or hierarchies in which the space is to be allocated. This request should be made only if main storage hierarchy support has been specified during system generation. If an IBM 2361 Core Storage, Model 1 or 2, is present in the system, processor storage is referred to as hierarchy 0 and 2361 core storage is referred to as hierarchy 1. If 2361 Core Storage is not present but main storage hierarchy support was specified in system generation, a two-part region is established in processor storage when a region is defined to exist in two hierarchies. The two parts are not necessarily contiguous.

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 one to five digits and cannot exceed 16383.

To request a region size and the hierarchy desired, the term nnnnnx is replaced with the number of contiguous 1024-byte areas to be allocated to the job in hierarchy 0; the term "nnnnny" is replaced with the number of contiguous

1024-byte areas to be allocated in hierarchy 1, e.g., REGION=(60K,200K). When only processor storage is used to include hierarchies 0 and 1, the combined values of nnnnnx and nnnnny cannot exceed 16383. If 2361 Core Storage is present, nnnnnx cannot exceed 16383 and, for a 2361 model 1, nnnnny cannot exceed 1024, or 2048 for a 2361 model 2. Each value specified should be an even number. (If an odd number is specified, the system treats it as the next higher even number.)

If storage is requested only in hierarchy 1, a comma must be coded to indicate the absence of the first subparameter, e.g., REGION=(,200K). If storage is requested only in hierarchy 0, or if hierarchy support is not present, the parentheses need not be coded, e.g., REGION=70K.

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.

- If main storage hierarchy support is not included but regions are requested in both hierarchies, the region sizes are combined and an attempt is made to allocate a single region from processor storage. If a region is requested entirely from hierarchy 1, an attempt is made to allocate the region from processor storage.

- For information on storage requirements to be considered when specifying a region size, see the publication IBM System/360 Operating System: Storage Estimates.

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
```

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.

Specifying Additional Storage (ROLL) (MVT only)

To allocate additional main storage to a job step whose own region does not contain any more available space, code the keyword parameter in the operand field of the JOB statement.

```
ROLL=(x,y)
```

In order to allocate this additional space to a job step, another job step may have to be rolled out, i.e., temporarily transferred to secondary storage. When x is replaced with YES, each of the programmer's job steps can be rolled out; when x is replaced with NO, the job steps cannot be rolled out. When y is replaced with YES, each job step can cause rollout; when y is replaced with NO, the job steps cannot cause rollout. If additional main storage is required for the job's steps, YES must be specified for y. If this parameter is omitted, ROLL=(YES,NO) is assumed. ROLL parameters can also be coded in EXEC statements, but are superseded by a ROLL parameter coded in the JOB statement.

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. For priority scheduling systems: 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 5 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 or if the step is going to be part of a cataloged procedure. Each stepname in a job or procedure must be unique.

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.

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."

Name	Oper-ation	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 } = nnnnnxK[,nnnnnyK]]
		[{ ROLL ROLL.procstep } = (x,y)]
		[{ RD RD.procstep } = request]
		[{ DPRTY DPRTY.procstep } = (value 1, value 2)]

[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 5. EXEC statement

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
- COPY member-name is specified and no SYSLIB statement is included
- Required device not available
- Not enough core 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.

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 three significant characters 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. Table 2 lists the significant characters for each

option (see "Options for the Compiler" for an explanation of each).

Table 2. Significant Characters for Various Options

Option	Significant Characters
LINECNT	CNT
SEQ	SEQ
FLAGE(W)	LAG,LAGW
SIZE	SIZ
BUF	BUF
SOURCE	SOU
DECK	DEC
LOAD	LOA
SPACE	ACE
DMAP	DMA
PMAP	PMA
SUPMAP	SUP
CLIST	CLI
TRUNC	TRU
APOST	APO
QUOTE	QUO
XREF	XRE
LIB	LIB
VERB	VER
ZWB	ZWB

Options for the Compiler

The IBM-supplied default options indicated by an underscore in the following discussion can be changed within each installation at system generation time. The format of the PARM parameter is illustrated in Figure 6.

Note:

- 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'.

SIZE=yyyyyyy
: indicates the amount of main storage, in bytes, available for compilation (see "Machine Considerations").

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, 80K is 81,920 decimal bytes.

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, verb references, and the location of the first generated instruction for each verb. CLIST and PMAP are mutually exclusive options.

Note: In nonsegmented programs, verbs are listed in source order. In segmented programs, verbs are listed in source order within each segment, with the root segment last. If the VERB option is specified, the verb-name is printed out. Otherwise, only the verb number (VERB1, VERB2, and so on) is printed out.

DMAP
NODMAP
: indicates whether or not a glossary is to be listed.

PMAP
NOPMAP
: indicates whether or not register assignments, global tables, literal pools and an assembler language expansion of the source modules are to be listed. CLIST and PMAP are mutually exclusive options.

LIB
NOLIB
: indicates that BASIS and/or COPY statements are in the source program. If either COPY or BASIS is present, LIB must be in effect. If COPY and/or BASIS statements are not present, use of the NOLIB option yields more efficient compiler processing.

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.

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.

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. The first three lines of the output listing are for the compiler headings. (For example, if nn=55 is specified, then 52 lines are printed on each page of the output listing.)

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 diagnostic messages are to be listed, but the warning messages are not to be listed.

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 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
: is an option that applies only to COMPUTATIONAL receiving fields in MOVE statements and arithmetic expressions. If TRUNC is specified, extra code is generated to truncate the final intermediate result of the arithmetic expression, or the sending field in the MOVE statement, to the number of digits specified in the PICTURE clause of the COMPUTATIONAL receiving field. If NOTRUNC is specified, the compiler assumes that the data being manipulated conforms to PICTURE and USAGE specifications. The compiler then generates code to manipulate the data based on the size of the field in core (halfword, etc.). TRUNC conforms to the American National Standard, while NOTRUNC leads to more efficient processing. This will occasionally cause dissimilar results for various sending fields because of the different code generated to perform the operation.

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.

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 the order in which they are referenced. Use of the XREF option considerably increases compile time. NOXREF will suppress any cross-reference 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.

Note: The default option cannot be changed at system generation time.

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

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 6. 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 IBM System/360 Operating System: 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 SYSPRINT data set. If the SYSPRINT 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.

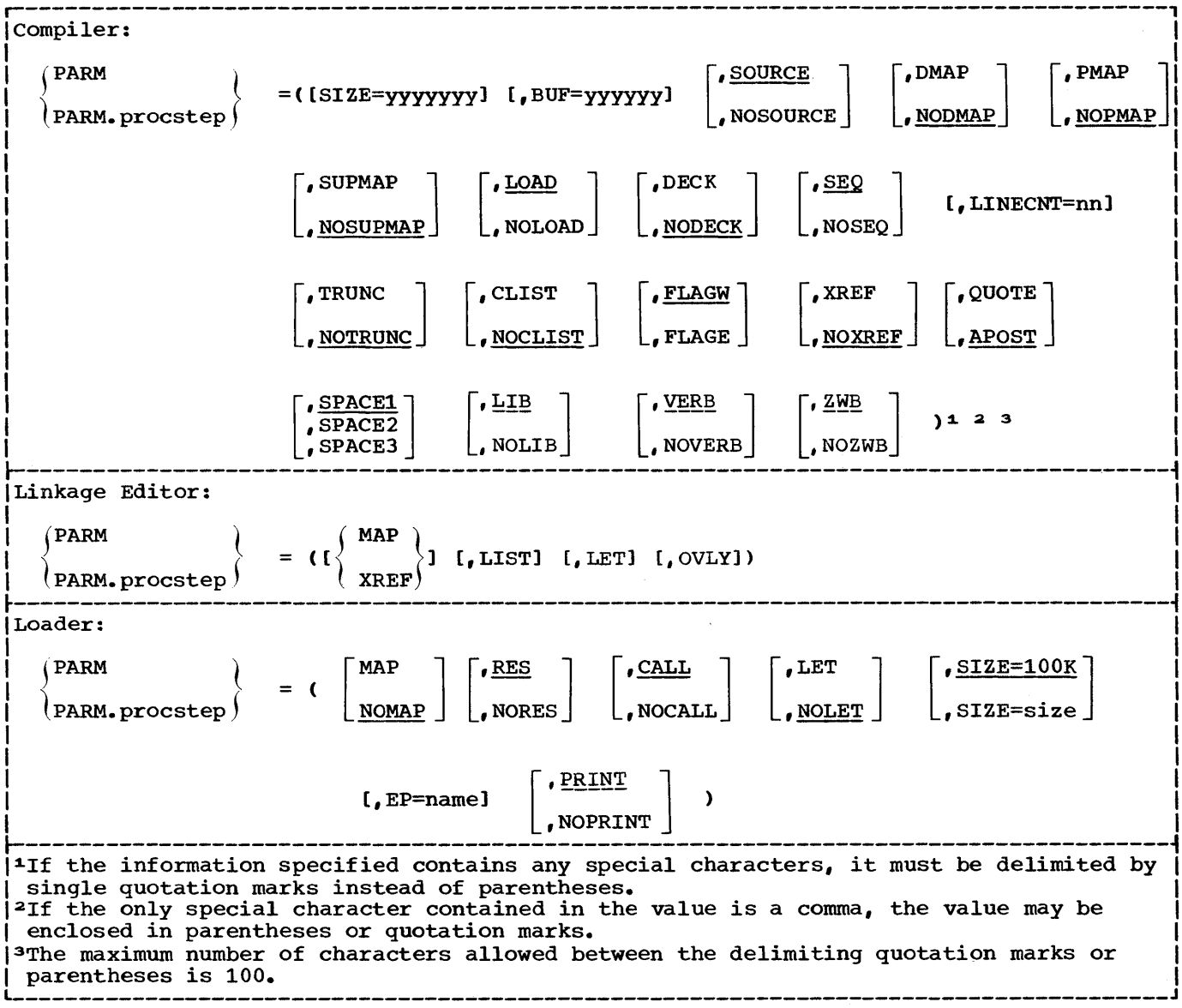


Figure 6. Compiler, Linkage Editor, and Loader PARM Options

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 6. The default options, indicated by an underscore, can be changed at system generation with the LOADER macro instruction.

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.
- When using a system with MVT or MFT, 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, refer to the publication IBM System/360 Operating System: Supervisor Services.

Priority Scheduling EXEC Parameters

Establishing a Dispatching Priority (DPRTY) (MVT 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.
- On an MVT system with time-slicing facilities, the DPRTY parameter can be used to make a job step part of a group of job steps to be time-sliced. The priorities of the time-sliced groups are selected at system generation. To cause the job step to be time-sliced, assign to "value 1" a number that corresponds to a priority number selected for time-slicing. "Value 2" is either omitted or assigned a value of 11.
- 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) (MFT or MVT only)

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)
(MVT only)

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 amount of main storage to be allocated to the job, and the storage hierarchy or hierarchies in which the space is to be allocated. This request should be made only if main storage hierarchy support has been specified during system generation. If an IBM 2361 Core Storage, Model 1 or 2, is present in the system, processor storage is referred to as hierarchy 0 and 2361 Core Storage is referred to as hierarchy 1. If 2361 Core Storage is not present but main storage hierarchy support was specified in system generation, a two-part region is established in processor storage when a region is defined to exist in two hierarchies. The two parts are not necessarily contiguous.

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

```
┌──────────────────────────────────────┐
│      REGION=(nnnnnxK[,nnnnnyK])      │
└──────────────────────────────────────┘
```

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

To request a region size and the hierarchy desired, the term nnnnnx is replaced with the number of contiguous 1024-byte areas to be allocated to the job in hierarchy 0; the term nnnnny is replaced with the number of contiguous 1024-byte areas to be allocated in hierarchy 1, e.g., REGION=(60K,200K). When only processor storage is used to include hierarchies 0 and 1, the combined values of nnnnnx and nnnnny cannot exceed 16383. If 2361 Core Storage is present, nnnnnx cannot exceed 16383 and, for a 2361 model 1, nnnnny cannot exceed 1024, or 2048 for a 2361 model 2. Each value specified should be an even number. (If an odd number is specified, the system treats it as the next higher even number.)

If storage is requested only in hierarchy 1, a comma must be coded to indicate the absence of the first subparameter, e.g., REGION=(,200K). If storage is requested only in hierarchy 0, or if hierarchy support is not present, the parentheses need not be coded, e.g., REGION=70K.

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.
- If main storage hierarchy support is not included but regions are requested in both hierarchies, the region sizes are combined and an attempt is made to allocate a single region from processor storage. If a region is requested entirely from hierarchy 1, an attempt is made to allocate the region from processor storage.
- For information on storage requirements to be considered when specifying a region size, see the publication IBM System/360 Operating System: Storage Estimates.

Specifying Additional Main Storage for a Job Step (ROLL)
(MVT only)

To allocate additional main storage to a job step whose own region does not contain any more available space, code the keyword parameter in the operand field of the EXEC statement.

```
┌---------------------------------------┐
|              ROLL=(x,y)               |
└---------------------------------------┘
```

In order to allocate this additional space to a job step, another job step may have to be rolled out, i.e., temporarily transferred to secondary storage. When x is replaced with YES, the job step can be rolled out; when x is replaced with NO, the job step cannot be rolled out. When y is replaced with YES, the job step can cause rollout; when y is replaced with NO, the job step cannot cause rollout. (If additional main storage is required for the job step, YES must be specified for y.) If this parameter is omitted, ROLL=(YES,NO) is assumed.

Notes:

- If the ROLL parameter is specified in the JOB statement, the ROLL parameter in the EXEC statements is ignored.
- When a job step uses a cataloged procedure, it can be indicated whether or not a single procedure step has the ability to be rolled out and to cause rollout of another job step. To indicate this, the procedure stepname, i.e., ROLL.procstepname, is included as part of the ROLL parameter. This specification overrides the ROLL 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 indicate whether or not all of the steps of a cataloged procedure have the ability to be rolled out and to cause rollout of other job steps, the ROLL parameter can be coded without a procedure stepname. This specification overrides all ROLL parameters in the procedure, if any are present.

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 7 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 IBM System/360 Operating System: Job Control Language Reference.

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

Operand[2]

Positional Parameters

[* / DATA / DUMMY] [3]

Keyword Parameters [4] [5]

[DDNAME=ddname]

[{ DSNAME / DSN } = { dsname / dsname(element) / *.ddname / *.stepname.ddname / *.stepname.procstep.ddname / &&name / &&name(element) }] [11]

[DCB=([dsname / *.ddname / *.stepname.ddname / *.stepname.procstep.ddname] [,subparameter-list])] [6]

[SEP=(subparameter list)[7] / AFF=ddname] [10]

Positional Subparameters Keyword Subparameters

[UNIT=(name[,[n/P][,DEFER]][,SEP=(list of up to 8 ddnames)])[8] / UNIT=(AFF=ddname)] [10] [12]

Positional Subparameters

SPACE=({ TRK / CYL / average-record-length } ,(primary-quantity[,secondary-quantity],
[directory- or index-quantity])[,RLSE] [,MXLG / ,ALX / ,CONTIG] [, ROUND])

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

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

SUBALLOC=({ TRK / CYL / average-reccrd-length } ,(primary-quantity[,secondary-quantity]
[,directory-quantity]), { ddname / stepname.ddname / stepname.procstep.ddname })

Figure 7. The DD Statement (Part 1 of 2)

Operand[2] (cont.)

Positional Subparameters

{VOLUME / VOL}=([PRIVATE],[RETAIN],[volume sequence number],[volume count])

Keyword Subparameters

[,SER=(volume-serial-number[volume-serial-number][9]...)]

[,REF={dsname / *.ddname / *.stepname.ddname / *.stepname.procstep.ddname}]

[LABEL=([data-set-sequence-number],{NL / SL / NSL / SUL}[,EXPDT=yyddd / ,RETPD=xxxx][,PASSWORD])]

[DISP=([NEW / OLD / SHR / MOD][,DELETE / ,KEEP / ,PASS / ,CATLG / ,UNCATLG][,DELETE / ,KEEP / ,CATLG / ,UNCATLG])]

[SYSOUT=classname
SYSOUT=(x[,program-name][,form-no.])]

[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]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.
[5]If subparameter-list is omitted, the entire parameter must be omitted.
[6]See "User-Defined Files" for the applicable subparameters.
[7]See the publication IBM System/360 Operating System: Job Control Language Reference.
[8]If only name is specified, the delimiting parentheses may be omitted.
[9]If only one volume-serial-number is specified, the delimiting parentheses may be omitted.
[10]The SEP and AFF parameters should not be confused with the SEP and AFF subparameters of the UNIT parameter.
[11]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.
[12]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.

Figure 7. The DD Statement (Part 2 of 2)

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.

Operand Field

* (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. If the EXEC statement specifies execution of a program, only one data set may be placed in the input stream. The end of the data set must be indicated by a delimiter statement. The data cannot contain // or /* in the first two characters of any record. The DD * statement must be the last DD statement of the job step. In MVT, for a step with a single input stream data set, DD * and a /* statement are not required. The system will supply both if missing. The default DDNAME will be SYSIN.

DATA (Defining Data in an Input Stream)
also indicates a source deck or data in the input stream. If the EXEC statement specifies execution of a program, only one data set may be placed in the input stream. The end of the data set must be indicated by a delimeter statement. The data cannot contain /* in the first two characters of any record. 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.

DUMMY (Bypassing Input/Output Operations on the Data Set)
allows the user's processing program to operate without performing input/output operations on the data set. The DUMMY parameter is valid only for sequential data sets to which reference is made by the basic sequential or queued sequential file processing techniques. If the DUMMY parameter is specified, a read request results in an end of data set exit. A write request is recognized, but no data is transmitted. No device allocation, external storage allocation, or cataloging takes place for dummy data sets.

Note: For a file defined in a COBOL source program, this operand may not be specified for a file opened OUTPUT. Any reference to the record area of a file opened OUTPUT with DD DUMMY will result in unpredictable results.

In multiprogramming environments, data in the input stream is temporarily transferred to a direct-access device for later high-speed retrieval. Normally, the reader procedure assigns a blocking factor for the data when it is placed on the direct-access device. The programmer may assign his own values through use of the BLKSIZE parameter 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)
```

If the programmer omits these parameters or assigns values greater than the capacity of the input reader, it is assumed that the established default values for the reader are in effect.

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 are ignored and

should be omitted when the DDNAME parameter appears (see "Using the Cataloged Procedures").

DDNAME Subparameter

ddname
: names 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.

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 IBM System/360 Operating System: Supervisor and Data Management Macro Instructions for detailed information about 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.

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. 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. 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 single cell within a data cell drive, 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 8. 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 notexceed the maximum block size permitted for the device. For example, the maximum block size for the IBM 2311 is 3,625 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 2311 and an IBM 2302 that may be used as utility devices, the user should write

BLOCK CONTAINS 3625 CHARACTERS

(or any number smaller than 3625) 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 indexed data set.

SEP=(list of up to 8 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 8. Device Class Names Required for IBM-Supplied Cataloged Procedures

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.

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 indexed data sets, only the CYL or ABSTR subparameter is permitted. When an indexed 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 indexed 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 indexed 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 indexed data set is being defined (see the publication IBM System/360 Operating System: Job Control Language 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 8 bytes, the location 4 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 indexed 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 indexed 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 IBM System/360 Operating System: Job Control Language 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 when one of the data sets is an indexed data set.

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 indexed data set.

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 the

UNIT 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 2301 Drum Storage.
- The volume from which the system is loaded (the IPL volume).
- The volume containing the system data sets SYS1.LINKLIB, SYS1.PROCLIB, and SYS1.SYSJOBQE.
- 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 PRESRES 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.

Table 3 shows how mass storage volumes are assigned their mount and allocation characteristics.

Table 3. Mass Storage Volume States

Mount Characteristic	Allocation Characteristic: Public	Private	Storage
Permanently Resident	PRESRES or Default	PRESRES	PRESRES
Reserved	PRESRES or MOUNT command	PRESRES or MOUNT command	PRESRES or MOUNT command
Removable	Default	VOLUME= PRIVATE	na

na = Not applicable

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. Table 4 shows the data set references.

When the data set resides in a tape volume and REF is specified, the data set is placed in the same volume, immediately behind the data set referred to by this subparameter. When the REF subparameter is used, the UNIT parameter, if supplied, is ignored.

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-written routines to write and process nonstandard labels have been incorporated into the operating system (see "User Label Processing" and the publication IBM System/360 Operating System: Systems Programmer's Guide for information on 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.)

Table 4. Data Set References

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")

{NL / SL / NSL / SUL}

specifies the kind of label used for the data set. NL indicates no labels. SL indicates standard labels. NSL indicates nonstandard label. SUL indicates 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.

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 the chapter, "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. Under the MVT or MFT option, for a 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.

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 or some 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. Both data sets and system messages can be routed through the same output stream when using a priority scheduler. In this case, the output class selected for the data sets must be the same output class as that selected for the MSGCLASS parameter in the JOB statement.

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 using priority schedulers.

(x[,program-name][,form-no])
is used for priority scheduling systems only. When priority schedulers are used, the data set is usually written on an intermediate mass storage device during program execution, and later routed through an output stream to a system output device. The x can be an alphabetic or numeric character specifying the system output class. 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.

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

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.

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 refer to the topic "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: When using a system with MFT or MVT, 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 certain jobs in an input stream. If the last DD statement in a job defines data in an input stream, the null statement should be used to mark the end of the job so that the card reader is 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.

DATA SET REQUIREMENTS

COMPILER

Nine data sets may be defined for a compilation job step; six of these (SYSUT1, SYSUT2, SYSUT3, SYSUT4, SYSIN, and SYSPRINT) are required. The other three data sets (SYSLIN, SYSPUNCH, and SYSLIB) 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	80
SYSPRINT	121

Note: When using the SYSUT1, SYSUT2, SYSUT3, SYSUT4, SYSPRINT, SYSPUNCH, or SYSLIN data sets, the following should be considered: If the primary space allocated for the data set is insufficient when compiling large programs, an area of core storage may be used to complete processing. This area would be used for an extra data extent block (DEB) and would be in the middle of the compiler's required core. Therefore, enough contiguous space may not be available to load a compiler phase. Such a condition will result in an abnormal termination of the job. The programmer should therefore attempt to allocate sufficient primary space to eliminate the need for a secondary allocation of space. The RLSE subparameter of the SPACE parameter should never be specified for any of these data sets.

The ddname that must be used in the DD statement describing the data set appears as the heading for each description that follows. Table 5 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

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. 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).

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 input/output device assigned to this data set can be either the device transmitting the input stream (the device designated as SYSIN at system generation time) or a device designated by the programmer. When using a cataloged procedure, the DD statement describing this data set usually appears in the input stream. For example,

```
//SYSIN    DD    *
```

specifies that the input data set follows in the input stream. If the asterisk or DATA convention is used, the SYSIN DD statement must be the last DD statement in the job step.

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.

Note: If SYSOUT=A is not specified for SYSPRINT, MOD must be specified in the DISP parameter.

SYSPUNCH

The data set defined by the SYSPUNCH DD statement is used to punch an object module 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: If SYSPUNCH is defined as a partitioned data set (PDS), then it must previously have been defined, and DISP=OLD must be specified.

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: If SYSLIN is defined as a partitioned data set (PDS), then it must previously have been defined, and DISP=OLD must be specified.

SYSLIB

The SYSLIB DD statement defines the library (PDS) that contains the data requested by a COPY statement (in the source module) or by a BASIS card in the input stream. 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). Libraries must always be on mass storage devices. Only one SYSLIB statement may be used in a compilation job step. For example, in the DD statement

```
//SYSLIB  DD  DSNAME=USERLIB,DISP=OLD
```

the name of the library is USERLIB, and DISP=OLD indicates that the library has been created in a previous job and is cataloged, or has been created in a previous step in this job. No other information need be given if the specified library has been cataloged.

Table 5. Data Sets Used for Compilation

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, standare output class A
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
SYSLIB (optional)	Library	Optional user source program library	Mass storage	SYSDA

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. Table 6 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

Table 6. Data Sets Used for Linkage Editing

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
User-specified (optional)	--	Additional object modules and load modules	Mass storage Magnetic tape	SYSDA, SYSSQ

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 magnetic tape. 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.).

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 DD statement 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).

Note: The SYSLIB statement can also define a sequential data set (see "Libraries").

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. Two are optional (SYSLIB, SYSLOUT). (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.

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 MVT. 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 record format of SYSLOUT must be FA, FBA, or 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. A DD statement that specifies an abnormal termination dump is also discussed. Use of the Sort and RERUN features require 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. If a DD statement for a DISPLAY statement is not specified, the DISPLAY statement becomes null, and message IKF999I, "UNSUCCESSFUL OPEN FOR SYSOUT" is issued. 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)
```

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

Example 2:

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

//SYSOUT DD applicable parameters

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

For example:

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

However, the programmer can specify an alternate line width 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
```

The DISPLAY statement can use a mnemonic-name rather than a system-name.

Example 3:

```
DISPLAY {identifier} ...UPON mnemonic-name
        {literal   }
```

where mnemonic-name is associated with the words 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. If a DD statement for an ACCEPT statement is not specified, the ACCEPT statement becomes null, and message IKF999I "UNSUCCESSFUL OPEN FOR SYSIN" is issued. 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.

Abnormal Termination Dump

To obtain a dump in case the job is abnormally terminated, 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 standard 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
```

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 system-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 system-name specified in the ASSIGN TO clause. In the system-name

```
UT-2400-S-TAXRATE
```

TAXRATE is the ddname portion of the system-name.

Note: The device-number specified in the system-name is used by the compiler for diagnostic purposes only, except for spanned records [that is, for RECORDING MODE S the block length for the file is checked against the maximum block length allowed for the particular device specified, and the smaller of the two lengths is used (placed in the DCB for the job)]. Actual device allocation is a function of the DD statement.

FILENAMES 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-2302-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 system-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 compiler to open its own file whose ddname is SYSOUT. If the programmer has also assigned one of his output files to SYSOUT, he must ensure that he has opened, written, and closed his file before the first execution of any of the aforementioned verbs. Additional considerations when using the Sort feature are detailed under "Additional DD Statements" in the chapter entitled "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 standard 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 it as a standard sequential data set.

FILE PROCESSING TECHNIQUES

DATA SET ORGANIZATION

A data set used by a COBOL program can have one of four types of organization: standard sequential, direct, relative, and indexed. The first type (sequential) may be on any input/output device. All other types must be on mass storage devices (see Figure 9 for information in determining the file processing technique to be used, according to data set organization).

1. A standard 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
		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
Write and read a mass storage file with relative record addressing	ACCESS SEQUENTIAL or omitted N=R		F	Mass Storage	BSAM
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 9. Determining the File Processing Technique

ACCESSING A STANDARD SEQUENTIAL FILE

A standard sequential file may only be accessed sequentially, i.e., records are read or written in the order they appear on the file. The file processing technique used to create and retrieve a standard sequential file is QSAM (Queued Sequential Access Method). Table 7 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.

Table 7. COBOL Clauses for Sequential File Processing

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]	
QSAM	MASS STORAGE	SEQUENTIAL	NOT ALLOWED	INPUT	READ [INTO] AT END	[UNIT] [LOCK]
				OUTPUT	WRITE [FROM] INVALID KEY WRITE [FROM] [{BEFORE / AFTER} ADVANCING] [AFTER POSITIONING]	
				I-O	READ [INTO] AT END WRITE [FROM] INVALID KEY REWRITE [FROM] INVALID KEY	[LOCK]

2. If the WRITE BEFORE/AFTER ADVANCING statement or the 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.

   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...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 standard 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.

Figures 10 and 11 show the parameters in the DD statement that may be used with standard 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 feature (see "Using the Sort Feature" for information on these parameters).

The DCB subparameters that can be specified in the DD statement for standard sequential files are as follows:

```
DCB=[DEN={0|1|2|3}]
    [,TRTCH={C|E|T|ET}]
    [,PRTSP={0|1|2|3}]
    [,MODE={C|E}]
    [,STACK={1|2}]
    [,OPTCD={W|C|WC|T}]
    [,BLKSIZE=integer]
    [,BUFNO=integer]
    [,EROPT={ACC|SKP|ABE}]
```

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 Table 8. 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.

Table 8. DEN Values

	Tape Recording Density (Bits per inch)--Model 2400	
DEN Value	7 Track	9 Track
0	200	--
1	556	--
2	800	800
3	--	1600

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|WC|T}
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 IBM System/360 Operating System: Data Management Services).
WC- To perform a validity check and use chained scheduling.
T - To request user totaling facility.

If this information is not supplied by any source, none of the services are provided, except in the case of the IBM 2321 mass storage device where OPTCD=W is specified by the operating system.

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. However, the maximum number allowed for an installation is established at system generation time.

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.

	Device Type		
Parameter	Mass Storage	Magnetic Tape	Unit Record
DSNAME	as		
UNIT	as		
VOLUME	as		na
LABEL	SL SUL	SL NL NSL SUL	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		EROPT=ACC (printer only) EROPT=ABE

as = Applicable subparameters
na = Not applicable

Figure 10. DD Statement Parameters Applicable to Standard 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	SL SUL	SL NL NSL SUL	na
SPACE	na		
SUBALLOC	na		
SPLIT	na		
DISP	{OLD / SHR} [,KEEP / ,PASS / ,CATLG / ,UNCATLG / ,DELETE]		
DCB Device Dependent	--	TRTCH, DEN	MODE, STACK
DCB General	OPTCD=C/T, BLKSIZE, BUFNO, EROPT=ACC, SKP, ABE		

as = Applicable subparameters
na = Not applicable

Figure 11. DD Statement Parameters Applicable to Standard Sequential INPUT and I-O Files

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:

a. If they appear on the same track, only the first occurrence can be retrieved (using BDAM).

b. 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 12 illustrates those parts of a directly organized file that are of importance to a COBOL programmer.

Each track contains the following:

Index Point
There is one index point to indicate the physical beginning of each track.

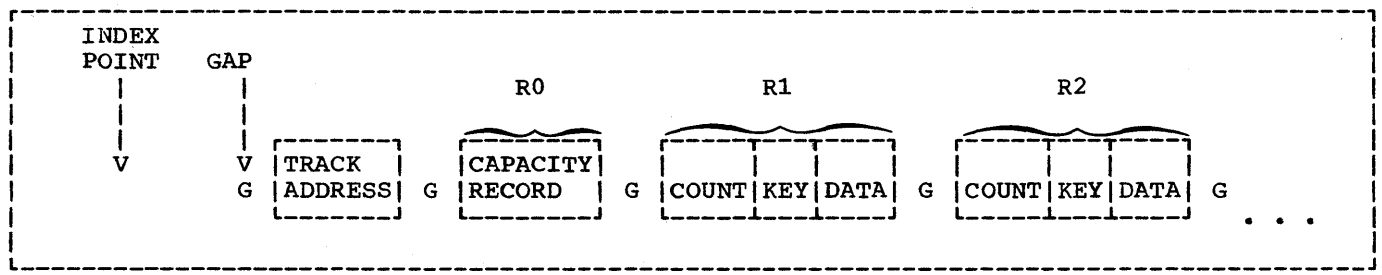


Figure 12. Directly Organized Data as it Appears on a Mass Storage Device

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. The length of the gap that follows a record depends on the length of that record.

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 -- the record identifier (1-255 bytes) as specified by the programmer in the ACTUAL KEY clause.

data -- 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).
```

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 13.

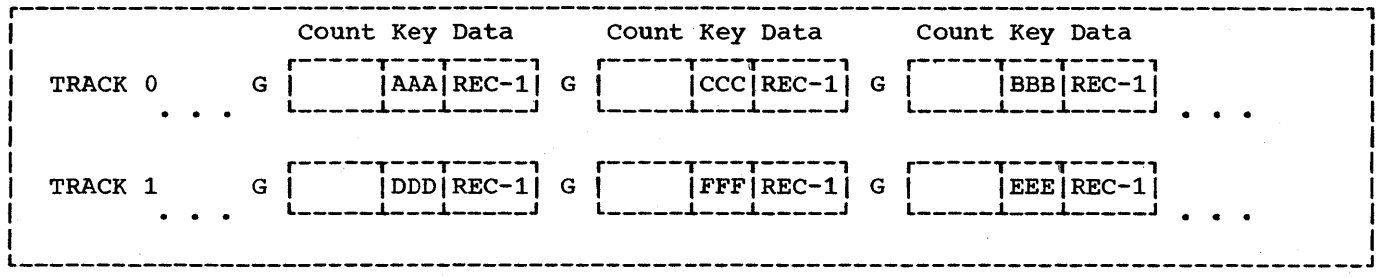


Figure 13. Sample Format of the First Two Tracks of a Direct File

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. 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 Table 13.
- The associated JCL parameters are summarized in Table 14.

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 14 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 = 500 and the file is closed after writing only 350-1/2 tracks:
   - 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:
   - 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 14, 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.

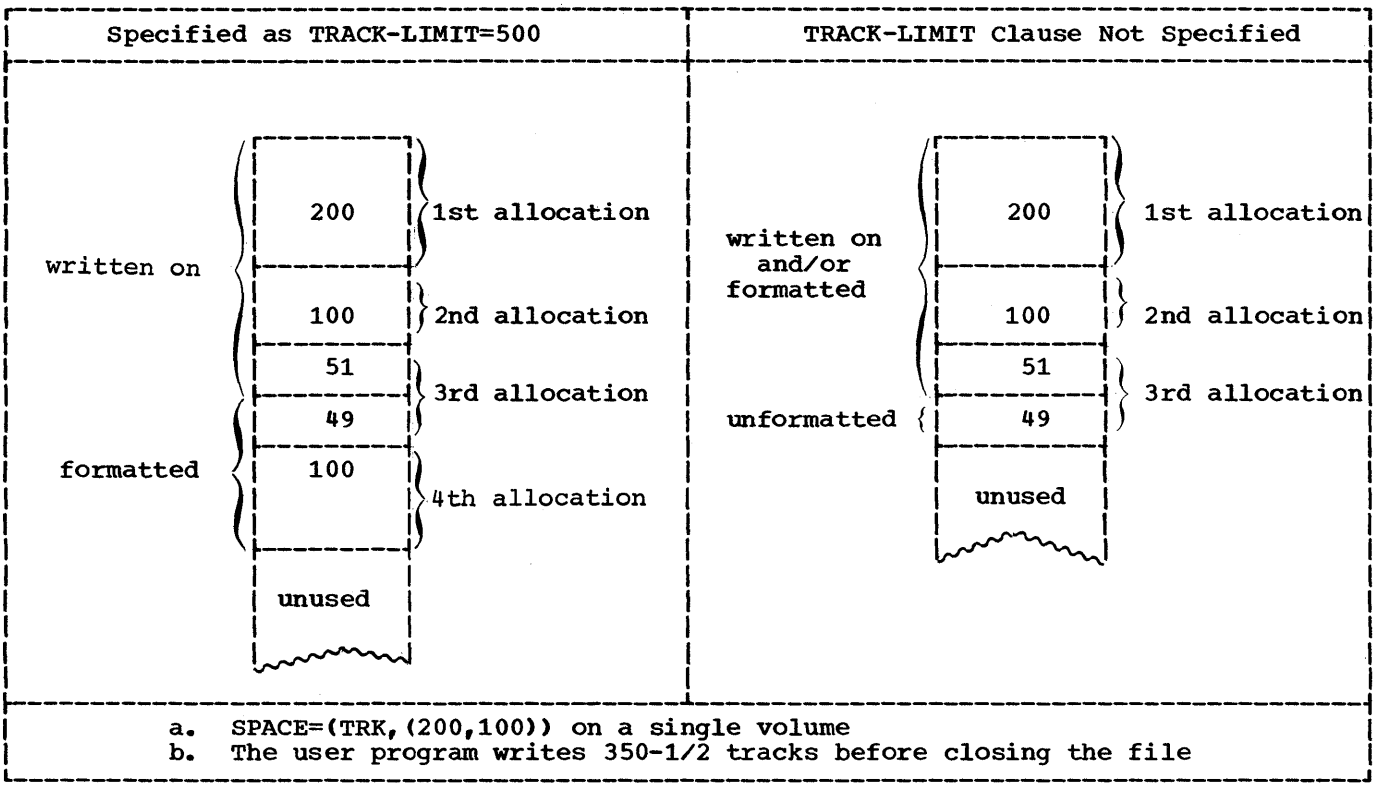


Figure 14. Sample Space Allocation for Sequentially Created Direct Files

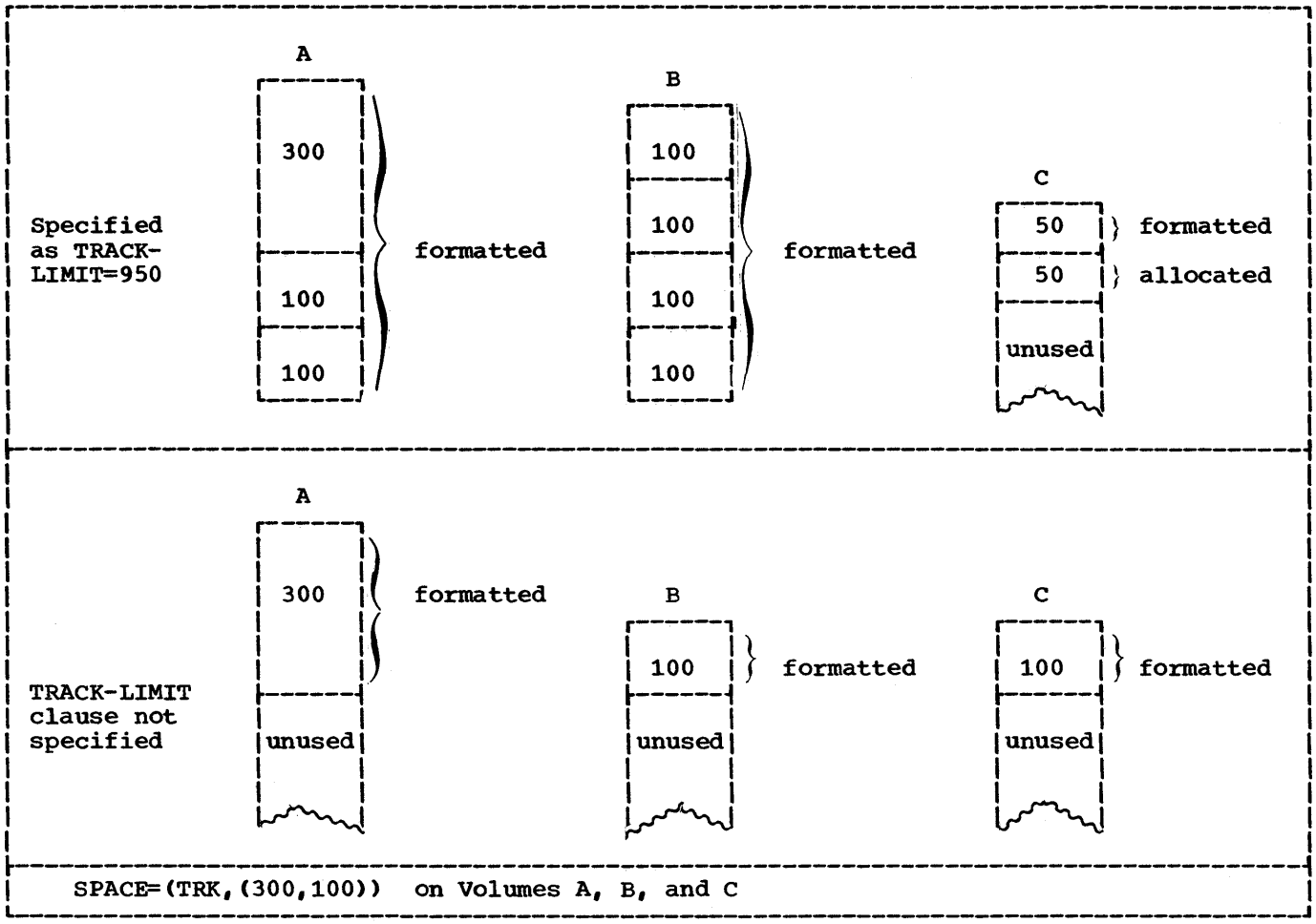


Figure 15. Sample Space Allocation for Randomly Created Direct Files

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 Table 13.
- The associated JCL parameters are summarized in Table 14.

Figure 16 (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 15 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 15), the system will do the following:

   a. Allocate tracks, by blocks, until the quantity specified by the TRACK-LIMIT clause has been equalled 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 15), 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 Table 13.
- The associated JCL parameters are summarized in Table 14.

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 Table 13.
- The associated JCL parameters are summarized in Table 14.

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.

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 16 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 80-byte records as evenly as posible on three 2311 volumes, 34 tracks per volume are required and the SPACE parameter should specify (34,34). After writing the 744th 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 15). It is important to note that a CLOSE UNIT statement is not permitted when creating a file randomly.

The following description pertains to Figure 15:

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. WRITE updates the file if the preceding input/output statement was a READ to the same record.

   b. WRITE adds a new record to the file, whether or not a duplicate key already exist, if the preceding READ was not to 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. 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 Table 9 may be used to compute the actual space required for each block, including device overhead.

Table 9. Mass Storage Device Overhead Formulas

Device Type	Bytes Required by Each Data Block			
	Blocks With Keys		Blocks Without Keys	
	Bi	Bn	Bi	Bn
2311	81+1.049(KL+DL)	20+KL+DL	61+1.049(DL)	DL
2314(2319)	146+1.043(KL+DL)	45+KL+DL	101+1.043(DL)	DL
2302	81+1.049(KL+DL)	20+KL+DL	61+1.049(DL)	DL
2303	146+KL+DL	38+KL+DL	108+DL	DL
2301	186+KL+DL	53+KL+DL	133+DL	DL
2321	100+1.049(KL+DL)	16+KL+DL	84+1.049(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	191+KL+DL	191+KL+DL	135+DL	135+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.

Table 10. Mass Storage Device Capacities

Device Type	Volume Type	Track Capacity	Tracks per Cylinder	Number of Cylinders	Total Capacity
2311	Disk	3625	10	200	7,250,000
2314(2319)	Disk	7294	20	200	29,176,000
2302	Disk	4984	46	246	56,398,944
2303	Drum	4892	10	80	3,913,600
2301	Drum	20483	8	25**	4,096,600
2321	Cell	2000	20***	980***	39,200,000
2305-1	Drum	14136	8	48	5,428,224
2305-2	Drum	14660	8	96	11,258,880
3330	Disk	13030	19	404	101,751,270

*Capacity indicated in bytes.
**There are 25 logical cylinders in a 2301 Drum.
***A volume is equal to one bin in a 2321 Data Cell.

Table 11. Mass Storage Device Track Capacity

Maximum Bytes per Record Formatted without Keys									Records per Track	Maximum Bytes per Record Formatted with Keys								
2311	2314 (2319)	2302	2303	2301	2321	2305-1	2305-2	3330		2311	2314 (2319)	2302	2303	2301	2321	2305-1	2305-2	3330
3625	7294	4984	4892	20483	2000	14136	14660	13030	1	3605	7249	4964	4854	20430	1984	13934	14569	12974
1740	3520	2403	2392	10175	935	6852	7231	6447	2	1720	3476	2383	2354	10122	920	6650	7140	6391
1131	2298	1570	1558	6739	592	4424	4754	4253	3	1111	2254	1505	1520	6686	576	4222	4663	4197
830	1693	1158	1142	5021	422	3210	3516	3156	4	0811	1649	1139	1104	4968	406	3008	3425	3100
651	1332	912	892	3990	320	2480	2773	2498	5	632	1288	893	854	3937	305	2278	2682	2442
532	1092	749	725	3303	253	1996	2278	2059	6	512	1049	730	687	3250	238	1794	2187	2003
447	921	634	606	2812	205	1648	1924	1745	7	428	877	614	568	2759	190	1446	1833	1689
384	793	546	517	2444	169	1388	1659	1510	8	364	750	527	479	2391	154	1186	1568	1454
334	694	479	447	2157	142	1186	1452	1327	9	315	650	460	409	2104	126	984	1361	1271
295	615	425	392	1928	119	1024	1287	1181	10	275	571	406	354	1875	103	822	1196	1125
263	550	381	346	1741	101	892	1152	1061	11	244	506	362	308	1688	85	690	1061	1005
236	496	344	308	1585	86	782	1040	962	12	217	452	325	270	1532	70	580	949	906
213	450	313	276	1452	73	688	944	877	13	194	407	294	238	1399	58	486	853	821
193	411	286	249	1339	62	608	863	805	14	174	368	267	211	1286	47	406	772	749
177	377	164	225	1241	53	538	792	742	15	158	333	245	187	1188	38	336	701	686
162	347	244	204	1155	44	478	730	687	16	143	304	224	166	1102	29	276	639	631
149	321	225	186	1079	37	424	676	639	17	130	277	206	148	1026	21	222	585	583
138	298	209	169	1012	30	376	627	596	18	119	254	190	131	959	15	174	536	540
127	276	196	155	952	24	334	584	557	19	108	233	176	117	899	9	132	493	501
118	258	183	142	897	20	296	544	523	20	99	215	163	104	844		94	453	467
109	241	171	130	848	15	260	509	491	21	90	198	152	92	795		58	418	435
102	226	161	119	804	10	230	477	463	22	82	183	142	81	751			386	407
95	211	151	109	763	6	200	448	437	23	76	168	132	71	710			357	381
88	199	143	100	726		174	421	413	24	69	156	123	62	673			330	357
82	187	135	92	691		150	396	391	25	63	144	116	54	638			305	335
77	176	127	84	659		128	373	371	26	58	133	108	46	606			282	315
72	166	101	77	630		106	352	352	27	53	123	102	39	577			261	296
67	157	114	70	603		88	332	335	28	48	114	95	32	550			241	279
63	148	108	64	577		70	314	318	29	44	105	89	26	524			223	262
59	139	102	58	554		52	297	303	30	40	96	83	20	501			206	247

Table 10 lists device storage capacity and Table 11 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 publication IBM System/360 Component-Description--2841 Storage Control, 2302 Disk Storage, Models 3 and 4, 2311 Disk Storage Drive, Model 7 2321 Data Call Drive 2303 Drum, Order No. A26-5988.

Component Summary -- 2835 Storage Control, 2305 Fixed Head Storage, Order No. GA26-1589.

Component Summary -- 3830 Storage Control, 3330 Disk Storage, Order No. GA26-1592.

Note: The programmer may use any of the following devices with this compiler by omitting the device-number in the system-name: the 2305 (Models 1 and 2), the 3330, and the 2319.

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. Table 12 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 2311 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 with 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., 12 per

track on a 2311, 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 Tables 9, 10, and 11).

2. Divide the relative number (2445) by the number of records to be stored on each track.

3. The result, quotient = 203, 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 16:

1. The number of records that may be contained on one track is 12. Therefore, if 10,000 record locations are to be provided, 834 tracks must be reserved.

2. The prime number nearest, but less than 834, is 829.

3. Divide the zone-stripped key by the prime value. (In the example, 25463514 divided by 829 provides a quotient of 30715 and a remainder of 779. The remainder is the track identifier.)

Table 12. Partial List of Prime Numbers (Part 1 of 2)

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

Table 12. Partial List of Prime Numbers (Part 2 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	8899
9000	8899
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 16 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 829 as its divisor, the largest possible remainder is 828. By the interaction between the TRACK-LIMIT clause (2) and the SPACE parameter (3) , the program formats 830 tracks (i.e., relative tracks 000-829). This establishes track 829 as the only track that can contain synonym overflow from track 828.

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.

```
00001    00101  IDENTIFICATION DIVISION.
00002    00102  PROGRAM-ID.  METHOD B.
00003    00103  ENVIRONMENT DIVISION.
00004    00104  CONFIGURATION SECTION.
00005    00105  SOURCE-COMPUTER. IBM-360.
00006    00106  OBJECT-COMPUTER.  IBM-360.
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 830.  <-------------------(2)
00012 **00103      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 829 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 16. Sample Program for a Randomly Created Direct File (Part 1 of 2)

```
STEP STEP2    TERMINATED. TIME 00.00 HR.HDRTH/HR * 00.00.23.30 HR.MIN.SEC.HDRTH/SEC*DATE 70.139
//STEP3  EXEC  PGM=*.STEP2.SYSLMOD
//SYSOUT  DD  SYSOUT=G
//SYSUDUMP DD  SYSOUT=A
//MASTER  DD  SPACE=(TRK,(500,100),RLSE),                              X
//             DCB=(OPTCD=E,LIMCT=5),UNIT=2314
//CARDS  DD *
//
```

(3) (4) (5)

```
TRACK-ID = 00801 C-REC = 82900801CD1
TRACK-ID = 00801 C-REC = 82900801CD2
TRACK-ID = 00801 C-REC = 82900801CD3
TRACK-ID = 00801 C-REC = 82900801CD4
TRACK-ID = 00031 C-REC = 82900031
TRACK-ID = 00801 C-REC = 82900801CD5
TRACK-ID = 00801 C-REC = 82900801CD6
TRACK-ID = 00801 C-REC = 82900801CD7
TRACK-ID = 00801 C-REC = 82900801CD8
TRACK-ID = 00801 C-REC = 82900801CD9
TRACK-ID = 00801 C-REC = 82900801CD10
TRACK-ID = 00801 C-REC = 82900801CD11
TRACK-ID = 00801 C-REC = 82900801CD12
TRACK-ID = 00801 C-REC = 82900801CD13
TRACK-ID = 00801 C-REC = 82900801CD14
TRACK-ID = 00801 C-REC = 82900801CD15
TRACK-ID = 00801 C-REC = 82900801CD16
TRACK-ID = 00000 C-REC = 829000003
TRACK-ID = 00801 C-REC = 82900801CD17
TRACK-ID = 00801 C-REC = 82900801CD18
TRACK-ID = 00801 C-REC = 82900801CD19
TRACK-ID = 00801 C-REC = 82900801CD20
TRACK-ID = 00809 C-REC = 82900809
TRACK-ID = 00801 C-REC = 82900801CD21
TRACK-ID = 00801 C-REC = 82900801CD22
TRACK-ID = 00801 C-REC = 82900801CD223
TRACK-ID = 00801 C-REC = 82900801CD24
TRACK-ID = 00801 C-REC = 82900801CD25
TRACK-ID = 00801 C-REC = 82900801CD26
```

Figure 16. Sample Program for a Randomly Created Direct File (Part 2 of 2)

Table 13. Direct File Processing on Mass Storage Devices

File Organization	Data Management Techniques	Access Method	KEY Clauses	OPEN Statement	Access Verbs	CLOSE Statement
	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]

Table 14. JCL Applicable to Directly Organized Files

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 as na RLSE	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	{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 as na RLSE	NEW [,KEEP / ,CATLG / ,PASS / ,DELETE] Note: MOD not meaningful	[DSORG=DA] OPTCD=[W,E] LIMCT=n

as = Applicable subparameters
na = Not applicable

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 17 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 50 times. With each execution of the WRITE statement, the content of THE-NOMINAL-KEY is incremented by 1, from 0 through 49. Since a 2311 mass storage device has room for only twenty-five 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 24 will be on the first track.
- Relative records 25 through 49 will be on the second track.

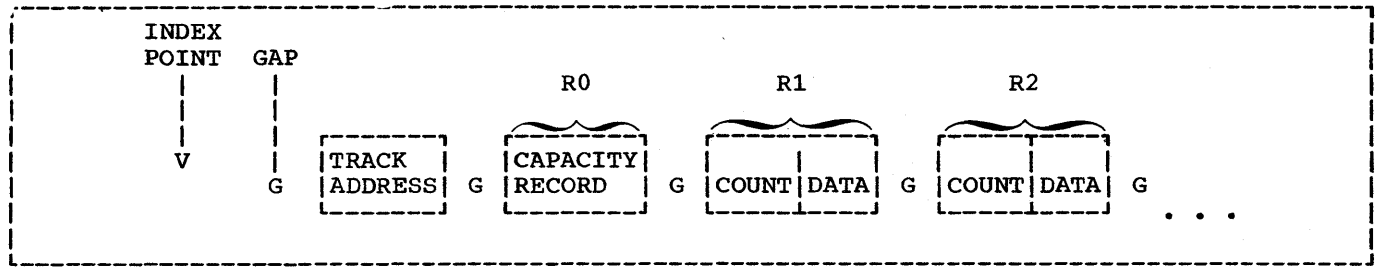


Figure 17. Relatively Organized Data as it Appears on a Mass Storage Device

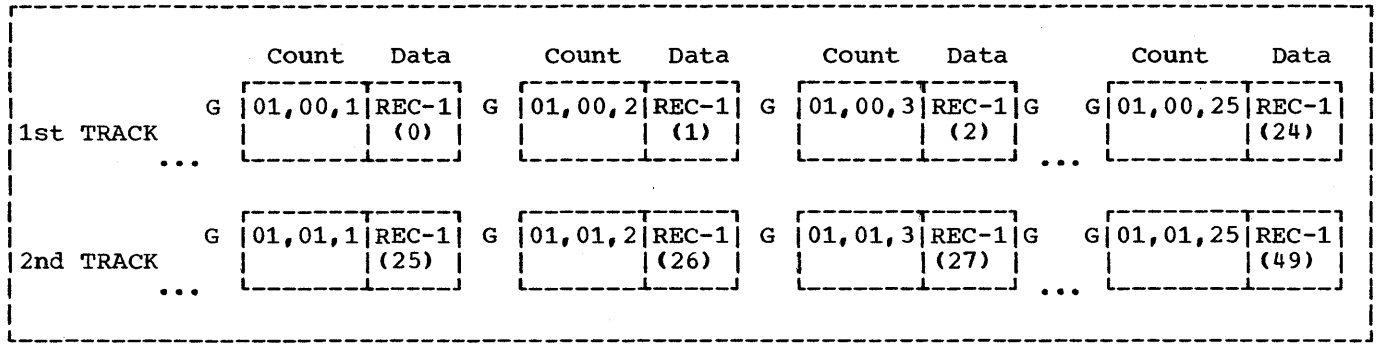


Figure 18. Sample Format of Two Tracks of a Relative File

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 18.

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 18, 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 Table 15.
- The associated JCL statements are summarized in Table 16.

Figure 19 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 1,000 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 Tables 9, 10, and 11.

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 Table 15.

- The associated JCL parameters are summarized in Table 16.

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 Table 15.

- The associated JCL statements are summarized in Table 16.

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 19 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 2311 mass-storage device, the track capacity for R-FILE is 25 record per track. Relative record 100 is, therefore, the first record written on track 4 (remember: the first 5 tracks of a file are actually relative tracks 0 through 4). Since the file is closed after writing relative record 100, the COBOL subroutine formats the rest of track 4. In this case, it means the addition of 24 dummy records, ⑤

```
00001   00101  IDENTIFICATION DIVISION.
00002   00102  PROGRAM-ID. CREATER.
00003   00103  REMARKS. ILLUSTRATE CREATION OF A RELATIVE FILE.
00004   00104  ENVIRONMENT DIVISION.
00005   00105  CONFIGURATION SECTION.
00006   00106  SOURCE-COMPUTER. IBM-360.
00007   00107  OBJECT-COMPUTER. IBM-360.
00008   00108  INPUT-OUTPUT SECTION.
00009   00109  FILE-CONTROL.
00010   00110      SELECT R-FILE ASSIGN DA-2311-R-MASTER
00011   00111          ACCESS IS SEQUENTIAL
00012   00112          NOMINAL KEY IS N-KEY.
00013   001125     SELECT C-FILE ASSIGN UR-S-CARDS.
00014   001126     SELECT R-FILE2 ASSIGN DA-2311-R-MASTER.
00015   001127     SELECT PRTFILE ASSIGN UR-S-PRTOUT.
00016   00113  DATA DIVISION.
00017   00114  FILE SECTION.
00018   00115  FD  R-FILE
00019   00116      LABEL RECORDS ARE STANDARD
00020   00117      RECORDING MODE IS F
00021   00118      DATA RECORD IS DISK.
00022   001184 01  DISK PIC X(80).
00023   001185 FD  R-FILE2 LABEL RECORDS ARE STANDARD.
00024   001186 01  DISK2 PIC X(80).
00025   00201  FD  C-FILE
00026   00202      LABEL RECORDS ARE OMITTED
00027   00203      DATA RECORD IS CARD.
00028   00204  01  CARD.
00029   002041     02  C-KEY PIC 9(3).
00030   002042     02  FILLER PIC X(77).
00031   002043 FD  PRTFILE  LABEL RECORDS ARE OMITTED.
00032   002044 01  PRT.
00033   002045     02  FILLER PIC X.
00034   002046     02  FIELD1  PIC X(132).
00035   00205  WORKING-STORAGE SECTION.
00036   00206  77  N-KEY PIC S9(8) COMP SYNC.
00037   00207  PROCEDURE DIVISION.
00038   00208      OPEN INPUT C-FILE
00039   00209           OUTPUT R-FILE.
00040   00210  R1. READ C-FILE AT END GO TO EOJ1.
00041   00211      MOVE C-KEY TO N-KEY.
00042   00212      WRITE DISK FROM CARD.
00043   00213      EXHIBIT NAMED N-KEY.  GO TO R1.
00044   00214  EOJ1.
00045   00215      CLOSE C-FILE R-FILE.
00046   00216      OPEN INPUT R-FILE2  OUTPUT PRTFILE.
00047   00217  R2. READ R-FILE2 AT END GO TO EOJ2.
00048   00218      MOVE DISK2 TO FIELD1.
00049   00219      WRITE PRT AFTER 1 LINES  GO TO R2.
00050   00220  EOJ2.
00051   00230      CLOSE R-FILE2 PRTFILE.  STOP RUN.
```

Figure 19. Sample Program for Relative File Processing (Part 1 of 4)

```
IEF285I   PROCPAST                                        PASSED
IEF285I   VOL SER NOS= LSASIA.
IEF285I   SYS69184.TC3C423.RVCCC.RFILE.CTL                DELETED
IEF285I   VOL SER NOS= MVTRES.
IEF285I   SYS69184.TC3C423.RVCOC.RFILE.AJCB               PASSED
IEF285I   VOL SER NOS= 222222.
IEF285I   SYS1.CCBLIB                                     KEPT
IEF285I   VOL SER NOS= LSASIA.
IEF285I   SYS69184.TC3C423.SVCCC.RFILE.RCOCC032           SYSOUT
IEF285I   VOL SER NOS= 231400.
IEF285I   SYS69184.TC3C423.RVCOC.RFILE.RACH               DELETED
IEF285I   VOL SER NOS= 222222.
STEP STEP2     TERMINATED. TIME CO.CC HR.MCRTH/HR * CC.CO.18.08 HR.MIN.SEC.MCRTH/SEC*DATE 69.184
//STEP3  EXEC  PGM=*.STEP2.SYSLMOD
//SYSCUT DD    SYSOUT=A
//SYSLOMP  DD  SYSOUT=A
//MASTER   DD  UNIT=2311,VOLUME=SER=CAC28,SPACE=(TRK,(5,5),,CONTIG),    X
//             DSNAME=RFILE,DISP=(NEW,KEEP)
//PRTOUT   DD  SYSOUT=A
//CARDS  DD  *
//
IEF236I ALLOC. FOR RFILE    STEP3
IEF237I JOBLIB   ON 193
IEF237I PGM=*.DD ON 19C
IEF237I SYSOUT   ON 23C
IEF237I SYSLOMP  ON 225
IEF237I MASTER   ON 192
IEF237I PRTOUT   ON 23C
IEF237I CARDS    ON 225

N-KEY = CCCCCC1C
N-KEY = CCCCCC2C
N-KEY = COOCOC3C
N-KEY = CCCCCC4C
N-KEY = CCCCCC5C
N-KEY = CCOCCC6C                                          (1)
N-KEY = CCCCCC61
N-KEY = CCCCCC7C
N-KEY = CCCCOC8C
N-KEY = CCCCCC9C
N-KEY = COCCO1CC


 1C  NAMEC1
 1C  NAMEC1
 1C  NAMEC1
 1C  NAMEC1
 1C  NAMEC1
 1C  NAMEC1                                               (2)
 1C  NAMEC1
 1C  NAMEC1
 1C  NAMEO1
 1C  NAMEC1
010  NAMEO1
 2C   NAMEC2
 2C   NAMEC2
 2C   NAMEC2
 2C   NAMEC2
 2C   NAMEC2                                              (3)
 2C   NAMEC2
 2C   NAMEC2
 2C   NAMEC2
 2C   NAMEC2
```

Figure 19. Sample Program for Relative File Processing (Part 2 of 4)

```
020   NAME02
 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
 4C    NAME04
 40    NAME04
 40    NAME04
 40    NAME04
 40    NAME04
 4C    NAME04
040    NAME04
 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
 6C    NAME06
060    NAME06
000   THIS  CARD  IS  OUT  OF  SEQUENCE   (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
 8C   NAME08
080   NAME08
 90    NAME09
 90    NAME09
 90    NAME09
 9C    NAME09
 90    NAME09
```

Figure 19. Sample Program for Relative File Processing (Part 3 of 4)

```
 90    NAME09
 90    NAME09
 90    NAME09
 90    NAME09
090    NAME09
 00  NAME10
 00  NAME10
 00  NAME10
 00  NAME10
 00  NAME10
 00  NAME10
 00  NAME10
 00  NAME10
 00  NAME10
100  NAME10
 00  NAME10
 00  NAME10
 00  NAME10
 00  NAME10
 00  NAME10
 00  NAME10
 00  NAME10
 00  NAME10
 00  NAME10
 00  NAME10
 00  NAME10
 00  NAME10
 00  NAME10
 00  NAME10
 00  NAME10
 00  NAME10
 00  NAME10
 00  NAME10
 00  NAME10
 00  NAME10
```

(5)

Figure 19. Sample Program for Relative File Processing (Part 4 of 4)

Table 15. Relative File Processing on Mass Storage Devices

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	

Table 16. JCL Applicable to Relatively Organized Files.

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	OPTCD={W,T} [DSORG=DA]

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		{OLD / SHR} {,PASS / ,KEEP / ,CATLG / ,UNCATLG / ,DELETE}	as has been specified

as = Applicable subparameters
na = Not applicable

INDEXED FILE PROCESSING

The indexed 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).

With these exceptions, the item specified by data-name may appear anywhere within the record.

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.

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 20).

Figure 20 is the representation of a track index with the following areas:

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 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 20 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

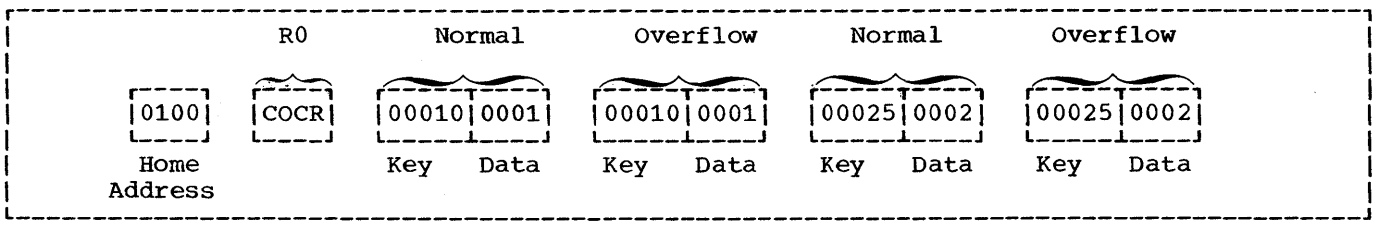


Figure 20. Track Index

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.

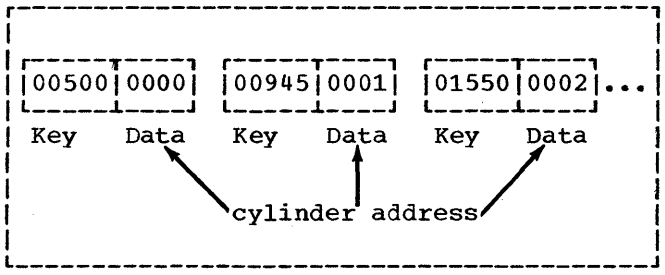


Figure 21. Cylinder Index

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 21).

The cylinder index is formatted in the same fashion as the track index. Figure 21 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 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 files, may be either blocked or unblocked; but must be F-mode records. Figures 22 and 23 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.

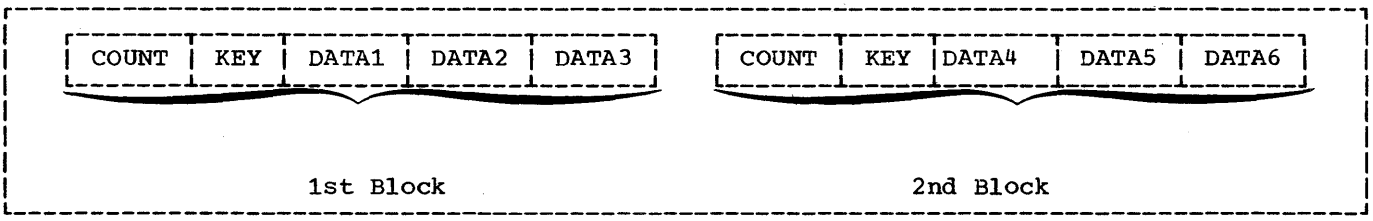


Figure 22. Blocked Records on an Indexed File

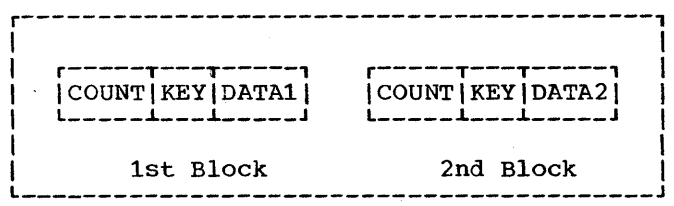


Figure 23. Unblocked Records on an Indexed File

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 File Areas

The programmer specifies the structure of an indexed 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 24). Tracks on each cylinder can be reserved to hold the overflow of that cylinder (cylinder overflow option).

2. Independent Overflow (Figure 25). 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).

Additional information about indexed file structure is contained in the publication IBM System/360 Operating System: Data Management Services.

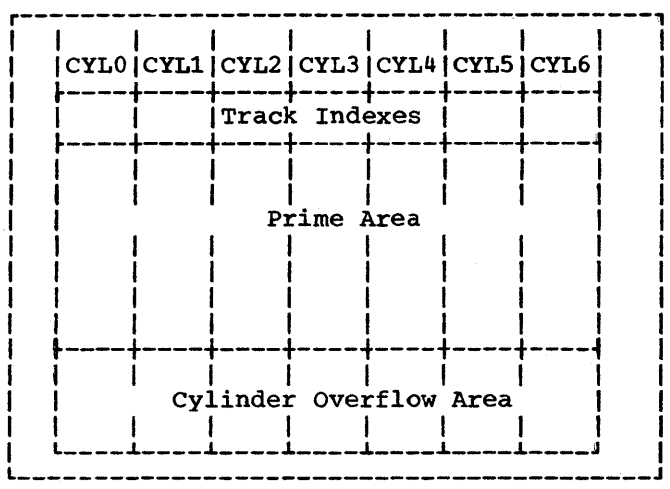


Figure 24. Cylinder Overflow Area

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.

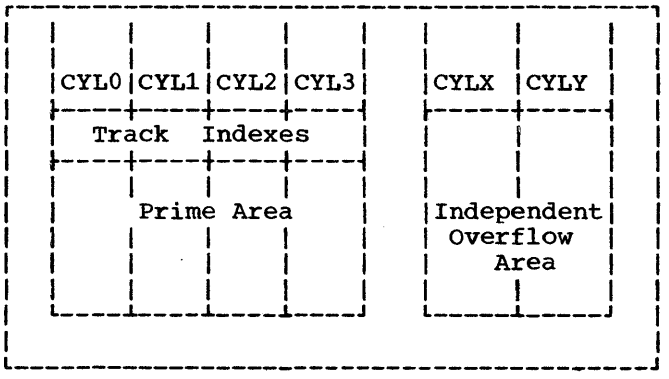


Figure 25. Independent Overflow Area

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 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 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 files are discussed below. The discussion is oriented to indexed 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 the publication IBM System/360 Operating System: Job Control Language Reference or IBM System/360 Operating System: Job Control Language Programmers Guide.

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 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 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 file, either for sharing a cylinder with indexed files or for sharing it with an indexed 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 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 following default services are provided: (1) the COBOL compiler will supply OPTCD=L; and (2) in the case of an IBM 2321 mass storage device, the operating system will supply OPTCD=W.

- 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 core storage until there are enough entries to fill a track. When the track is full all the entries will be written out. If enough core 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 26 shows the parameters that may be used in a DD statement when processing indexed files opened as output. Additional information about indexed file structure is contained in the publication IBM System/360 Operating System: Data Management Services.

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 IBM System/360 Operating System: Job Control Language Reference.

All three areas for an indexed 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 / ABSTR] ,... [,,CONTIG]
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 26. 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 27 defines a new indexed file that consists of three separate areas. All three areas reside on the same volume. The volume is on an IBM 2311 Disk Storage Drive.

```
//FILE    DD  DSNAME=ISM(INDEX),UNIT=2311,SPACE=(CYL,(1)),   X
//            VOLUME=SER=111111,DCB=(DSORG=IS,OPTCD=I,...),  X
//            DISP=(,KEEP)
//        DD  DSNAME=ISM(PRIME),UNIT=2311,SPACE=(CYL,(5)),   X
//            VOLUME=SER=111111,DISP=(,KEEP),DCB=*.FILE
//        DD  DSNAME=ISM(OVFLOW),UNIT=2311,SPACE=(CYL,(1)),  X
//            VOLUME=SER=111111,DISP=(,KEEP),DCB=*.FILE
```

Figure 27. Example of DD Statements for New Indexed Files

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 IBM System/360 Operating System: Utilities).

An indexed file that is being created cannot be cataloged if its areas are on different device types. An existing indexed file cannot be cataloged through the specification of the CATLG subparameter of the DISP parameter in the DD statement.

Note: The DD statement(s) defining a new or existing indexed file can appear in cataloged procedures.

Calculating Space Requirements: To determine the number of cylinders required for an indexed 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 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 IBM System/360 Operating System: Data Management Services.

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 Files" in "Creating Indexed 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 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 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 Table 17.

Reading or Updating Indexed Files Sequentially

QISAM can be used to read or update an existing indexed 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 files that can be contained on a single volume. Additional information about processing existing indexed files accessed sequentially, including multivolume files, can be found in the publication IBM System/360 Operating System: Job Control Language 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 subparameter 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 files).

Note: For further information about Indexed parameters, see "DD Statement Requirements for Indexed Files" in "Creating Indexed 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 files can be found in the publication IBM System/360 Operating System: Job Control Language Reference.

Note: Figure 28 shows the parameters that may be used in a DD statement when processing indexed files opened as input or I-O. Additional information about indexed file structure is contained in the publication IBM System/360 Operating System: Data Management Services.

Reorganizing Files: As new records are added to an indexed 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 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 IBM System/360 Operating System: Data Management Services.

Sequential Retrieval Using the START Statement: For indexed 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.

Note: If SETL is to be issued from a user-written assembler language program against a QISAM file opened by a COBOL program, either a null START statement which has never been branched to should

appear in the COBOL program, or an assembler language program should be called before the file is opened. This program must set the MACRF field of the DCB to ensure loading of the SETL and ESETL routines.

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)
[1]CATLG UNCATLG not permitted	

Figure 28. DD Statement Parameters Applicable Indexed 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 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. 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 File Randomly."

Accessing an Indexed 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 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 System/360 Operating System: Full American National Standard COBOL 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.

Note:

- 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.

Table 17. Indexed File Processing on Mass Storage Devices

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] INVALID KEY	
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	

- 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 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 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 Table 17.

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 DATA SET

When creating a 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 29 shows the subparameters that are frequently used in creating data sets. Additional subparameters are discussed in "Job Control Procedures."

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.

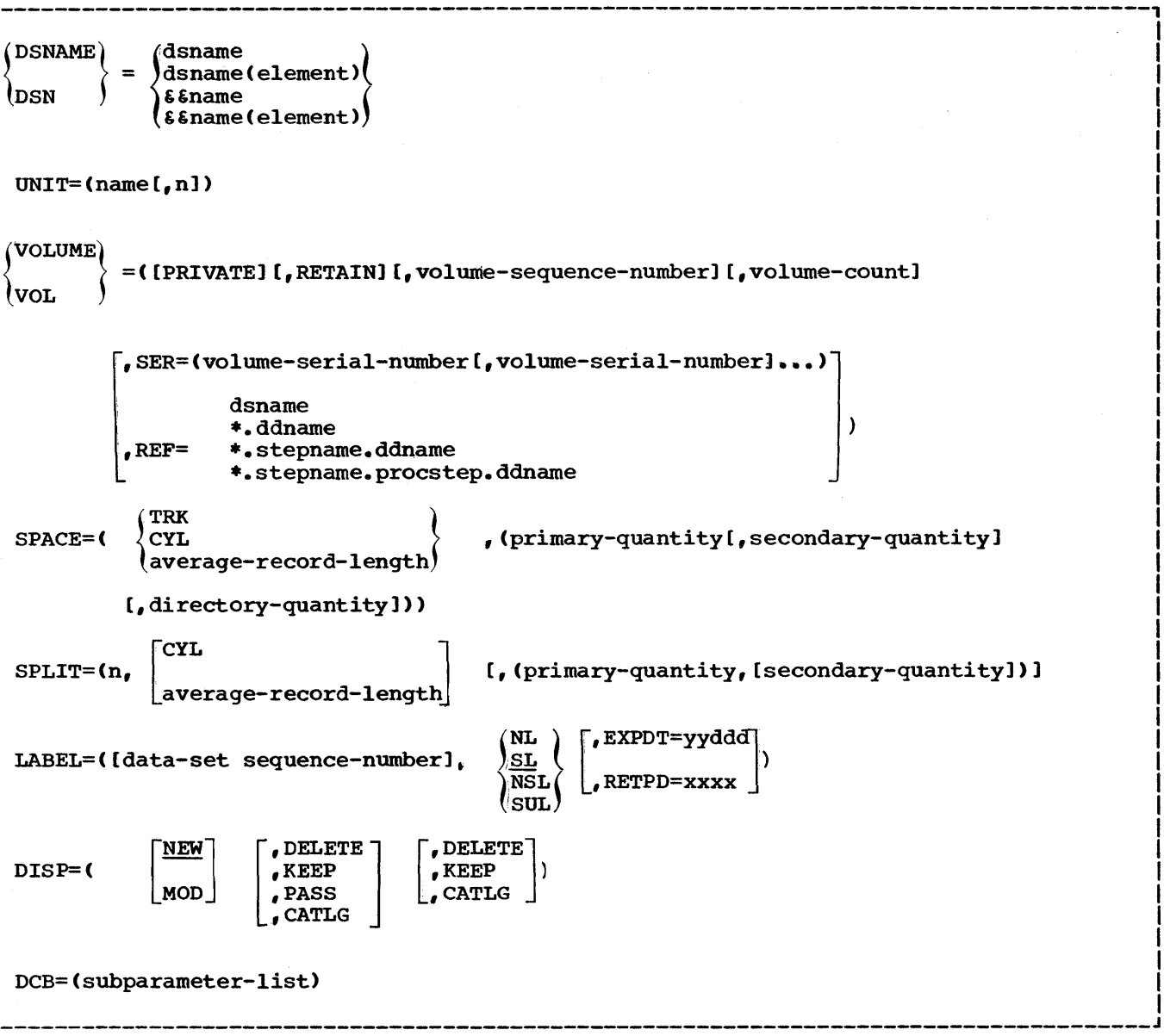


Figure 29. DD Statement Parameters Frequently Used in Creating Data Sets

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 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. When using a sequential scheduler, 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. (Do not use classes 0 through 9 except in cases where the other classes are not sufficient.)

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, the priority 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 IBM System/360 Operating System: Operator's Reference, Order No. GC28-6691.

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=69031),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 1969.

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. The member INVENT takes on the disposition of the library.

Example 6: Creating a library and its first member.

```
//SYSLMOD   DD DSNAME=USERLIB(MYPROG),DISP=(,CATLG),                   X
//              SPACE=(TRK,(50,30,3)),UNIT=2311,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 2311 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 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 30 shows the parameters that are used to retrieve 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.

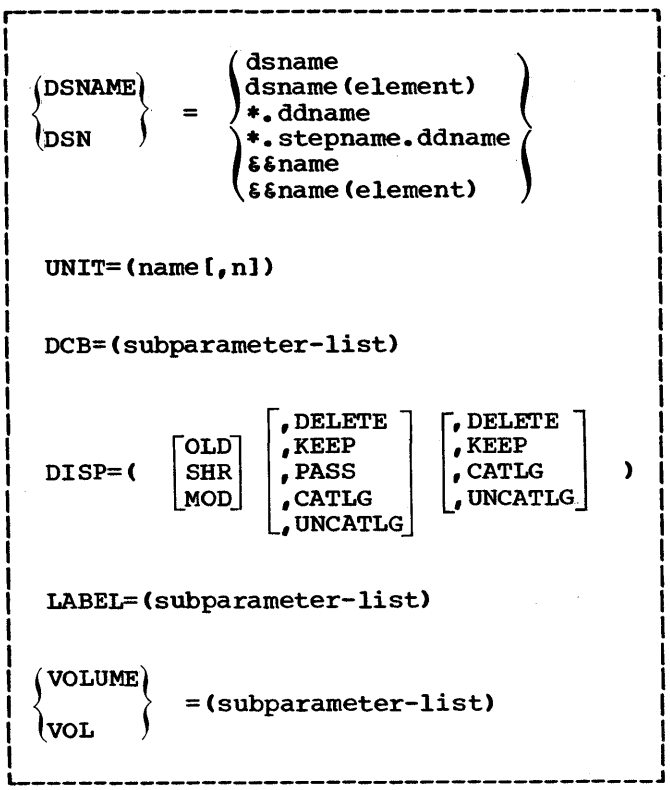


Figure 30. Parameters Frequently Used in Retrieving Previously Created Data Sets

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 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.

Extending Data Sets With Additional Output

A processing program can extend an existing data set by adding records to it instead of reading it as input. Such a data set is retrieved using the same subsets of DD statement parameters described under the preceding three topics, depending on whether it was cataloged, kept, or passed when created. In each case, however, the DISP parameter must indicate a status of MOD. 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.

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. In systems with MFT or MVT, data in the input stream is temporarily transferred to a mass storage device. The DCB subparameters BLKSIZE and BUFNO allow blocking of this data as it is placed on the mass storage device.

Notes:

When using a sequential scheduler:

- The input stream must be on a card reader or magnetic tape.
- Each job step and procedure step can be associated with only one data set in the input stream.
- The DD statement must be the last in the job step or procedure step.
- The records must be unblocked, and 80-characters in length.
- The characters in the records must be coded in BCD or EBCDIC.

When using a priority scheduler:

- 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.

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 31 shows the parameters of special interest for these purposes.

```
{ *    }
{ DATA }

  SYSOUT=A

  UNIT=name

  DCB=(subparameters)
```

Note: The DCB parameter can be specified, where permissible, for data sets on unit record devices. For example, it can be specified for compiler data sets (other than SYSUT1, SYSUT2, SYSUT3, and SYSUT4) and data sets specified by the DD statements required for the ACCEPT and DISPLAY statements, when any of these data sets are assigned to unit-record devices.

Figure 31. 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. This statement must be the last DD statement for the job step. The data must be followed by a delimiter statement.

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 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 Data Sets," and Example 1 in "Retrieving 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.

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 IBM System/360 Operating System: Data Management Services.

DATA CONTROL BLOCK

Each 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 overridden by 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 32).

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 IBM System/360 Operating System: Data Management Services.

ERROR PROCESSING FOR COBOL FILES

System Error Recovery

During the processing of a COBOL file, data transmission to or from an input/output device may not be successful the first time it is attempted. If it is not successful, standard error recovery routines, provided by the operating system, attempt to clear the failure and allow the program to continue uninterrupted.

If an input/output error cannot be corrected by the system, an abnormal termination (ABEND) of the program may occur unless the programmer has specified some means of error analysis. Error processing routines initiated by the programmer are discussed in the following paragraphs, and in "Appendix G: Input/Output Error Conditions."

For sequential files, the programmer can specify a DD statement option (EROPT) that specifies the type of action to be taken by the system if an error occurs. This option can be specified whether or not a declarative is written. If a declarative is specified, the DD statement option is executed when a normal exit is taken from the declarative. See "Accessing a Standard Sequential File" for further information.

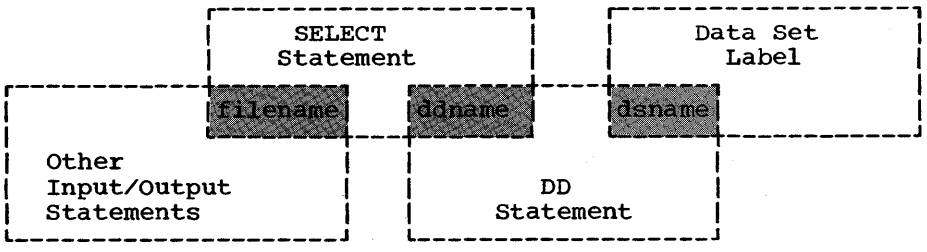


Figure 32. Links between the SELECT Statement, the DD Statement, the Data Set Label, and the Input/Output Statements

INVALID KEY Option

INVALID KEY errors may occur for files accessed randomly, or for output files accessed sequentially. A test to determine these errors may be made by using the INVALID KEY option of the READ, WRITE, REWRITE, or START verb.

Note: Secondary space allocation must be specified when the INVALID KEY option is used in a WRITE statement for QSAM and BSAM.

USE AFTER ERROR Option

The programmer may specify the USE AFTER ERROR option in the declarative section of the Procedure Division to determine the type of the input/output error. With the USE AFTER ERROR option, the programmer can pass control to an error-processing routine to investigate the nature of the error. If the GIVING option of the USE AFTER ERROR declarative is specified, data-name-1 will contain information about the error condition. Data-name-2, if specified, will contain the block in error if the last input/output operation was a read. If the file was opened as output, data-name-2 in the GIVING option cannot be referenced.

Data-name-2 of the GIVING option contains valid data only if data was actually transferred on the last input/output operation. For example, if the declarative is entered after execution of a START verb for a QISAM file on which no INVALID KEY option was present, an attempt to access data-name-2 results in an abnormal termination, because no transfer of data has taken place. If data-name-2 is specified in other than the Linkage Section, an abnormal termination will occur on entry to the USE ERROR declarative, if the declarative is invoked by an I/O request other than a READ (or any READ error in which no data transfer has taken place). Therefore, data-name-2 should be specified in the Linkage Section, and should be referred to only if the error is a READ error in which data transfer took place. This can be determined by examining data-name-1.

Either or both the INVALID KEY clause and the USE AFTER ERROR declarative may be specified for a file. If both have been specified and an INVALID KEY error occurs, the imperative-statement specified in the INVALID KEY option will be executed. If both have been specified and any other type of input/output error occurs the USE AFTER ERROR declarative will be entered. For a file other than standard sequential, if an I/O error occurs which is not INVALID KEY, and the USE ERROR declarative is not active for the file, the execution will be terminated in a manner equivalent to a STOP RUN. However, if such an error occurs in a sort, input, or output procedure, the execution will abnormally terminate. Table 18 is a generalized summary of the means available for recovery from an invalid key condition or an input/output error. Table 19 lists the error processing facilities available for each type of file organization. The following discussion summarizes the action taken by each facility for each type. For further information on the USE AFTER ERROR option, see the publication IBM System/360 Operating System: Full American National Standard COBOL.

STANDARD SEQUENTIAL

- Operating System: If the error cannot be corrected (read only), the program will ABEND in the absence of a DD statement option, USE AFTER STANDARD ERROR declarative, or INVALID KEY option. If both the DD statement option and USE section are specified, the control program will execute the USE declarative first and then the DD option if normal exit is taken from the declarative section. If no EROPT subparameter is indicated, or if ABS is specified and a USE AFTER STANDARD ERROR declarative exists, the declarative will receive control. After a normal exit, the job will abnormally terminate.

- DD Statement Option: The EROPT subparameter in the DCB parameter specifies one of three actions: accept the error block (ACC), skip the error block (SKP), or terminate the job (ABE).

- INVALID KEY: A transfer of control to the procedure indicated in the INVALID KEY phrase occurs if additional space cannot be allocated to write the record requested. This condition occurs when either no more space is available or 16 extents have already been allocated on the last volume assigned to the data set. The transfer of control occurs only if a secondary-quantity is specified in the DD statement SPACE, SPLIT, or SUBALLOC parameter. If no secondary-quantity is specified, the primary-quantity is assumed to be the exact amount of space required for the data set, and any attempt to write a record beyond the storage allocated

causes the program to end abnormally. When an INVALID KEY error occurs, the file can be closed so that it may subsequently be reopened for retrieval as INPUT or I-O.

- USE AFTER STANDARD ERROR: The programmer may specify this option in order to display the cause of the error. Control goes to the declarative section; the programmer can then display a message indicating the error and execute his DD statement option on a normal exit from the declarative section.

INDEXED (RANDOM)

- INVALID KEY: If the error is caused by an invalid key, recovery is possible. If the error is not an invalid key and the USE AFTER ERROR option is not specified, the program is terminated.

- USE AFTER STANDARD ERROR: Control goes to the declarative section. The programmer can check the error type in the section by specifying data-name-1 in the GIVING option. If the error is caused by a key error or the "no space found" condition, recovery is possible. On a READ error, the block can be skipped by executing additional READ statements. If the error persists (more bad READ statements than the blocking factor), processing is limited to a CLOSE statement. Any other error cannot be corrected. The program may continue executing, but processing of the file is limited to CLOSE. If the programmer closes the file, he may do so in either the declarative section or in the main body of his program.

INDEXED (SEQUENTIAL)

A. WRITE (load mode)

- Operating System: If the error cannot be corrected, the program will ABEND unless an error processing option is specified.

- INVALID KEY: If the error is caused by an invalid key, recovery is possible. (The programmer may attempt to reconstruct the key and retry the operation, or may bypass the error record.)

- USE AFTER STANDARD ERROR: Control goes to the declarative section. The programmer can check the error type in the section by specifying data-name-1 in the GIVING option. If the error is the result of a key error, recovery is possible. If the error is not a key error, the error cannot be corrected. The program may continue executing, but processing of the file is limited to CLOSE. If the programmer closes the file, he may do so in either the declarative section or in the main body of his program.

Table 18. Recovery from an Invalid Key Condition or from an Input/Output Error

Error \ Specified in COBOL Program	INVALID KEY option	USE AFTER STANDARD ERROR	Both	Neither on This Statement	Neither in Entire Program
Invalid key	Go to invalid key routine	Go to user's routine	Go to invalid key routine	Error ignored; next sequential instruction executed	Abend
Input/Output Error	Return to system	Go to user's routine	Go to user's routine	Return to system	Abend

Table 19. Input/Output Error Processing Facilities

Error Processing Facility Available / Organization	Operating System	DD Statement Option	COBOL Clauses: INVALID KEY	COBOL Clauses: USE AFTER STANDARD ERROR
Sequential	X	X	Note 1	X
Indexed (Sequential)				
WRITE	X		X	X
READ	X		Note 2	X
Indexed (Random)	Note 3		X	X
Direct or Relative (Sequential)	X		Note 1	X
Direct or Relative (Random)	X		X	X

Notes:
1. Holds only for WRITE.
2. Error cannot be caused by an invalid key.
3. No system error processing facility is available. If errors occur, they are ignored and processing continues, unless a programmer-specified error processing routine is specified.

B. READ, REWRITE (scan mode)

- Operating System: If the error cannot be corrected, the program will ABEND unless an error processing option is specified.
- INVALID KEY: The error cannot be caused by an invalid key. A source program coding error is implied and a compiler diagnostic message is generated.
- USE AFTER STANDARD ERROR: The programmer may specify this option in order to display the cause of the error. Control goes to the declarative section. The programmer can check the error type in the section by specifying data-name-1 in the GIVING option. Since the error cannot be caused by an invalid key, processing of the file is limited to CLOSE. If the programmer elects to close the file, he may do so in either the declarative section or in the main body of his program.

DIRECT or RELATIVE (RANDOM)

- Operating System: If the error cannot be corrected, the program will ABEND unless an error processing option is specified.
- INVALID KEY: If the error is caused by an invalid key, recovery is possible.
- USE AFTER STANDARD ERROR: Control goes to the declarative section. The programmer can check the error type in the section by specifying data-name-1 in the GIVING option. If the error is the result of a key error or the "no space found within the search limit" condition, recovery is possible. Any other error cannot be corrected. The program may continue executing, but processing of the file is limited to CLOSE. If the programmer closes the file, he may do so in either the declarative section or in the main body of his program.

DIRECT or RELATIVE (SEQUENTIAL)

- Operating System: If no error processing option is specified, a message is written to the console providing identification of the file and type of input/output error. Then control is returned to the system. For sequential data sets, if EROPT has SKP or ACC (as specified in the JCL for the data set), an ABEND will not occur and processing will continue.

- INVALID KEY: A transfer of control to the procedure indicated in the INVALID KEY phrase occurs if additional space cannot be allocated to write the record requested. This condition occurs when either no more space is available or 16 extents have already been allocated on the last volume assigned to the data set. The transfer of control occurs only if a secondary-quantity is specified in the DD statement SPACE, SPLIT, or SUBALLOC parameter. If no secondary-quantity is specified, the primary-quantity is assumed to be the exact amount of space required for the data set, and any attempt to write a record beyond the storage allocated causes the program to end abnormally. When an INVALID KEY error occurs, the file can be closed so that it may subsequently be reopened for retrieval as INPUT or I-O.

- USE AFTER STANDARD ERROR: The programmer may specify this option in order to display the cause of the error. Control goes to the declarative section. The programmer can check the error type in the section by specifying data-name-1 in the GIVING option. If the error is not the result of an invalid key, processing of the file is limited to CLOSE. If the programmer elects to close the file, he may do so in either the declarative section or in the main body of his program.

Notes: The user should consider the following points 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 INVALID KEY clause. When an input/output operation is requested, a storage area (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 to 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.

2. The error declarative dynamically allocates storage for a register save area upon entry. If a GO TO statement is used to exit from the declarative, neither this save area nor the IOB is freed. To make the maximum space available to other users, the programmer should rely on the declarative as much as possible, taking a normal exit from it. Otherwise, it is recommended that the programmer specify a larger region.

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.

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 IBM System/360 Operating System: Data Management Services and IBM System/360 Operating System: Tape Labels, Order No. GC28-6680.

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 LABEL PROCESSING

Standard label processing as performed by the system consists of the following basic functions:

- Checking the labels on input data sets to ensure that the correct volume is mounted, and to identify, describe, and protect the data set being processed.
- Checking the existing labels on output data sets to ensure that the correct volume is mounted and to prevent overwriting of vital data.
- Creating and writing new labels on output data sets.

When a data set is opened for input, the volume label and the header labels are processed. For an input end-of-data condition, the trailer labels are processed when a CLOSE statement is executed. For an input end-of-volume condition, the trailer labels on the current volume are processed, and then the volume label and header labels on the next volume are processed.

When a data set is opened for output, the existing volume label and HDR1 label are checked, and new header labels are written. For an output end-of-volume condition, trailer labels are written on the current volume, the existing volume labels and header labels on the next volume are checked, and then new header labels are written on the next volume. When an output data set is closed, trailer labels are written.

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 standard 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. 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.

For further information on user label totaling, see the publication IBM System/360 Operating System: Full American National Standard COBOL.

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 programmer must:

- 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 as part of the SVC library (SYS1.SVCLIB).

- 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 into transient areas. 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 of 1024 bytes each.
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 Procedures."

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. A more detailed discussion of the USE...LABEL PROCEDURE statement is contained in the publication IBM System/360 Operating System: American National Standard COBOL.

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 IBM System/360 Operating System: System's Programmer's Guide.

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 33).

Code	Exit List
1	(USE section for header labels)
2	(USE section for trailer labels)
.	.
.	.
.	.

Note:
Code 1 is set to X'01' indicating INPUT, or X'02' indicating OUTPUT.
Code 2 is set to 'X'OD' indicating INPUT, or X'04' indicating OUTPUT.

Figure 33. 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 34) 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 34. 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 extend 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.

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 35. Label Routine Returns Codes

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 35). 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 +38.

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 System/360 Operating System: System Programmer's Guide.

RECORD FORMATS

Logical records may be in one of four formats: 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 36. The optional control character, represented by the letter C in Figure 36 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. This position will be automatically filled in with the carriage control or stacker select character. 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 the key field required when direct organization is used.

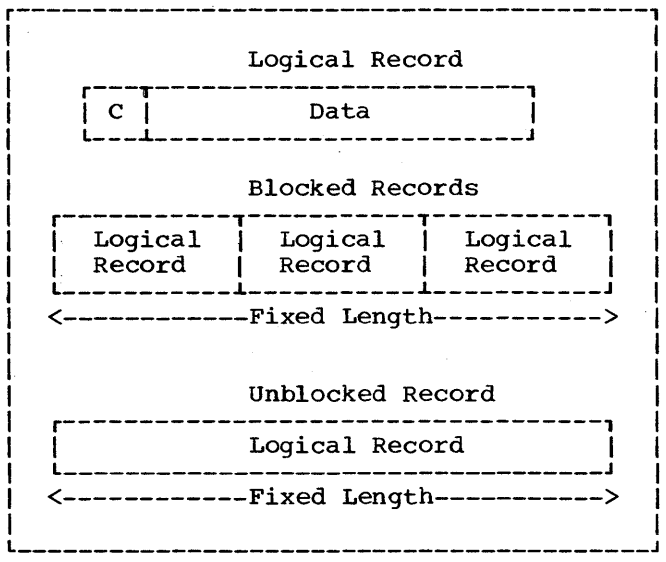


Figure 36. Fixed-Length (Format F) Records

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 37. 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 standard 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 the key field required when direct organization is used.

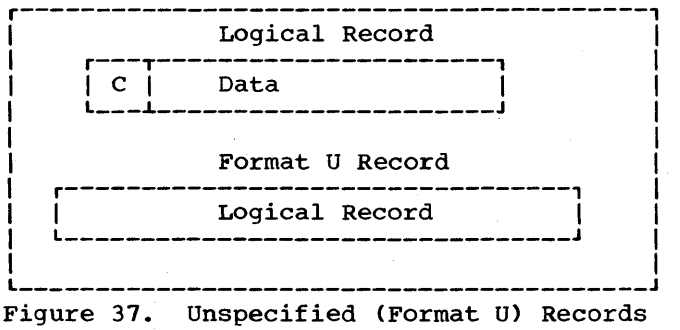


Figure 37. Unspecified (Format U) Records

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 standard 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 standard 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 38 and 39.

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 38), the Data portion + CC + cc constitute the block.

For blocked V-mode records (Figure 39), the Data portion of each record + the cc of each record + CC constitute the block.

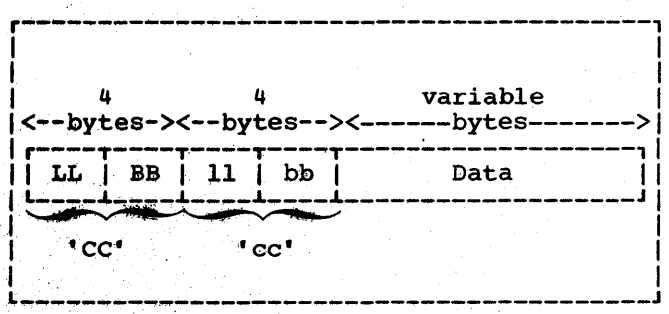


Figure 38. Unblocked V-Mode Records

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 or S-mode file, the size of the current record for that file is used as the source field length in the MOVE statement. All other rules of the MOVE statement apply.

Example 1:

Consider the following standard 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. 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 8 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.

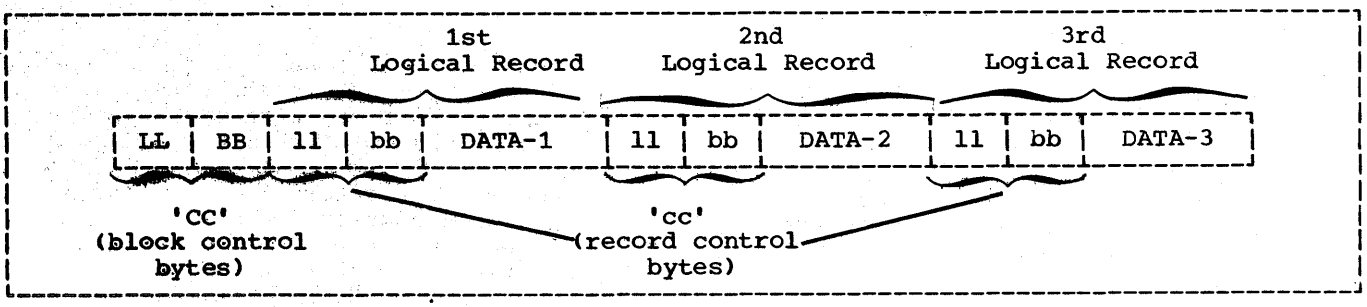


Figure 39. Blocked V-Mode Records

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 40.

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 41.

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 standard 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 VARAIBLE-FILE-2 will appear on the external storage device as shown in Figure 42.

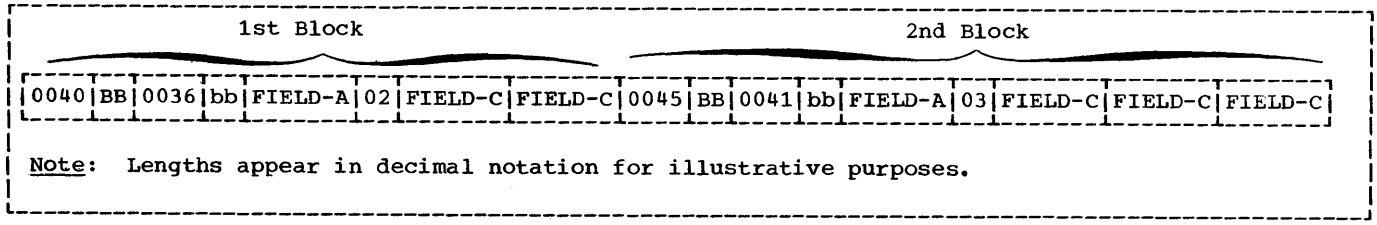


Figure 40. Fields in Unblocked V-Mode Records

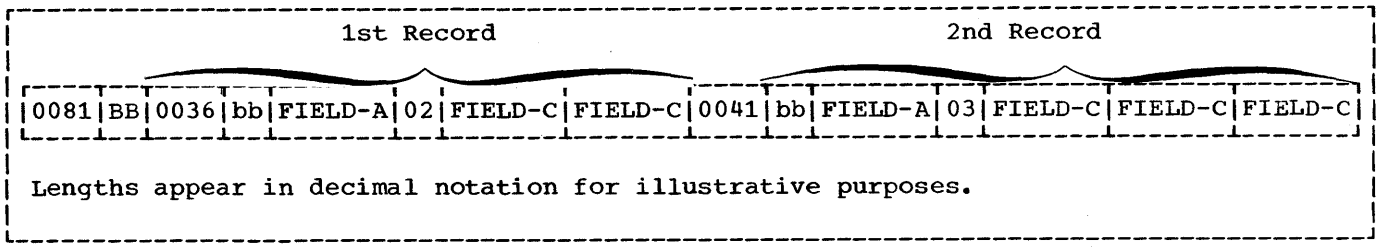


Figure 41. Fields in Blocked V-Mode Records

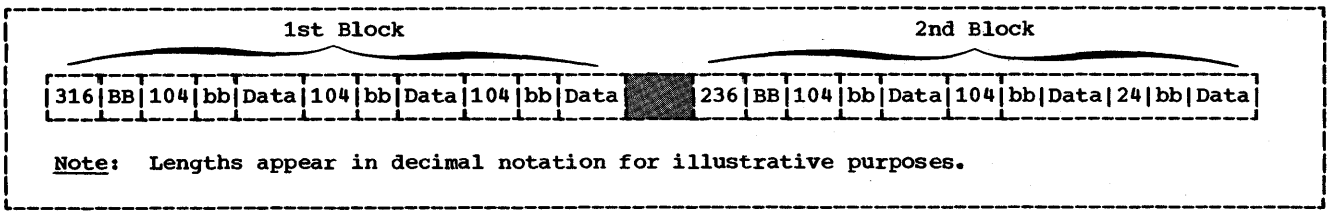


Figure 42. First Two Blocks of VARIABLE-FILE-2

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 the key field required when direct organization is used.

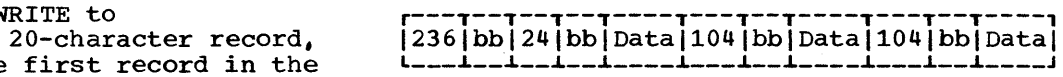
APPLY WRITE-ONLY Clause

The APPLY WRITE-ONLY clause is used to make optimum use of buffer space when creating a standard sequential file with blocked V-mode records.

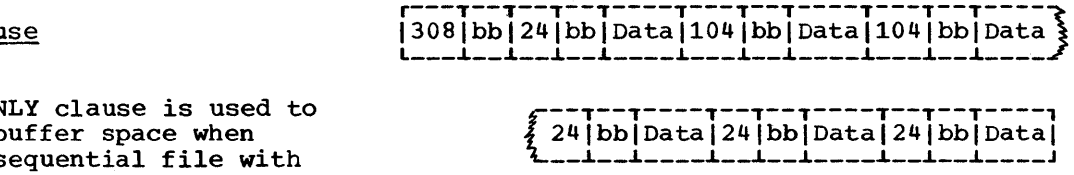
Suppose VARIABLE-FILE-2 is being created with the following file description entry:

```
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 turncated 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 standard 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 43 illustrates the control fields.

BDF (Block Descriptor Field):

LL -- represents 2 bytes designating the length of the physical block (including the block descriptor field itself).

BB -- represents 2 bytes reserved for system use.

SDF (Segment Descriptor Field):

ll -- represents 2 bytes designating the length of the record segment (including the segment descriptor field itself).

bb -- represents 2 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 44 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 44 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 support the 2400 tape series, the 2311 and 2314 disk storage devices, and the 2321 data cell drive.

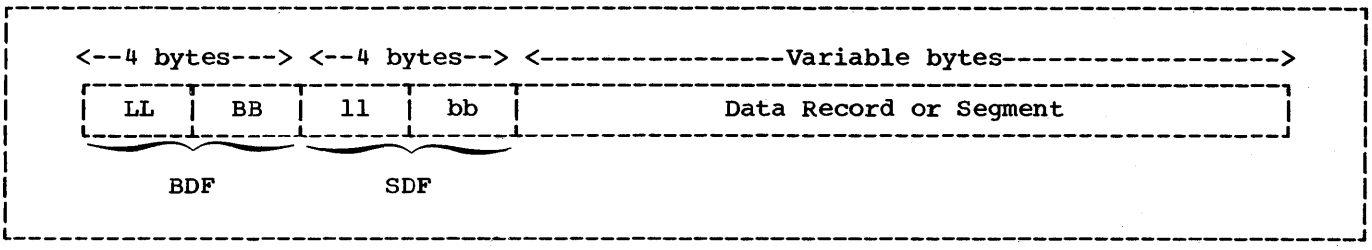


Figure 43. Control Fields of an S-Mode Record

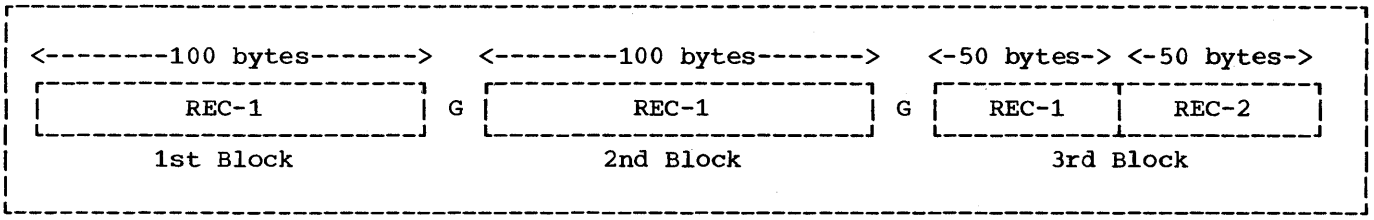


Figure 44. One Logical Record Spanning Physical Blocks

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.

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 45 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.

Note:

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.

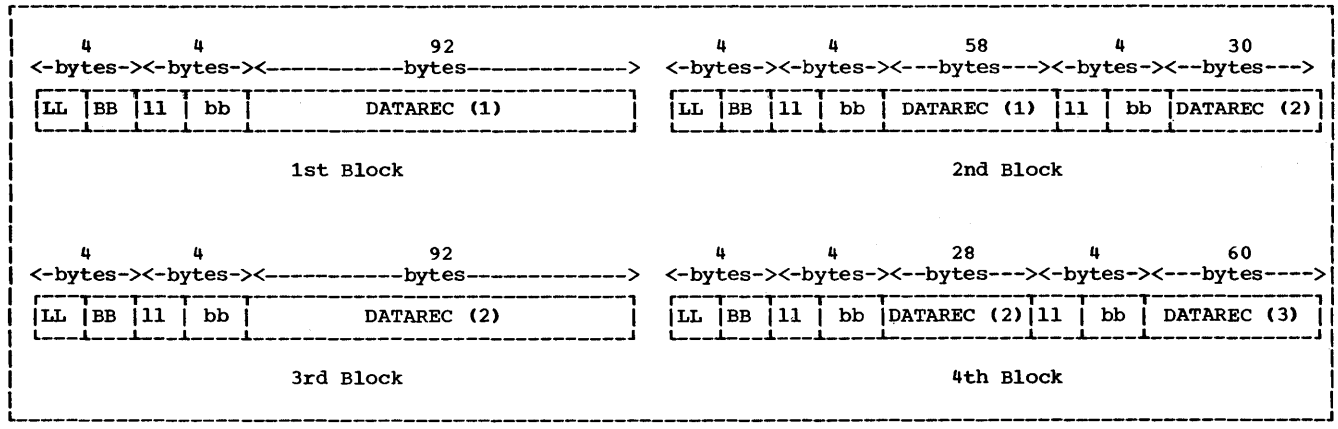


Figure 45. First Four Blocks of SPAN-FILE

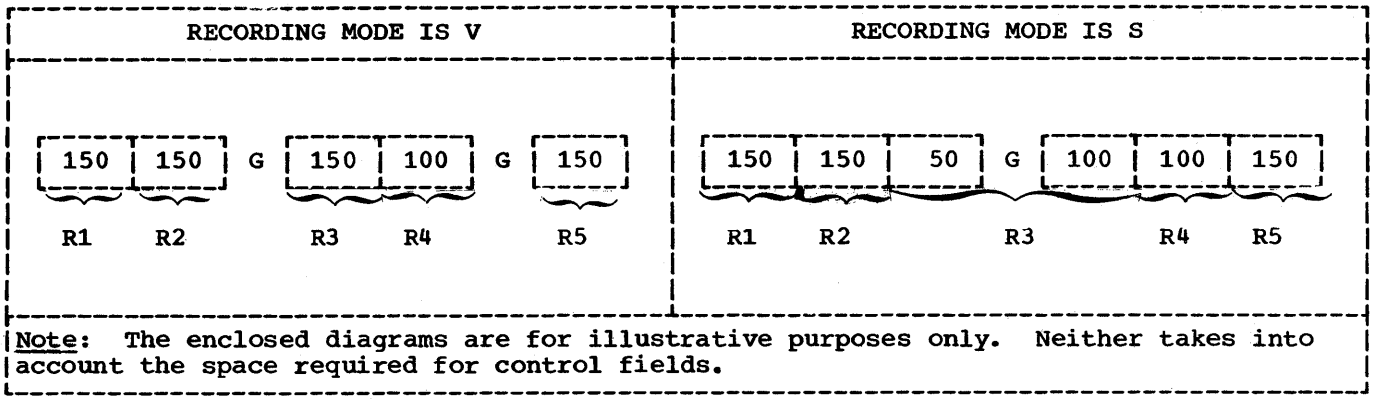


Figure 46. Advantage of S-Mode Records Over V-Mode Records

For V-mode records, buffers are truncated if the next logical record is too large to be completely contained in the block (Figure 46). 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.
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 long 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 46).

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

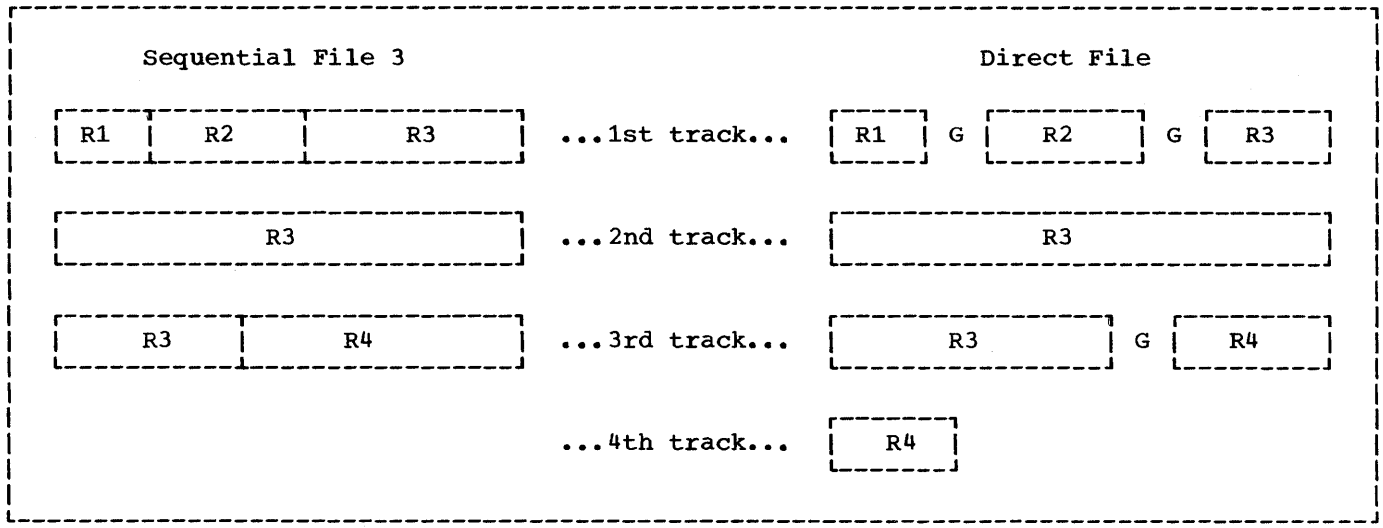


Figure 47. Direct and Sequential Spanned Files on a Mass Storage Device

The size of a 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.

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 3,625 characters, any record smaller than 3,625 characters may be written as a single physical block. If a logical record is greater than 3,625 characters, the record is segmented. The first segment may be contained in a physical block of up to 3,625 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 47 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 3625 CHARACTERS (track capacity = 3,625)

2. RECORD CONTAINS 500 TO 5000 CHARACTERS

In the sequential file, each physical block is 3,625 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 2,000 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 1,000 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 record is 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 the 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 0 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 48).

```
FD  INPUT-FILE
    .
    .
    .
    DATA RECORDS ARE RECORD-1 RECORD-2 RECORD-3.

01  RECORD-1.
    02  CONTROL-1              PIC 99.
    02  FIELD-1 OCCURS 0 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 0 TO 10 TIMES DEPENDING ON CONTROL-3  PIC X(4).
```

Figure 48. Calculating Record Lengths When Using the OCCURS Clause with the DEPENDING ON Option

If the BLOCK CONTAINS clause is omitted, the buffer size is calculated from the longest level-01 record description entry. In Figure 48, 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 48). 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 48, 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 48 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.

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 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 5 TIMES
        DEPENDING ON F.


WORKING-STORAGE SECTION.
    .
    .
    .
01  TABLE-3.
    05  H OCCURS 10 TIMES DEPENDING ON B.


01  TABLE-4.
    05  I OCCURS 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 9.
    05  FIELD-2 OCCURS 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 99.
    05  CONTROL-2     PIC 99.
    05  VARY-FIELD-1 OCCURS 10 TIMES
        DEPENDING ON CONTROL-1 PIC X(5).
    05  VARY-FIELD-2 OCCURS 10 TIMES
        DEPENDING ON CONTROL-2 PIC X(9).

01  STORE-VARY-FIELD-2.
    05  VARY-FLD-2 OCCURS 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 VARY-FIELD-2 TO STORE-VARY-FIELD-2.
ADD 1 TO CONTROL-1.
MOVE 'additional field' TO
  VARY-FIELD-1 (CONTROL-1).
MOVE STORE-VARY-FIELD-2 TO VARY-FIELD-2.
```

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 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

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 49 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 49 is an explanation of the illustration.

```
(1) //TEST   JOB  NY16090101,'SCHOEN 1',MSGLEVEL=1,CLASS=C
    //JOBLIB  DD   DSN=PRODTEST,DISP=SHR
    //STEP1   EXEC PGM=IKFCBL00,PARM='DMAP,PMAP,XREF,QUOTE',REGION=86K
    //SYSUT1  DD   DSNAME=&&UT1,UNIT=SYSDA,SPACE=(TRK,(100,10))
    //SYSUT2  DD   DSNAME=&&UT2,UNIT=SYSDA,SPACE=(TRK,(100,10))
    //SYSUT3  DD   DSNAME=&&UT3,UNIT=SYSDA,SPACE=(TRK,(100,10))
    //SYSUT4  DD   DSNAME=&&UT4,UNIT=SYSDA,SPACE=(TRK,(100,10))
    //SYSLIN  DD   DSNAME=&&PNCH,UNIT=SYSDA,SPACE=(TRK,(100,10)),          X
    //             DISP=(NEW,PASS)
    //SYSPRINT DD  SYSOUT=A
    //SYSIN   DD   *
(2) IEF236I ALLOC. FOR TEST     STEP1
    IEF237I 234    ALLOCATED TO JOBLIB
    IEF237I 190    ALLOCATED TO SYSUT1
    IEF237I 230    ALLOCATED TO SYSUT2
    IEF237I 190    ALLOCATED TO SYSUT3
    IEF237I 235    ALLOCATED TO SYSUT4
    IEF237I 190    ALLOCATED TO SYSLIN
    IEF237I 00C    ALLOCATED TO SYSIN
```

Figure 49. Examples of Compiler Output (Part 1 of 3)

```
          .
          .
          .
 00018   100180 DATA DIVISICN.
 00019   100190 FILE SECTION.
 00020   100200 FD  FILE-1
 00021   100210     LABEL RECORDS ARE CMITTED
 00022   100220     BICCK CCNTAINS 100 CHARACTERS
 00023   100225     RECORD CONTAINS 20 CHARACTERS
 00024   100230     RECCRDING MCDE IS F
 00025   100240     DATA RECORD IS RECORD-1.
 00026   100250   01 RECCRD-1.
 00027   100260     02 FIELD-A PICTURE IS X(20).
 00028   10027C FD  FILE-2
          .
          .
          .
 00074   100730 STEP-6. READ FILE-2 RECORD INTO WCRK-RECORD AT END GO TO STEP-8.
 00075   100740 STEP-7. IF NC-CF-DEPENDENTS IS ECUAL TO "0" MOVE "Z" TO
 00076   100750     NO-OF-DEPENDENTS. EXHIBIT NAMED WORK-RECORD. GC TO
 00077   100760     STEP-6.
 00078   10077C STEP-8. CLOSE FILE-2.
 00079   100780     STOP RUN.
```

(3)

```
      A           B     C                                D       E        F             G           H            I
 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=1-167      01  RECORD-1                           BL=1    000     DNM=1-167     DS 0CL20    GROUP
 DNM=1-188      02  FIELD-A                            BL=1    000     DNM=1-188     DS 20C      DISP
 DNM=1-205      FD  FILE-2                           DCB=02            DNM=1-205                 QSAM                F
 DNM=1-224      01  RECORD-2                           BL=2    000     DNM=1-224     DS 0CL20    GROUP
```

(4)

```
      A
           MEMORY MAP

        TGT                        00230

 SAVE AREA                         00230
 SWITCH                            00278
 TALLY                             0027C
 SORT SAVE                         00280
 ENTRY-SAVE                        00284
      B
 LITERAL POOL (HEX)

 004A8 (LIT+0)     004805EF  00228000  C9D2C6F9  F9F9C940  E4D5E2F4  C3C3C5E2
 004C0 (LIT+24)    E2C6E4D3  40D6D7C5  D540C6D6  D9400000  00000000  00000200
 004D8 (LIT+48)    40200001  C0000CC1  001A4800  F0E90000  C0000000

        DISPLAY LITERALS (BCD)

 004EC (LTL+68)   'WCRK-RECCRD'
      C
        PGT                        00458

 OVERFLCW CELLS                    00458
 VIRTUAL CELLS                     00458
 PROCEDURE NAME CELLS              00464
 GENERATED NAME CELLS              00478
```

(5)

```
 REGISTER ASSIGNMENT

  REG 6    BL =3
  REG 7    BL =1
  REG 8    BL =2
```

(6)

```
 A        B              C         D                 E          F                         G
 55      *BEGIN
                     0004F8                        START      EQU   *
                     0004F8   58 F0 C 004                     L     15,004(0,12)          V(ILBODSP0)
                     0004FC   05 1F                           BALR  1,15
                     0004FE   0C0140                          DC    X'000140'
                     000501   04F5F5404040                    DC    X'04F5F5404040'
 55        READY     0005C8   96 40 D 048                     OI    048(13),X'40'         SWT+0
 58      *STEP-1
                     00050C   58 F0 C 004                     L     15,004(0,12)          V(ILBODSP0)
                     000510   05 1F                           BALR  1,15
                     000512   000140                          DC    X'000140'
                     000515   04F5F8404040                    DC    X'04F5F8404040'
```

(7)

Figure 49. Examples of Compiler Output (Part 2 of 3)

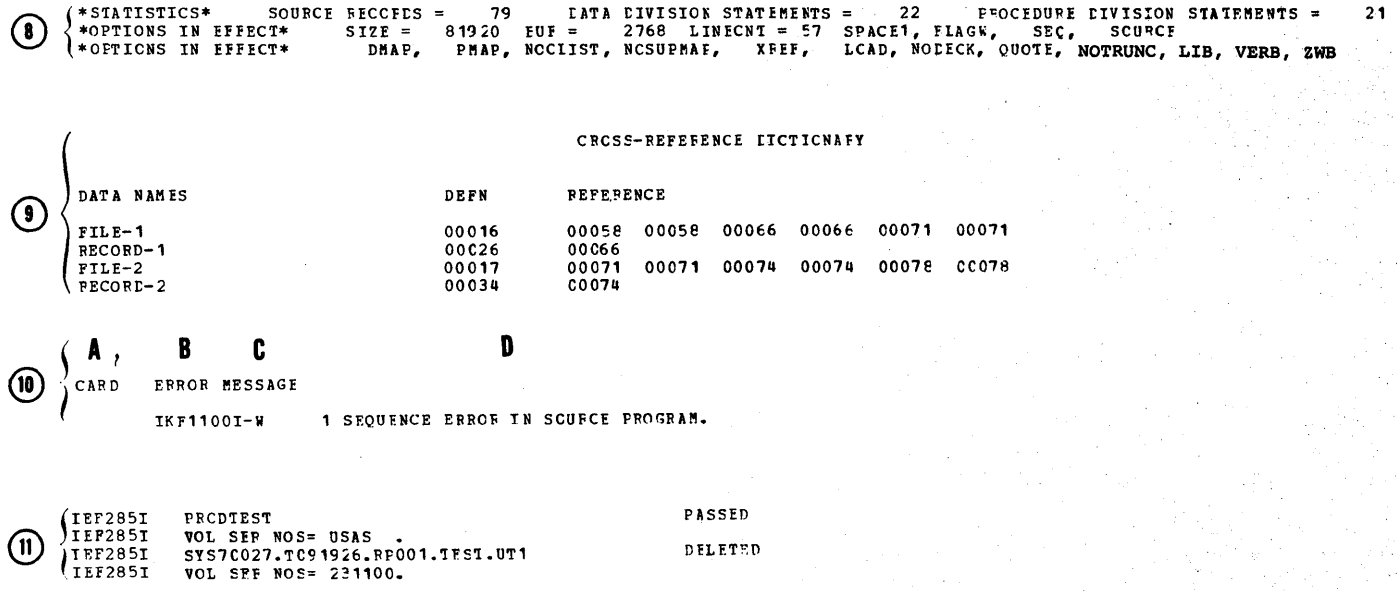


Figure 49. Examples of Compiler Output (Part 3 of 3)

1. Listing of job control statements associated with this job step. These statements are listed because MSGLEVEL=1 is specified in the JOB statement.

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 190 ALLOCATED TO SYSUT1

   indicates that the data set for SYSUT1 has been assigned to the device 190.

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 listing, and in the object code listing. 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, and RD 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 and DECB, if any.

G. This column defines storage for each data item. It is represented in assembler-like terminology. Table 20 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

5. Global Tables and Literal Pool: The global table is listed when the PMAP 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-6, namely:

```
READ FILE-6.
```

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.

6. Register Assignment: This lists the register assigned to each base locator in the object program.

7. 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. The number refers to the COBOL statement in the source module that contains the verb which is listed under column B.

B. The COBOL procedure-name and the COBOL verb. The procedure-name is indicated by an asterisk.

The statement within which the COBOL verb appears determines the information under columns C, D, F, and G.

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.

F. The object code instruction in the form that closely resembles assembler language (displacements are in hexadecimal notation).

G. Compiler-generated information about the operands of the generated instruction. This includes names and relative locations of literals. Table 21 refers to information in this column.

Note: The programmer can produce a condensed listing by specifying CLIST as an option in place of MAP or PMAP. The CLIST option produces only the source card number, the EBCDIC representation of the verb-name if the VERB option is in effect, and the location of the first generated instruction, as follows:

54	READY	000380	57	OPEN	000394
57	MOVE	0003B8	61	ADD	000444
61	ADD	000450	61	MOVE	00045C

8. 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.

9. Cross Reference Dictionary: The XREF dictionary is produced when the XREF option is specified. It consists of two parts:

A. The XREF dictionary for data names followed by the generated number of the card on which the statement begins. 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 of the card on which the statement begins.

All the names appear in the order in which they are defined in the source program. The number of references appearing for a given name is based on the number of times the name is referred to in the generated code.

10. Diagnostic messages: The diagnostic messages associated with the compilation are always listed. The format of the diagnostic message is:

A. Compiler-generated line 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 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 D-level error is detected will generate a return code 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 generates 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."

A complete list of compiler diagnostic messages is contained in "Appendix J: Diagnostic Messages." Multiple console support considerations for console messages are discussed there also.

11. 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 resides.

Table 20. Glossary Definition and Usage

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.

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 IBM System/360 Operating System: 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.

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

Table 21. Symbols Used in the Listing and Glossary to Define Compiler-Generated Information

Symbol	Definition
DNM	Source Data Name
SAV	Save Area Cell
SAV2	Input/Output Error Save Cell
SAV3	OPEN Parameter
SWT	Switch Cell
TLY	Tally Cell
WC	Working Cell
TS	Temporary Storage Cell
TS2	Temporary Storage (Non-Arithmetic)
TS3	Temporary Storage (Synchronization)
TS4	Temporary Storage (Table-Handling)
VLC	Variable Length Cell
SBL	Secondary Base Locator
BL	Base Locator
BLL	Base Locator for Linkage Section
ON	On Counter
PFM	Perform Counter
PSV	Perform Save
VN	Variable Procedure Name
DEC	DECB Address
SBS	Subscript Address
XSW	Exhibit Switch
XSA	Exhibit Save Area
PRM	Parameter
PN	Source Procedure Name
GN	Generated Procedure Name
DCB	DCB Address
VNI	Variable Name Initialization
LTL	Literal
INX	Index Cell
V(BCDNAME)	Virtual
RSV	Report Save Area
SSV	Sort Save Area
CKP	Checkpoint Counter

```
(1) //STEP2  EXEC  PGM=IEWL,PARM='XREF',REGION=96K
    //SYSUT1 DD    DSNAME=&&UT1,UNIT=SYSDA,SPACE=(TRK,(100,10))
    //SYSLMOD DD   DSNAME=&&GJCB(GO),UNIT=SYSDA,SPACE=(TRK,(100,10,1)),     X
    //             DISP=(NEW,PASS)

(2) IEF236I ALLOC. FOR TEST     STEP2
    IEF237I 234   ALLOCATED TO JCBLIB
    IEF237I 190   ALLOCATED TO SYSUT1
    IEF237I 230   ALLOCATED TO SYSLMOD

(3) F88-LEVEL LINKAGE EDITOR OPTIONS SPECIFIED XREF
          VARIABLE OPTIONS USED - SIZE=(153600,51200)              DEFAULT OPTION(S) USED

                                                CROSS REFERENCE TABLE
(5)       A                                     B
    CONTROL SECTION                           ENTRY
      NAME     ORIGIN   LENGTH                  NAME   LOCATION     NAME   LOCATION     NAME   LOCATION     NAME   LOCATION
    TESTRUN       00      88C
    ILBODSP0*    890      69A
    ILBOSTP0*    F30      11C
                                               ILBOSTP1    1016     PDTSZE      104B

(6)       A              B              C
        LOCATION  REFERS TO SYMBOL  IN CONTROL SECTION          LOCATION  REFERS TO SYMBOL  IN CONTROL SECTION
           458            ILBOSTP0       ILBCSTP0                 45C            ILBODSP0       ILBODSP0
           460            ILBCSTP1       ILBCSTP0

(5)   C ENTRY ADDRESS       00
      D TOTAL LENGTH      1050

(4)       ****GO        DOES NOT EXIST BUT HAS BEEN ADDED TO DATA SET

(7) IEF285I   PRODTEST                                    PASSED
    IEF285I   VOL SER NOS= USAS  .
    IEF285I   SYS70027.T091926.RP001.TEST.UT1             DELETED
    IEF285I   VOL SER NOS= 231100.
```

Figure 50. Linkage Editor Output Showing Module Map and Cross Reference List

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 50 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 50 is an explanation of the illustration.

1. The job control statements. These statements are listed because MSGLEVEL=1 is specified on the JOB statement for this job, shown in Figure 49.

2. Allocation messages from the job scheduler. These messages provide information about the device allocation for the data sets in the job step. For example, the message

   IEF2371 190 ALLOCATED TO SYSUT1

   indicates that the data set for SYSUT1 has been assigned to the device 190.

3. Linkage editor informative message. This message lists the PARM options that were specified.

4. Linkage editor informative message. 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, 460 is the location where another program is called. ILBOSTP1 is the entry point of the called program. ILBOSTP0 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 may generate informative or diagnostic messages. A complete list of these messages is included in the publication IBM System/360 Operating System Linkage Editor and Loader.

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 diagnostic messages is found in the publication IBM System/360 Operating System: Linkage Editor and Loader.

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 51 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

A dump as well as system diagnostic messages are generated automatically if a program contains errors that cause abnormal termination.

Figure 52 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 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 publication IBM System/360 Operating System: Messages and Codes.
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.

Figure 51. Module Map Format Example

```
                                        OS/360 LOADER

OPTIONS USED - PRINT,MAP,NOLET,CALL,NORES,SIZE=424176

    NAME   TYPE  ADDR       NAME   TYPE  ADDR       NAME   TYPE  ADDR       NAME   TYPE  ADDR       NAME   TYPE  ADDR

  SAMPL2B    SD   161E0    SAMPL2BA  SD   16EC8    IHEMAIN    SD   17CF8    IHENTRY    SD   17D00    IHESPRT    SD   17D10
  SYSIN      SD   17D48    IHEVQC  * SD   17D80    IHEVQCA  * LR   17D80    IHEVQB   * SD   17FD8    IHEVQBA  * LR   17FD8
  IHEDIA   * SD   183C0    IHEDIAA * LR   183C0    IHEDIAB  * LR   183C2    IHEVPE   * SD   18608    IHEVPEA  * LR   18608
  IHEVPA   * SD   18870    IHEVPAA * LR   18870    IHEVFC   * SD   189D0    IHEVFCA  * LR   189D0    IHEVPC   * SD   189F8
  IHEVPCA  * LR   189F8    IHEVFE  * SD   18BE8    IHEVFEA  * LR   18BE8    IHEVSC   * SD   18C08    IHEVSCA  * LR   18C08
  IHEDNC   * SD   18CB8    IHEDNCA * LR   18CB8    IHEDOA   * SD   18F30    IHEDOAA  * LR   18F30    IHEDOAB  * LR   18F32
  IHEDMA   * SD   19010    IHEDMAA * LR   19010    IHEVFD   * SD   19108    IHEVFDA  * LR   19108    IHEVFA   * SD   19160
  IHEVFAA  * LR   19160    IHEVPB  * SD   19248    IHEVPBA  * LR   19248    IHEXIS   * SD   193F0    IHEXISO  * LR   193F0
  IHEIOB   * SD   19488    IHEIOBA * LR   19488    IHEIOBB  * LR   19490    IHEIOBC  * LR   19498    IHEIOBD  * LR   194A0
  IHESARC  * LR   1A9C8    IHESADD * LR   1A9DE    IHESAFF  * LR   1AA18    IHEPRT   * SD   1AB70    IHEPRTA  * LR   1AB70
  IHEBEGA  * LR   1AE28    IHEERR  * SD   1AE68    IHEERRD  * LR   1AE68    IHEERRC  * LR   1AE72    IHEERRB  * LR   1AE7C
  IHEERRA  * LR   1AE86    IHEERRE * LR   1B4E2    IHEIOF   * SD   1B580    IHEIOFB  * LR   1B580    IHEIOFA  * LR   1B582
  IHEITAZ  * LR   1B81E    IHEITAX * LR   1B82A    IHEITAA  * LR   1B83E    IHEDCN   * SD   1B860    IHEDCNA  * LR   1B860
  IHEDCNB  * LR   1B862    IHEIOD  * SD   1BA50    IHEIODG  * LR   1BA50    IHEIODP  * LR   1BA52    IHEIODT  * LR   1BB4A
  IHEVTB   * SD   1BCF0    IHEVTBA * LR   1BCF0    IHEVQA   * SD   1BD78    IHEVQAA  * LR   1BD78



  IHEQINV    PR      00    IHEQERR    PR       4    SAMPL2BB   PR       8    SAMPL2BC   PR       C    IHEQSPR    PR      10
  SYSIN      PR      14    IHEQLSA    PR      18    IHEQLW0    PR      1C    IHEQLW1    PR      20    IHEQLW2    PR      24
  IHEQLW3    PR      28    IHEQLW4    PR      2C    IHEQLWE    PR      30    IHEQLCA    PR      34    IHEQVDA    PR      38
  IHEQFVD    PR      3C    IHEQCFL    PR      40    IHEQFOP    PR      48    IHEQADC    PR      4C    IHEQXLV    PR      50
  IHEQEVT    PR      58    IHEQSLA    PR      60    IHEQSAR    PR      64    IHEQLWF    PR      68    IHEQRTC    PR      6C
  IHEQSFC    PR      70


IEW1001  IHEUPBA
IEW1001  IHEUPAA
IEW1001  IHETERA
IEW1001  IHEM91C
IEW1001  IHEM91B
IEW1001  IHEM91A
IEW1001  IHEDDOD
IEW1001  IHEVPFA
IEW1001  IHEVPDA
IEW1001  IHEDBNA
IEW1001  IHEVSFA
IEW1001  IHEVSBA
IEW1001  IHEVCAA
IEW1001  IHEVSAA
IEW1001  IHEDNBA
IEW1001  IHEUPBB
IEW1001  IHEUPAB
IEW1001  IHEVSEB

  TOTAL LENGTH     5068
  ENTRY ADDRESS    17D00

IEW1001  WARNING - UNRESOLVED EXTERNAL REFERENCE (NOCALL SPECIFIED)
```

```
(1) //STEP3  EXEC  PGM=*.STEP2.SYSLMOD
    //SYSOUT DD    SYSOUT=A
    //SYSUDUMP DD SYSOUT=A
    //SAMPLE DD    UNIT=2400,LABEL=(,NL)

(2) IEF236I ALLOC. FOR TEST     STEP3
    IEF237I 234   ALLOCATED TO JOBLIB
    IEF237I 230   ALLOCATED TO PGM=*.DD
    IEF237I 183   ALLOCATED TO SAMPLE

(3) IEF285I   PRODTEST                                        PASSED
    IEF285I   VOL SER NOS= USAS  .
    IEF285I   SYS70027.T091926.RP001.TEST.GJOB                PASSED
    IEF285I   VOL SER NOS= 231400.
    IEF285I   SYSOUT                                          SYSOUT
    IEF285I   VOL SER NOS=       .
    IEF285I   SYS70027.T091926.RP001.TEST.R0000006            DELETED
    IEF285I   VOL SER NOS= I00001.

(4) 66
    62
    66
    71
    74
    75
    WORK-RECORD = A 0001 NYC Z
    74
    75
    WORK-RECORD = B 0002 NYC 1
    74
    75
    WORK-RECORD = C 0003 NYC 2
    74
    75
    WORK-RECORD = D 0004 NYC 3
    74
    75
    WORK-RECORD = E 0005 NYC 4
    74
    75
    WORK-RECORD = F 0006 NYC Z
    74
    75
    WORK-RECORD = G 0007 NYC 1
    74
    75
    WORK-RECORD = H 0008 NYC 2
    74
    75
    WORK-RECORD = I 0009 NYC 3
    74
    75
    WORK-RECORD = J 0010 NYC 4
    74
    75
    WORK-RECORD = K 0011 NYC Z

(5) IEA000A  INT REQ,00C,02,0E00,1000
    IEA000A  INT REQ,00C,02,0E00,0800
    IEF233A M 181,SCRTCH,NL,TEST,STEP3
    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
```

Figure 52. Execution Job Step Output

REQUESTS FOR OUTPUT

1. The programmer can request data to be displayed by using the DISPLAY statement.

2. Messages to the operator can also be displayed on the console when requested in the source program (DISPLAY UPON CONSOLE).

3. The programmer can request a full dump, in case his program is terminated abnormally, by including

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

   in the job control procedure.

Note: Under MVT, the SPACE parameter should also be included in the DD statement. For example:

```
//SYSABEND DD SYSOUT=A,                          X
//            SPACE=(125,(200,1000),RLSE)
```

Dumps are 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 J: Diagnostic Messages" under the heading "Object Time Messages." Multiple console support 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 these messages is found in the publication IBM System/360 Operating System: Messages and Codes.

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. Table 22 lists these codes, along with an identification of each.

Table 22. System Message Identification Codes

Code	Identification
IEA	An on-line console message from the supervisor.
IEC	An on-line console 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.
IET	A message from the assembler.
IEW	A message from the linkage editor.
IHB	A message from the supervisor and data management.

A programmer using the COBOL compiler under the System/360 Operating System has several methods available to him for testing and debugging his programs or revising them for increased efficiency of operation.

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.

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 debugging language and other methods of program checkout.

The three 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 section and 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.

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 (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.

Note: If an error occurs in an ON statement, the diagnostic message may refer to the previous statement number.

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).

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 53.

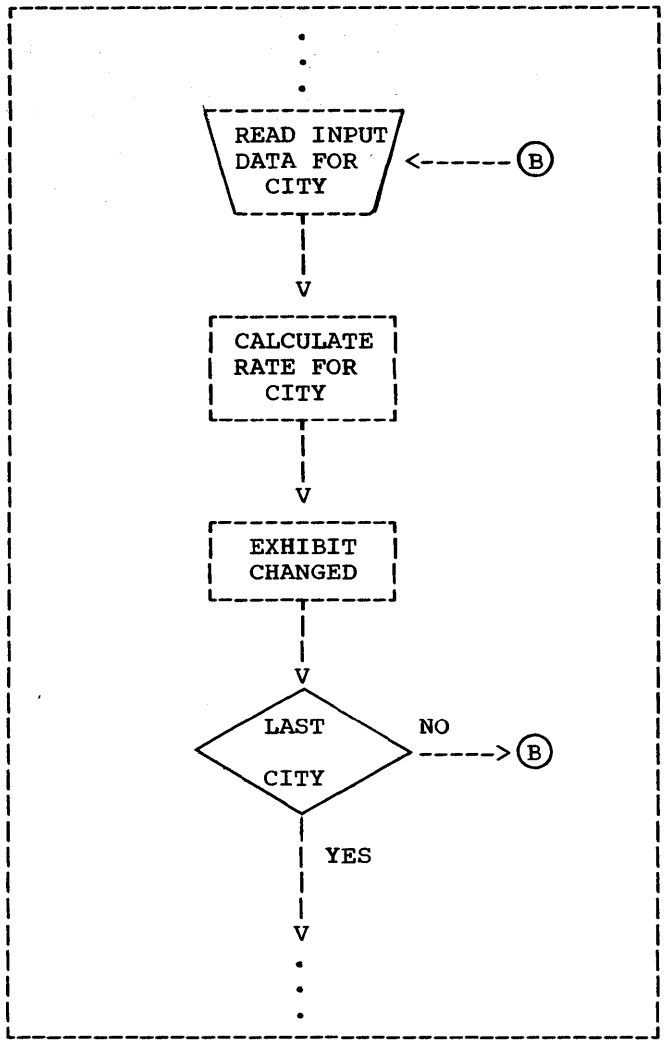


Figure 53. Example of Program Flow

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 54.

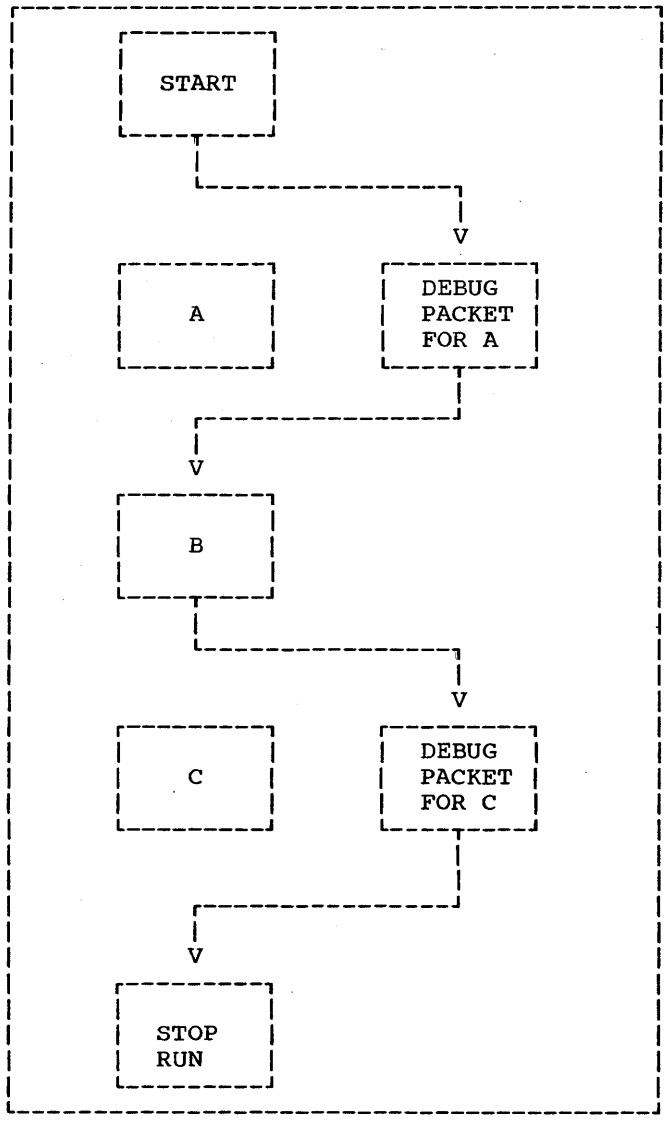


Figure 54. Selective Testing of B

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.

A debug packet can also be used in conjunction with the BASIS card to debug a program or to test deletions or additions to it. The debug packet is inserted in the input stream immediately following the BASIS card and any INSERT or DELETE cards.

DUMPS

If a serious error occurs during execution of a program, the job is 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 is identified by the heading

*** ABDUMP REQUESTED ***

If a SYSABEND DD statement is specified, the contents of the nucleus are 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 MVT.

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.

A dump cannot be requested in the COBOL language. The explanation of the system-generated completion codes and a complete description of the dumps are contained in the publication IBM System/360 Operating System: Programmer's Guide to Debugging.

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. If the file being processed is organized sequentially and no error-processing declarative or INVALID KEY option has been specified for the file, the job is terminated. If it is a QSAM file, the job will be terminated when there is no declarative or INVALID KEY option and the EROPT=ABE option in the DD statement has been specified.

Data in the record area of a file can be accessed only if the file is OPEN. If the file is opened as INPUT or I-O, the last I-O operation to the file must have been a READ. For example, if a READ is followed by WRITE, REWRITE, START, or CLOSE {REEL/UNIT}, the record area of the file can be accessed only after another READ is successfully executed.

In different cases, varying results occur; for example, abnormal termination, the program itself could be overlayed, other records on the file could be overlayed, etc.

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. For an item whose usage is COMPUTATIONAL, COMPUTATIONAL-1, or COMPUTATIONAL-2, either the alignment is incorrect, or the description of the item does not specify the proper alignment. Some examples are:

   a. A redefining entry contains one or more of the above items and the redefined entry is not properly aligned. Alignment will not be performed for items that cause the starting address of the redefining item to be changed.

   b. A record in the Linkage Section of a called program is described by an 01 entry and contains one or more of the above items, and the corresponding argument in the calling program is not properly aligned.

   c. A file, containing one or more of the above items, is blocked, but the required inter-record slack bytes were not inserted when the file was created. If the file is later read as an input file, the alignment may not be correct.

3. An input file 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.

4. 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.)

5. 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.

6. The SIZE ERROR option is not specified for a DIVIDE statement, and an attempt is made to divide by zero.

7. 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.

8. 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."

9. The data description in the Linkage Section of a called program does not correctly describe the data defined in the calling program.

10. Blanks read into data fields defined as numeric generate an invalid sign.

11. 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.

12. 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.

13. 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.

14. 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.

15. A subscript or index set at zero will address data outside the range of the table.

16. 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.

Other Errors

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 or WRITE associated with the unlocated file will result in an ABEND. A similar situation exists when a file

is closed WITH LOCK and an attempt is made to reopen it (see "Appendix G: Compiler Diagnostic Messages" for the format of the generated error message).

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. Under MVT, 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.

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.

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.

1. F13--Check register 2 of registers at the entry to ABEND. This address points to the DCB in conflict.

2. 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, the programmer should 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.

3. 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.

4. 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. Alignment for COMPUTATIONAL data is incorrect when record is blocked, and inter-record slack bytes were not inserted.

   g. Check for initialized subscript or index value.

5. 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.

6. 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.

7. 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.

8. 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, check register 14 of the registers at entry to ABEND. This address points to the file that has no DSCB.

9. 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.

10. 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.

11. 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.

12. 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.

13. 80A--Insufficient contiguous core storage for linkage to some phase of the compiler. The programmer should look to see if secondary data-set allocation has caused an extra DEB to be built at lower core 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.

14. 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.

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 49 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 MFT. For information about the abnormal termination dumps under MVT, see the publication IBM System/360 Operating System: Programmer's Guide to Debugging. Note that under the MVT option no indicative dumps are given.

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 core 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 core 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 input files, the register is not initialized until the first <u>successful</u> read; it is not

initialized when the file is opened. 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, and its output shown in Figure 55, illustrates in detail how an object code listing, cross reference list, and abnormal termination dump can be used together. Circled numbers in the example refer to the corresponding numbers in the text that follows. Note that all values are expressed in hexadecimal format.

```
00001          IDENTIFICATION DIVISION.
00002          PROGRAM-ID.   ABEND.
00003          REMARKS.
00004              THIS IS A PROGRAM TO ILLUSTRATE THE ABNORMAL TERMINATION.
00005          ENVIRONMENT DIVISION.
00006          CONFIGURATION SECTION.
00007          SOURCE-COMPUTER.  IBM-360-H50.
00008          OBJECT-COMPUTER.  IBM-360-H50.
00009          DATA DIVISION.
00010           WORKING-STORAGE SECTION.
00011              01  RECORDA.
00012                02  A  PICTURE S9(4) VALUE 1234.
00013                02  B REDEFINES A  PICTURE S9(7)  COMPUTATIONAL-3.   (9)
00014          PROCEDURE DIVISION.
00015              COMPUTE B = B + 1.   (8)
00016              STOP RUN.
```

```
15       COMPUTE    000272  (6)                      START    EQU   *
                    000272  F8 70 D 1C0 C 008                ZAP   1C0(8,13),008(1,12)   TS=01          LIT+0
                    000278  FA 43 D 1C3 6 000          (7)   AP    1C3(5,13),000(4,6)    TS=04          DNM=1-63
                    00027E  F8 33 6 000 D 1C4                ZAP   000(4,6),1C4(4,13)    DNM=1-63       TS=04+1
16       STOP       000284  58 F0 C 004                      L     15,004(0,12)          V(ILBOSTP1)
                    000288  07 FF                           BCR   15,15
                    00028A  05 F0                           BALR  15,0
                    00028C  91 10 D 048                     TM    048(13),X'10'         SWT+0
                    000290  47 E0 F 012                     BC    14,012(0,15)
                    000294  94 EF D 048                     NI    048(13),X'EF'         SWT+0
                    000298  58 F0 C 004                     L     15,004(0,12)          V(ILBOSTP1)
                    00029C  07 FF                           BCR   15,15
                    00029E  58 F0 D 1B0                     L     15,1B0(0,13)
                    0002A2  90 0E F 038                     STM   0,14,038(15)
                    0002A6  48 F0 D 05C                     LH    15,05C(0,13)
                    0002AA  58 D0 D 004                     L     13,004(0,13)
                    0002AE  98 0C D 014                     LM    0,12,014(13)
                    0002B2  58 EC D 00C                     L     14,00C(0,13)
                    0002B6  07 FE                           BCR   15,14
                    0002B8  50 D0 5 008             INIT2   ST    13,008(0,5)
                    0002BC  50 5C D 004                     ST    5,004(0,13)
                    0002C0  50 F0 D 054                     ST    14,054(0,13)
                    0002C4  58 F0 C 000                     L     15,000(0,12)          VIR=1
                    0002C8  05 EF                           BALR  14,15
                    0002CA  12 00                           LTR   0,0
                    0002CC  07 89                           BCR   8,9
                    0002CE  96 10 D 048                     OI    048(13),X'10'         SWT+0
                    0002D2  05 F0                   INIT3   BALR  15,0
                    0002D4  91 20 D 048                     TM    048(13),X'20'         SWT+0
                    0002D8  47 E0 F 016                     BC    14,016(0,15)
                    0002DC  98 2D B 040                     LM    2,13,040(11)
                    0002E0  58 00 B 038                     L     0,038(0,11)
                    0002E4  58 E0 D 054                     L     14,054(0,13)
                    0002E8  07 FE                           BCR   15,14
                    0002EA  96 20 D 048                     OI    048(13),X'20'         SWT+0
                    0002EE  41 60 0 004                     LA    6,004(0,0)
                    0002F2  41 80 D 1EC                     LA    8,1EC(0,13)           OVF=1
                    0002F6  41 70 D 1BF                     LA    7,1BF(0,13)           TS=01-1
                    0002FA  05 10                           BALR  1,0
                    0002FC  58 00 8 000                     L     0,000(0,8)
                    000300  1E 0B                           ALR   0,11
                    000302  50 00 8 000                     ST    0,000(0,8)
                    000306  87 86 1 000                     EXLR  8,6,000(1)
```

Figure 55. COBOL Program with Abnormal Termination Dump (Part 1 of 3)

```
                                CROSS REFERENCE TABLE

CONTROL SECTION                          ENTRY
  NAME     ORIGIN  LENGTH                 NAME   LOCATION     NAME   LOCATION     NAME   LOCATION     NAME   LOCATION
ABEND          00     314  (5)
ILBOSTP0*     318      35
                                         ILBOSTP1     32E

 LOCATION  REFERS TO SYMBOL  IN CONTROL SECTION          LOCATION  REFERS TO SYMBOL  IN CONTROL SECTION
     268            ILBOSTP0        ILBOSTP0                26C            ILBOSTP1        ILBOSTP0

ENTRY ADDRESS        00
TOTAL LENGTH        350   (4)

****GO        DOES NOT EXIST BUT HAS BEEN ADDED TO DATA SET
```

Figure 55. COBOL Program with Abnormal Termination Dump (Part 2 of 3)

```
  * * * A B D U M P   R E Q U E S T E D * * *

JOB JOB1              STEP STEP3           TIME 040354    DATE 69105

COMPLETION CODE       SYSTEM = 0C7  (1)

PROGRAM INTERRUPTION (DATA) AT LOCATION 010A98

INTERRUPT AT 010A9E  (2)

PSW AT ENTRY TO ABEND  800C7000 00000000

TCB  000180  RB  0007FBE0   PIE    0C000000   DEB 0007FB6C   TICT 0007FF48   CMP 800C7000   TRN 00000000
             MSS 00005918   PK/FLG 10910408   FLG 00000000   LLS  00000000   JLB 0007FE90   JSE 00000000
             FSA 0407FFB4   TCB    0CC00C00   TME 0000592C

ACTIVE RBS                                                                              (3)

PRB  010800  NM GO          SZ/STAB 006E00C0   USE/EP 00010820   PSW FF15000D E0010A9E   Q 000000   WT/LNK 00000180

SVRB 07FE00  NM SVC-401C    SZ/STAB 0012F072   USE/EP 000034A8   PSW FF040033 4000366C   Q E003E0   WT/LNK 00010800
             RG 0-7   000108A0    50010B1C    0000006C    00000181    000068F0    0007FFB4    000108A0    00010A6F
             RG 8-15  0C01CA7C    00010AF2    00010820    00010820    00010A88    000108B0    00010A92    50010AF4

SVRB 07FBE0  NM SVC-105A    SZ/STAB 000CF072   USE/EP 000034A8   PSW FF04000F 8000C4FC   Q E003E0   WT/LNK 0007FE00
             RG 0-7   00010B70    00010BC8    000036C8    400034AA    00000000    00000000    00010B70    8000365A
             RG 8-15  00005918    00010C3E    0C080000    00000180    00000180    00010ED8    500036F0    20010A98

P/P STORAGE BOUNDARIES 00010800 TO 0C080000

FREE AREAS     SIZE

  010E70      0006ECA0
  07FBB0      00000030

SAVE AREA TRACE

GO        WAS ENTERED

SA   07FFB4  WD1 00000181   HSA 00000000   LSA 000108B0   RET 000053E8   EPA 70010820   P0  00000030
             R1  0007FF18   R2  0000006C   R3  00000181   R4  000068F0   R5  0007FF5C   R6  00005398
             R7  00000080   R8  00000078   R9  C00260E2   R10 0007FF08   R11 00005398   R12 00000180

SA   0108B0  WD1 00000000   HSA 0007FFB4   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

PROCEEDING BACK VIA REG 13
```

Figure 55. COBOL Program with Abnormal Termination Dump (Part 3 of 3)

1. The completion code, (1), in the dump indicates the condition causing the abnormal termination. If the system part of the code is nonzero, the explanation is found in the publication IBM System/360 Operating System: Programmer's Guide to Debugging. In this example, invalid data is the reason for termination.

2. The PROGRAM INTERRUPTION (DATA) AT LOCATION 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). In the example, the address is 10A9E.

3. 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 USE/EP entry, (3), of the first ACTIVE RBS (Request Blocks) specification. The last six digits of this entry are the address of the entry point (INIT1) in the COBOL program. In this case, the address is 10820 in hexadecimal format.

   The total length of the load module is indicated in the TOTAL LENGTH entry, (4), in the linkage editor output (350, in the example). The highest location in the load module is:

   I0820 + 350 = 10B70

   Thus, the range is from 10820 to 10B70. Since the 10A9E address falls within this range, the instruction initiating the dump must be within the load module.

4. To determine the relative location within the load module of the instruction indicated in the INTERRUPTION entry, subtract the load address from the address of the instruction. In the example, this becomes:

   10A9E - 10820 = 27E

5. To determine whether or not the instruction occurred in the object module generated for the program, compare its relative location (27E) with the total length, (4), 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 program, (5), and the total length 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, 27E is less than the program size (314), so the instruction occurred in the main program.

6. To determine whether or not the instruction was 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 (27E > 272), 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, (8), is located and 'B' is seen to be the data-name that caused the trouble. Data item B is defined at (9), as a COMPUTATIONAL-3 or packed 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 zoned decimal format since there is no USAGE clause specified. The configuration of A is invalid for B and results in an interrupt.

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.

Since the sample problem program shown in Figure 56 was interrupted because of a data exception, the programmer should locate the contents of field B at the time of the interrupt. The circled numbers in the explanation that follows refer to the corresponding numbers in the sample program.

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 050. The sum of the value of base locator 3 and the hexadecimal displacement value 50 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) . Register 6 contains the address 000141F0.

4. The location of data-name B, (4) , can now be determined by adding the contents of register 6 and the hexadecimal displacement value 50. The result, 14240, is the address of the leftmost byte of the 4-byte field B.

Note: Field B contains F1F2F3C4. This is external decimal representation and does not correspond to the USAGE COMPUTATIONAL-3 defined in the source listing.

Some program errors may destroy the contents of the general registers referred to above. In such cases, an alternate method of locating data-names is useful.

The location assigned to a given data-name may also be found by using the BL CELLS pointer in the TGT Memory Map, (5) The location of the BL cells is found by adding 003E4 (from the TGT table) to the load point address, 14020, of the object module, (6). In this example, the BL cells begin at location 14404:

003E4 + 14020 = 14404

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 value 141F0. This is the same value as that contained in register 6.

```
00001    000010 IDENTIFICATION DIVISION.
00002    000020 PROGRAM-ID. TESTRUN.
00003    000030    AUTHOR. PROGRAMMER NAME.
00004    000040    INSTALLATION. NEW YORK PROGRAMMING CFNTER.
00005    000050    DATE-WRITTEN. SEPTEMBER 10, 1968.
00006    000060 DATE-COMPILED. MAY 16,1970
00007    000070    REMARKS. THIS PROGRAM HAS BEEN WRITTEN AS A SAMPLE PROGRAM FOR
00008    000080      COBOL USERS. IT CREATES AN OUTPUT FILE AND READS IT BACK AS
00009    000090      INPUT.
00010    000100
00011    000110 ENVIRONMENT DIVISION.
00012    000120 CONFIGURATION SECTION.
00013    000130 SOURCE-COMPUTER. IBM-360-H50.
00014    000140 OBJECT-COMPUTER. IBM-360-H50.
00015    000150 INPUT-OUTPUT SECTION.
00016    000160 FILE-CONTROL.
00017    000170      SELECT FILE-1 ASSIGN TO UT-S-PRTOUT.
00018 **000170      SELECT FILE-2 ASSIGN TO UT-S-PRTOUT.
00019    000190
00020    000200 DATA DIVISION.
00021    000210 FILE SECTION.
00022    000220 FD  FILE-1
00023    000230      LABEL RECORDS ARE STANDARD
00024    000240      BLOCK CONTAINS 5 RECORDS
00025    000250      RECORDING MODE IS F
00026    000255      RECORD CONTAINS 20 CHARACTERS
00027    000260      DATA RECORD IS RECORD-1.
00028    000270 01  RECORD-1.
00029    000280      05 FIELD-A PIC X(20).
00030    000290 FD  FILE-2
00031    000300      LABEL RECORDS ARE STANDARD
00032    000310     BLOCK CONTAINS 5 RECORDS
00033    000320     RECORD CONTAINS 20 CHARACTERS
00034    000330     RECORDING MODE IS F
00035    000340     DATA RECORD IS RECORD-2.
00036    000350 01  RECORD-2.
00037    000360     05 FIELD-A PIC X(20).
00038    000370 WORKING-STORAGE SECTION.
00039    000380 01  FILLER.
00040    000390     02 COUNT PIC S99 COMP SYNC.
00041    000400     02 ALPHABET PIC X(26) VALUE IS "ABCDEFGHIJKLMNOPQRSTUVWXYZ".
00042    000410     02 ALPHA REDEFINES ALPHABET PIC X OCCURS 26 TIMES.
00043    000420     02 NUMBR PIC S99 COMP SYNC.
00044    000430     02 DEPENDENTS PIC X(26) VALUE "01234012340123401234012340".
00045    000440     02 DEPEND REDEFINES DEPENDENTS PIC X OCCURS 26 TIMES.
00046    000450 01  WORK-RECORD.
00047    000460     05 NAME-FIELD PIC X.
00048    000470     05 FILLER PIC X.
00049    000480     05 RECORD-NO PIC 9999.
00050    000490     05 FILLER PIC X VALUE IS SPACE.
00051    000500     05 LOCATION PIC AAA VALUE IS "NYC".
00052    000510     05 FILLER PIC X VALUE IS SPACE.
00053    000520     05 NO-OF-DEPENDENTS PIC XX.
00054    000530     05 FILLER PIC X(7) VALUE IS SPACES.
00055    000534 01  RECORDA.
00056    000535     02 A PICTURE S9(4) VALUE 1234.
00057    000536     02 B REDEFINES A PICTURE S9(7) COMPUTATIONAL-3.
```

Figure 56. Sample Program (Part 1 of 5)

```
00058   000540
00059   000550 PROCEDURE DIVISION.
00060   000560 BEGIN. READY TRACE.
00061   000570     NOTE THAT THE FOLLOWING OPENS THE OUTPUT FILE TO BE CREATED
00062   000580     AND INITIALIZES COUNTERS.
00063   000590 STEP-1. OPEN OUTPUT FILE-1. MOVE ZERO TO COUNT, NUMBR.
00064   000600     NOTE THAT THE FOLLOWING CREATES INTERNALLY THE RECORDS TO BE
00065   000610     CONTAINED IN THE FILE, WRITES THEM ON TAPE, AND DISPLAYS
00066   000620     THEM ON THE CONSOLE.
00067   000630 STEP-2. ADD 1 TO COUNT, NUMBR. MOVE ALPHA (COUNT) TO
00068   000640     NAME-FIELD.
00069   000645         COMPUTE B = B + 1.
00070   000650     MOVE DEPEND (COUNT) TO NO-OF-DEPENDENTS.
00071   000660     MOVE NUMBR TO RECORD-NO.
00072   000670 STEP-3. DISPLAY WORK-RECORD UPON CONSOLE. WRITE RECORD-1 FROM
00073   000680     WORK-RECORD.
00074   000690 STEP-4. PERFORM STEP-2 THRU STEP-3 UNTIL COUNT IS EQUAL TO 26.
00075   000700     NOTE THAT THE FOLLOWING CLOSES THE OUTPUT FILE AND REOPENS
00076   000710     IT AS INPUT.
00077   000720 STEP-5. CLOSE FILE-1. OPEN INPUT FILE-2.
00078   000730     NOTE THAT THE FOLLOWING READS BACK THE FILE AND SINGLES
00079   000740     OUT EMPLOYEES WITH NO DEPENDENTS.
00080   000750 STEP-6. READ FILE-2 RECORD INTO WORK-RECORD AT END GO TO STEP-8.
00081   000760 STEP-7. IF NO-OF-DEPENDENTS IS EQUAL TO "0" MOVE "Z" TO
00082   000770     NO-OF-DEPENDENTS. EXHIBIT NAMED WORK-RECORD. GO TO STEP-6.
00083   000780 STEP-8. CLOSE FILE-2.
00084   000790     STOP RUN.
```

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=1-167	01	RECORD-1	BL=1	000	DNM=1-167	DS 0CL20	GROUP				
DNM=1-188	02	FIELD-A	BL=1	000	DNM=1-188	DS 20C	DISP				
DNM=1-205	FD	FILE-2	DCB=02		DNM=1-205		QSAM				F
DNM=1-224	01	RECORD-2	BL=2	000	DNM=1-224	DS 0CL20	GROUP				
DNM=1-245	02	FIELD-A	BL=2	000	DNM=1-245	DS 20C	DISP				
DNM=1-265	01	FILLER	BL=3	000	DNM=1-265	DS 0CL56	GROUP				
DNM=1-284	02	COUNT	BL=3	000	DNM=1-284	DS 1H	COMP				
DNM=1-299	02	ALPHABET	BL=3	002	DNM=1-299	DS 26C	DISP				
DNM=1-317	02	ALPHA	BL=3	002	DNM=1-317	DS 1C	DISP	R	O		
DNM=1-335	02	NUMBR	BL=3	01C	DNM=1-335	DS 1H	COMP				
DNM=1-350	02	DEPENDENTS	BL=3	01E	DNM=1-350	DS 26C	DISP				
DNM=1-370	02	DEPEND	BL=3	01E	DNM=1-370	DS 1C	DISP	R	O		
DNM=1-386	01	WORK-RECORD	BL=3	038	DNM=1-386	DS 0CL20	GROUP				
DNM=1-410	02	NAME-FIELD	BL=3	038	DNM=1-410	DS 1C	DISP				
DNM=1-430	02	FILLER	BL=3	039	DNM=1-430	DS 1C	DISP				
DNM=1-449	02	RECORD-NO	BL=3	03A	DNM=1-449	DS 4C	DISP-NM				
DNM=1-468	02	FILLER	BL=3	03E	DNM=1-468	DS 1C	DISP				
DNM=1-487	02	LOCATION	BL=3	03F	DNM=1-487	DS 3C	DISP				
DNM=2-000	02	FILLER	BL=3	042	DNM=2-000	DS 1C	DISP				
DNM=2-019	02	NO-OF-DEPENDENTS	BL=3	043	DNM=2-019	DS 2C	DISP				
DNM=2-045	02	FILLER	BL=3	045	DNM=2-045	DS 7C	DISP				
DNM=2-064	01	RECORDA	BL=3	050	DNM=2-064	DS 0CL4	GROUP				
DNM=2-084	02	A	BL=3	050	DNM=2-084	DS 4C	DISP-NM				
DNM=2-095	02	B (1)	BL=3	050	DNM=2-095	DS 4P	COMP-3	R			

Figure 56. Sample Program (Part 2 of 5)

```
                    MEMORY MAP

                TGT                         00228

              SAVE AREA                     00228
              SWITCH                        00270
              TALLY                         00274
              SORT SAVE                     00278
              ENTRY-SAVE                    0027C
              SORT CORE SIZE                00280
              RET CODE                      00284
              SORT RET                      00286
              WORKING CELLS                 00288
              SORT FILE SIZE                003B8
              SORT MODE SIZE                003BC
              PGT-VN TBL                    003C0
              TGT-VN TBL                    003C4
              VCONPTR                       003C8
              LENGTH OF VN TBL              003CC
              LABEL RET                     003CE
              UNUSED                        003CF
              INIT1 ADCON                   003D8
              UNUSED                        003DC
              OVERFLOW CELLS                003E4
(5)-->        BL CELLS                      003E4
              DECBADR CELLS                 003F0
              TEMP STORAGE                  003F0
              TEMP STORAGE-2                003F8
              TEMP STORAGE-3                00428
              TEMP STORAGE-4                00428
              BLL CELLS                     00428
              VLC CELLS                     00430
              SBL CELLS                     00430
              INDEX CELLS                   00430
              SUBADR CELLS                  00430
              ONCTL CELLS                   00430
              PFMCTL CELLS                  00430
              PFMSAV CELLS                  00430
              VN CELLS                      00434
              SAVE AREA =2                  00438
              SAVE AREA =3                  00438
              XSASW CELLS                   00440
              XSA CELLS                     00440
              PARAM CELLS                   00440
              RPTSAV AREA                   00444
              CHECKPT CTR                   00444
              VCON TBL                      00448

LITERAL POOL (HEX)

004A0 (LIT+0)     004805EF  00228000  C9D2C6F9  F9F9C940  E4D5E2E4  C3C3C5E2
004B8 (LIT+24)    E2C6E4D3  40D6D7C5  D540C6D6  D9400000  00000000  00000200
004D0 (LIT+48)    40200001  00000001  1C00001A  4800F0E9  C0000000

     DISPLAY LITERALS (BCD)

004E4 (LTL+68)   'WORK-RECORD'
```

Figure 56. Sample Program (Part 3 of 5)

```
                PGT                     00450

            OVERFLOW CELLS              00450
            VIRTUAL CELLS               00450
            PROCEDURE NAME CELLS        0045C
            GENERATED NAME CELLS        00470
            DCB ADDRESS CELLS           00490
            VNI CELLS                   00498
            LITERALS                    004A0
            DISPLAY LITERALS            004E4

 REGISTER ASSIGNMENT

  REG 6    BL =3 <-(2)
  REG 7    BL =1
  REG 8    BL =2

60      *BEGIN
                0004F0                          START   EQU   *
                0004F0  58 F0 C 004                     L     15,004(0,12)          V(ILBODSP0)
                0004F4  05 1F                           BALR  1,15
                0004F6  000140                          DC    X'000140'
                0004F9  04F6F0404040                    DC    X'04F6F0404040'
60       READY  000500  96 40 D 048                     OI    048(13),X'40'         SWT+0
63      *STEP-1
                000504  58 F0 C 004                     L     15,004(0,12)          V(ILBODSP0)
                000508  05 1F                           BALR  1,15
                00050A  000140                          DC    X'000140'
                00050D  04F6F3404040                    DC    X'04F6F3404040'
                .
                .
                .

 * ABDUMP REQUESTED *

JOB BUFFERS          STEP GO              TIME 064255   DATE 70136

COMPLETION CODE      SYSTEM = 0C7

PROGRAM INTERRUPTION (DATA) AT LOCATION 0145D2

INTERRUPT AT 0145D8

PSW AT ENTRY TO ABEND  FF25000D E00145D8

TCB  0053D8    RB   00030068    PIE    00000000    DEB  0002FFEC    TIOT 00030520    CMP   800C7000    TRN  00000000
               MSS  00005460    PK/FLG 20912408    FLG  000003F8    LLS  00000000    JLB   00030778    JSE  00000000
               FSA  180307B0    TCB    00000000    TME  00005488    PIB  E0009FC0    NSTAE 00000000    TCT  00004600
               USER 00000000

ACTIVE RBS                                                          <-(6)

PRB    014000  NM RUN         SZ/STAB 01F100C0   USE/EP 00014020   PSW FF25000D E00145D8   Q 000000   WT/LNK 000053D8

SVRB   030490  NM SVC-401C    SZ/STAB 0012D172   USE/EP 000041A8   PSW FF040033 40004370   Q E803E8   WT/LNK 00014000
               RG 0-7   00000030    000145A2    00000001    00000001    000141F2    50014870    000141F0    00030130
               RG 8-15  00014020    00014844    00014020    00014020    00014470    00014248    7001457E    000148E0

SVRB   030068  NM SVC-105A    SZ/STAB 000CD172   USE/EP 000041A8   PSW FF040235 8000D53C   Q C803C8   WT/LNK 00030490
               RG 0-7   00014F88    00014FE0    000012C0    400041AA    00000000    00000000    00014F88    8000435E
               RG 8-15  0000545E    00015056    00030800    000053D8    000053D8    00014FF0    600043FA    200145D2

P/P STORAGE BOUNDARIES 00014000 TO 00030800
```

Figure 56. Sample Program (Part 4 of 5)

```
REGS AT ENTRY TO ABEND

FL.PT.REGS 0-6       00.000000 00000000      00.000000 00000000      00.000000 00000000 (3) 00.000000 00000000

REGS 0-7          00000030   000145A2   00000001   00000001          000141F2   50014870   000141F0   00030130
REGS 8-15         00014020   00014844   00014020   00014020          00014470   00014248   7001457E   000148B0
                     .
                     .
                     .

P/P STORAGE

014000   D9E4D540 40404040 01F100C0 00014020   FF25000D E00145D8 00000000 000053D8   *RUN     .1.... ........Q.......Q*
014020   070090EC D00C185D 05F0989F F01207FF   96021034 07FE41F0 000107FE 00014844   *.........0..0..........0........*
014040   00014020 00014020 00014470 00014248   00014510 00014826 00030005 00060007   *.. ... ........................*
014060   00080009 000A000B 000C000E 000F0010   0012001C 001D001E 0020003C 007A00FF   *.............................*
014080   0FFFFFFF FFFEFFFD FFFCFFFB D2000002   03004940 FFFFFFFF E3C5E2E3 D9E4D540   *.............K...... ....TESTRUN *
0140A0   00000000 00000000 00000000 00000000   00000000 00014036 00000001 00000001   *.......................6 .........*
0140C0   00000000 80000000 00000000 00000000   00000000 00000000 00818300 02030128   *.............................*
0140E0   00004000 00000001 46000001 900140A0   00400048 00030224 9200CCC0 0000CF58   *.. ........... .. ..............*
014100   06000001 00090064 28012828 420300C8   00030194 00030144 00000014 00000001   *.............H.................*
014120   00000000 0000DE68 05EF0000 00000000   00000000 00000000 00000000 00000000   *.............................*
014140   00E3E5F1 D7E3E5F2 00000000 00000000   00000000 00000000 00000000 00014036   *.TV1PTV2.....................6*
014160   00000001 00000001 00000000 80000000   00000000 00000000 00000000 00000000   *.............................*
014180   00000000 00000001 00004000 00000001   42000001 90014148 D7D9E3D6 E4E34040   *.......... ..........PRTOUT  *
0141A0   02000000 00000000 00000001 00000064   00000000 00000000 00000000 00000000   *.............................*
0141C0   00000014 00000000 00000000 00000000   00000000 00000000 00000000 00000000   *.............................*
0141E0   00000000 00000000 00000000 0DC9D3C2   0001C1C2 C3C4C5C6 C7C8C9D1 D2D3D4D5   *.............ILB..ABCDEFGHIJKLMN*
014200   D6D7D8D9 E2E3E4E5 E6E7E8E9 0001F0F1   F2F3F4F0 F1F2F3F4 F0F1F2F3 F4F0F1F2   *OPQRSTUVWXYZ..0123401234012340 12*
(4)014220   F3F4F0F1 F2F3F4F0 C1E3C9D5 C74040D5   E8C3404B C9404040 40404040 4040D9C5   *34012340ATING  NYC .I          RE*
014240   F1F2F3C4 40E3E8D7 007EC66B 000307B0   E3C87E4D 340040E2 D6D9E340 50014B3A   *123D TYP..F.....TH.... SORT ....*
014260   9200CCC0 00000030 8F0306E0 000140C8   00030800 0001459C 00000004 000303A1   *.............. H.............H.*
014280   00000079 900149AC 6000CD10 0000CF58   7000004B 00000000 00000000 00014510   *.............................*
0142A0   00000000 00000000 004805EF 48480070   30464811 13480A0A 48501070 30404858   *.............................. ..*
0142C0   2070104C 484B2070 10524850 2070104C   48581070 20144896 01702017 48484070   *.............................. .*
0142E0   1004484C 40701006 48410070 40084841   10701000 480A0A48 48007020 46485810   *.... ....... .....................*
014300   70204048 0A0A4842 30705001 48883070   00084842 30705000 48920070 50024892   *.. ...............................*
014320   00705003 48423070 10014888 30700008   48423070 10004890 EC70D00C 48185D48   *.............................*
014340   05F04845 0001459A 000140C8 01000048   7001457E 000148B0 00000030 000145A2   *.0........ H.....................*
014360   000140C8 00030800 D204A45C 50014870   000141F0 00030130 00014020 00014844   *.. H....K..........0...... .....*
014380   00014020 00014020 00014470 000145A2   000148E0 00000030 8F0306E0 000140C8   *.. ... ........................ H*
0143A0   00014807 0001459A 48077148 47C07030   08481424 48078148 07414847 8070300A   *.............................*
0143C0   48078148 4740703Q 0A485850 70300C48   58105C00 00485010 00000000 00000000   *..... ..............................*
0143E0   0C489280 70300448 40107030 06485A30   70104C48 4B307010 00014020 70104C48   *........ .................. .....*
014400   50307010 00030130 00014020 000141F0   1C000280 00000348 4110641C 0001481E   *.......... ....0.................*
                                    (7)
```

Figure 56. Sample Program (Part 5 of 5)

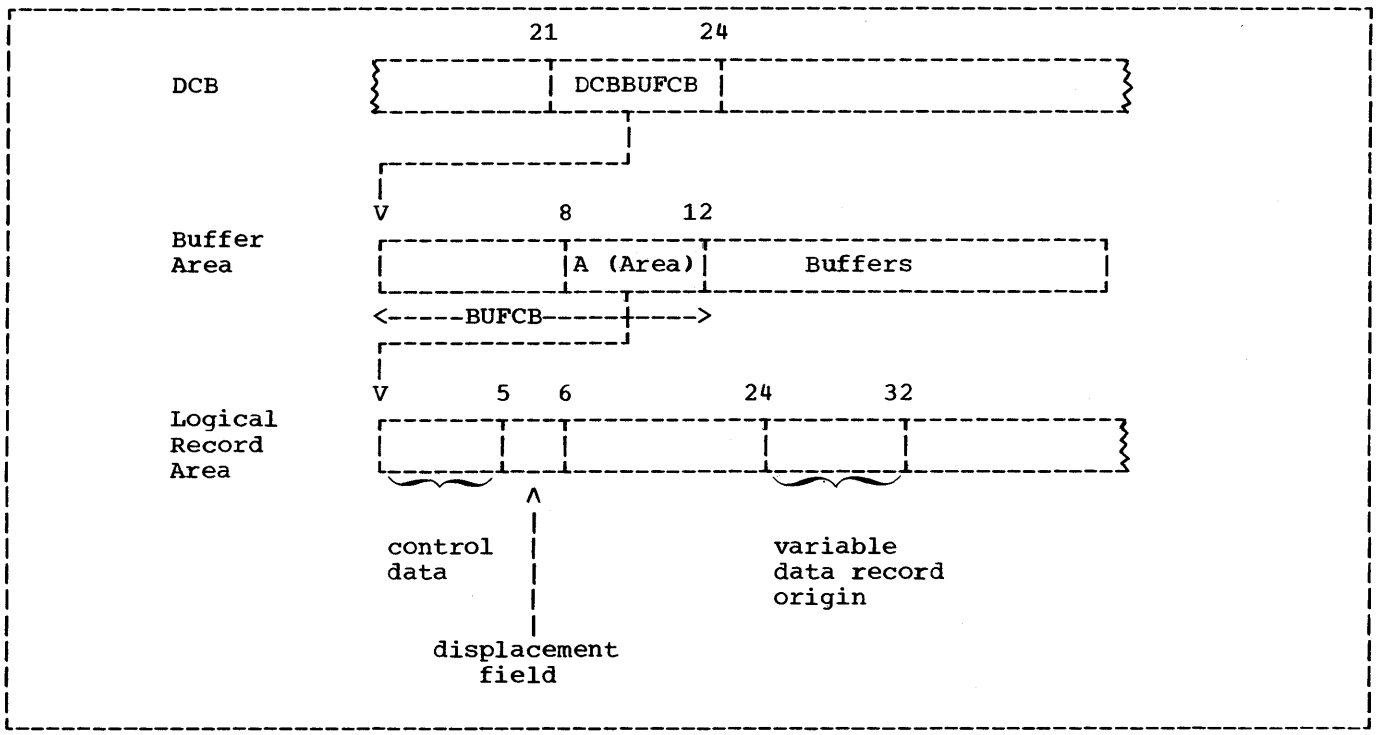


Figure 57. Locating the QSAM Logical Record Area

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 57 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.

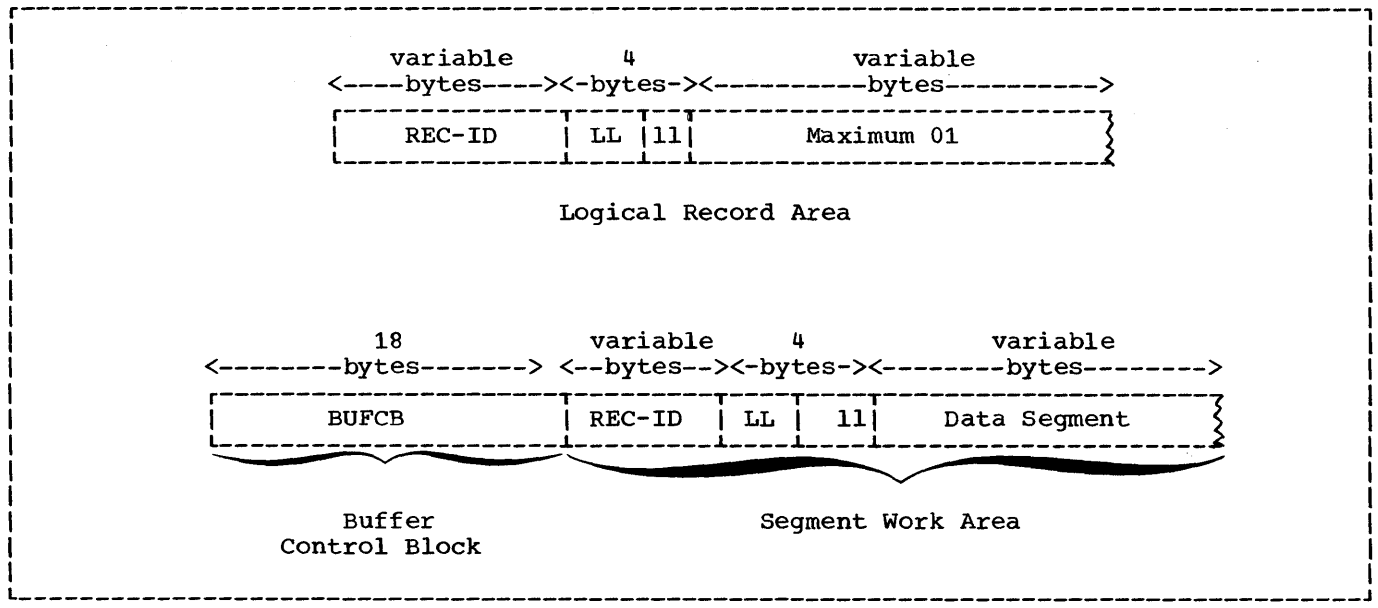


Figure 58. Logical Record Area and Segment Work Area for BDAM and BSAM Spanned Records

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 58 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 58.

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 58 (see Figure 56 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.

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 setsfor which the programmer has assigned the name.

SCRATCHING 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.

Note: The information in this section about scratching data sets applies only when running under MFT. Under the MVT option, direct-access volume data sets are scratched automatically. For use of the system utilities under MVT, see the publication IBM System/360 Operating System: Utilities.

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

name.jobname

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 IBM System/360 Operating System: Utilities. The following example illustrates the job control statements that might be used to scratch temporary data sets:

```
//SCR        JOB      ,SCRATCH,MSGLEVEL=1
//STEP1      EXEC     PGM=IEHPROGM
//SYSPRINT   DD       SYSOUT=A
//DD1        DD       UNIT=2311,DISP=OLD
//DD2        DD       UNIT=2311,DISP=OLD,   X
//                    VOLUME=SER=222222
                .
                .
                .
//SYSIN      DD       *
           SCRATCH   DSNAME=GOJOB.TEMP,    X
                     VOL=2311=222222,PURGE
           SCRATCH   VTOC,VOL=2311=222222,X
                     SYS,PURGE
                .
                .
                .
/*
```

In this example, the SYSPRINT DD statement specifies the output data set for the listing and the DD1 DD statement specifies the system residence volume. The other DD statements specify the volume serial number of the volumes that can be mounted on which the data sets have been written. These DD statements are needed to allocate the required devices. The first SCRATCH statement eliminates a data set for which DSNAME=&&TEMP had been specified on the DD statement, and the second SCRATCH statement eliminates all data sets on the volume for which no DSNAME parameter had been specified.

Note that the possibility of duplicate name definition also applies to cataloged procedures in which temporary data sets are used.

For those procedures that are executed often, the programmer may wish to include, at the beginning of his job, a procedure to scratch all temporary data sets.

Some techniques for increasing the efficiency of a COBOL program are described in this chapter. It is divided into six parts. The first four parts deal in general with coding a COBOL program. The fifth is concerned with the Report Writer feature and the last with Table Handling.

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.

ENVIRONMENT DIVISION

APPLY WRITE-ONLY Clause

To make optimum use of buffer space allocated when creating a standard 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

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 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 core 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.

OVERALL CONSIDERATIONS

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.

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

A simple method of locating the Working-Storage Section of a program in object-time dumps is to include the two following statements as the first and last Working-Storage statements, respectively, in the program.

```
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, 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         PICTURE X(23).
        10  NUMBERX      PICTURE S9(5) COMP.
    05  DATE-RECORD      PICTURE 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 redefining a subscript, for if the value of the redefining data item is changed in the Procedure Division, no new calculation for the subscript is performed.

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 prevail:

   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 prevail:

   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.

Table 23. Data Format Conversion

Usage	Bytes Required	Boundary Alignment Required	Typical Use	Converted for Arithmetic Operations	Special Characteristics
DISPLAY (external decimal)	1 per digit (except for V)	No	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)	No	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	No	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	halfword fullword fullword	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). Extra instructions are generated for binary computations if the SYMCHRONIZED clause is not specified. Fields of over 9 digits require more handling.
COMP-1 (internal floating point)	4 (short-precision)	fullword	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. Extra instructions are generated if the SYNCHRONIZED clause is not specified.
COMP-2 (internal floating point)	8 (long-precision)	double-word	Fractional exponentiation when more precision is required	No	Same as COMPUTATIONAL-1

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.

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 Table 23).

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.

To Preform 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.

To Perform Arithmetic Operations: 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 arithmetic 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.

SYNCHRONIZED Clause

DATA FORMATS: As shown in Table 23, COMPUTATIONAL, COMPUTATIONAL-1, and COMPUTATIONAL-2 items have specific boundary alignment requirements. To ensure correct alignment, either the programmer or the compiler may have to add slack bytes, or the compiler must generate instructions to move the item to a correctly aligned work area when reference is made to the item.

The SYNCHRONIZED clause may be used at the elementary level to specify the automatic alignment of elementary items on their proper boundaries or at level-01 to synchronize all elementary items within the group. For COMPUTATIONAL items, if the PICTURE is in the range of S9 through S9(4), the item is aligned on a halfword boundary. If the PICTURE is in the range of S9(5) through S9(18), the item is aligned on a fullword boundary. For COMPUTATIONAL-1 items, the item is aligned on a fullword boundary. For COMPUTATIONAL-2 items, the item is aligned on a double-word boundary. The SYNCHRONIZED clause and slack bytes are fully discussed in the publication IBM System/360 Operating System: Full American National Standard COBOL.

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 FIELDS: COMPUTATIONAL fields can be aligned on either a halfword or fullword boundary. If an operation involves COMPUTATIONAL fields of different lengths, the halfword field is automatically expanded to a fullword field. Therefore, mixed halfword and fullword fields require no additional operations.

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/360 Principles of Operation, Order No. A22-6821.

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:

```
| F1 | F2 | F3 | F4 |
                 Byte

| F1 | F2 | F3 | D4 |
                 Byte
```

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. Table 24 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:

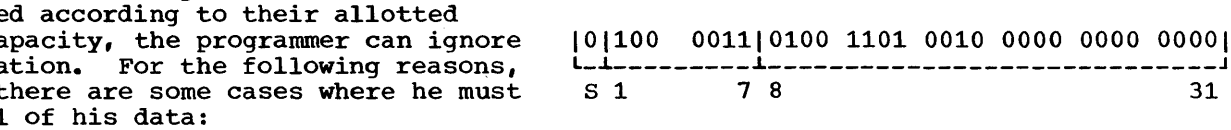
1. When the ON SIZE ERROR option is used, the size test is made on the basis of the maximum value allowed by the picture of the result field. If a size error condition exists, the value of the result field is not altered and control is given to the imperative statements specified by the error option.

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.

Table 25 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:



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/360 Principles of Operation.

Table 24. Relationship of PICTURE to Storage Allocation

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

Table 25. Treatment of Varying Values in a Data Item of PICTURE S9

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

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.

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. In the publication IBM System/360 Operating System: Full American National Standard COBOL, the section "Appendix A: Intermediate Results" 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 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.

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

ACCEPT Statement

If end-of-file is reached on SYSIN, no further I-O requests will be made for the file, and if the number of records existing is not enough to fill it, that portion of the receiving field exceeding the current capacity of the input records will remain as it was before the ACCEPT statement.

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.

Performing an IF operation for an item greater than 256 bytes in length requires the generation of more instructions than are required for that of an IF operation for an item of 256 bytes or less.

The ALL figurative constant can be used in an IF statement to generate a CLI instruction rather than a CLC instruction if the field being tested is a one-byte field. For example, the statement

```
IF FIELD-1 IS EQUAL TO ALL '1' GO TO PARA-1
```

will generate a CLI instruction. The same statement without the ALL figurative constant will generate a CLC instruction.

MOVE Statement

Performing a MOVE operation for an item greater than 256 bytes in length requires the generation of more instructions than are required for that of a MOVE operation for an item of 256 bytes or less.

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.

The compiler assumes that the data being moved conforms to PICTURE and USAGE specifications. If it does not, dissimilar results will occasionally occur because of the different code generated for various sending and receiving fields. This fact is most apparent when the sending field is COMPUTATIONAL, the value in the item exceeds the number of digits specified in the PICTURE clause, and the option NOTRUNC is in effect.

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 two 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.

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.

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 by keeping 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 DAY
```

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 DAY.
    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 DAY.
    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 59 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 59. 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 the desired code is found. The record should be printed starting from the third character, as illustrated in Figure 60.

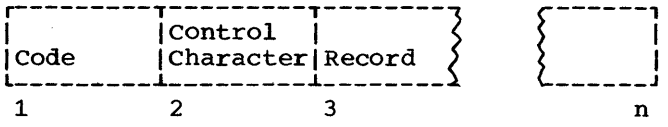


Figure 60. Format of a Report Record When the CODE Clause Is Specified

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, DAY
        .
        .
        .
01  TYPE CONTROL FOOTING DAY
    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 DAY)

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 (PH, PF, 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 respective RD. This is reflected when either PMAP or CLIST is in effect.

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 referenced 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 core storage.

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.

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

---

[1]For a discussion of the use of the OCCURS DEPENDING ON clause in a sort program, see "Sorting Variable-Length Records."

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 100
        TIMES DEPENDING ON A.
    05  GROUPB.

        (Subordinate data items.)

            (End of record.)
```

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.

If the value of an object of OCCURS DEPENDING ON is changed because of a change in the value of an item that either redefines the object, is redefined by the object, or is subordinate to such an item, the group sizes and the locations of items affected by the OCCURS DEPENDING ON clause are unpredictable.

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.

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).
```

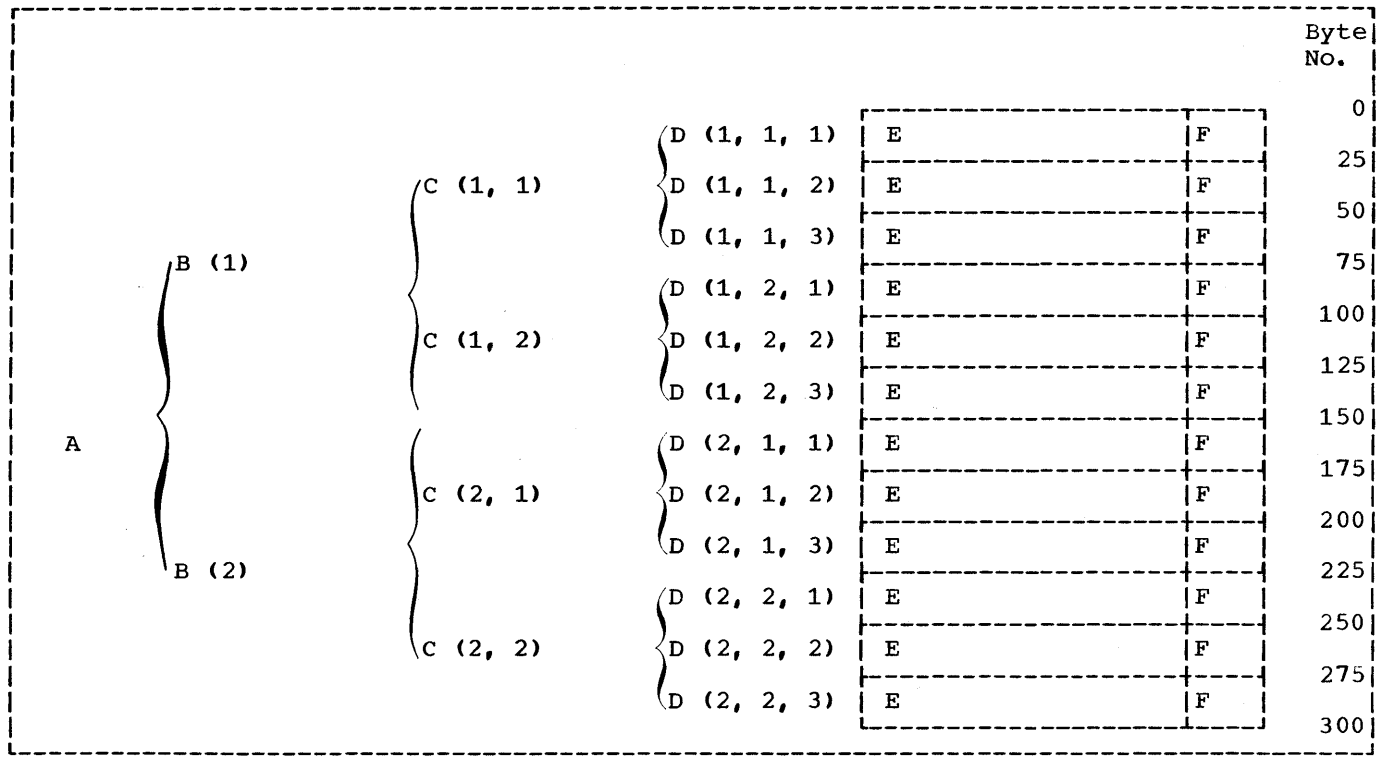


Figure 61. Storage Layout for Table Reference Example

Figure 61 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 <u>incorrect</u>:

```
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 <u>not</u> the address of D (2,2,3).

The following steps will find the <u>correct</u> 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 Table 26.

Table 26. Rules for the SET Statement

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

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, is used for a serial search. Format 2, SEARCH ALL, is used for a binary search.

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 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 or object 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  TABLE.
    05  ENTRY-IN-TABLE 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-TABLE 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 TABLE 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 TABLE, 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.

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. Control is returned in the same order of calling; that is, a called program returns control to its own calling program, not to an earlier calling program. 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.

All called and calling programs to be executed as a single job step must be linkage edited together; they must all be included in the same load module.

This chapter describes the accepted linkage conventions for calling and called programs in both COBOL and assembler language and discusses how such programs are linkage edited. 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.

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 64.1 for an example of the linkage statements required in a typical calling/called situation.

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
  [USING identifier-list].
```

Literal-1 is the name of the entry point in the called program to which control is to be transferred. The first eight characters of literal-1 are used to make the correspondence between the calling and the called programs. 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 may also be a file-name or a procedure-name. If the identifier 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. 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 is a procedure-name, the value passed is the beginning address of the procedure. If no identifiers are passed, the USING clause is omitted.

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.

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. When a group item with a level number other than 01 is specified as an identifier of the calling program, proper word-boundary alignment is required if subordinate items are described as COMPUTATIONAL, COMPUTATIONAL-1, or COMPUTATIONAL-2. If the identifier of the calling program corresponds to a level-01 identifier of the called program, doubleword alignment is required.

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. Table 27 shows the conventions for use of general registers as 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

Table 27. 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.

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 62 shows a portion of an assembler-language program that illustrates the conventions used in a calling program.

A GOBACK statement or a STOP RUN statement issued within a COBOL program will (always for STOP RUN, but only in a main program for GOBACK) reference the COBOL library subroutine ILBOSTP0. Furthermore, the STOP RUN statement will end the run unit, which is assumed to begin with the highest-level COBOL program called. To circumvent this assumption, a higher-level assembler language program must call the COBOL library subroutine ILBOSTP0 before making any calls to other COBOL programs. This should be done as soon as possible after entry to the assembler-language program, as part of the program's initialization procedure.

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 63). 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; (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 (branching to the address in register 14 to complete the return); and places a return code 08 or zero, in register 15.

Figure 64 shows a portion of an assembler-language program that illustrates the conventions used in called programs. Figure 65 shows the JCL suggested for compiling, link-editing, and executing a calling assembler-language program and a called COBOL program.

```
         .
         .
         .
      (Calling Sequence)
         LA     1,ARGLST      Loads into register 1 the address of the identifier list
                                to be passed.
         LA     13,AREA       Loads into register 13 the address of the save area to be
                                passed.
         CALL   CALLED        Transfers control to the entry point of the called
         .                      program. (The CALL macro instruction generates coding
         .                      that loads a V-type address constant -- CALLED --
         .                      into register 15 and that places into register 14 the
                                return address, that is, the address of the first
                                byte following the macro expansion.)
      (Save Area)
AREA     DC     18F'0'        Reserves the save area of 18 fullwords to be passed to the
                                called program.
      (Identifier List)
ARGLIST  DC     A(ARG1,ARG2)  Creates address constants for the first two identifiers of
                                the identifier list (called ARG1 and ARG2).
         DC     X'80'         Sets to 1 the first bit of the next word.
         DC     AL3(ARG3)     Creates an address constant for this identifier and stores
                                it in the last three bytes of the word. Since the first
                                bit of the first byte of the word is a 1, this third
                                identifier is specified to be the last identifier of the
                                list.
```

Note: Since the calling program containing this coding could have previously 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 62. Sample Linkage Coding Used in a Calling Assembler Language Program

FILE-NAME AND PROCEDURE-NAME ARGUMENTS

A calling COBOL program that calls an assembler-language program can pass file-names and procedure-names, in addition to data-names, as identifiers. In the actual identifier list that the compiler generates, the procedure-name is passed as the address of the procedure. For a queued file, the file-name is passed as the address of the DCB (the data control block); for a basic file, the file-name is passed as the address of the DECB (the data event control block).

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/1, FORTRAN, and, as in the foregoing discussion, assembler language. 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/1 together presents such a large number of problems that a considerable amount of study is necessary to implement anything but the most basic application. For further information, see the publications IBM System/360 Operating System: Linkage Editor and Loader, Order No. GC28-6538; and IBM System/360 Operating System: PL/1 (F) Programmer's Guide, Order No. GC28-6594.

Word No.	Area No.	Contents
1	AREA	Not used by COBOL or assembler language programs.
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 63. Save Area Layout and Contents

LINKAGE EDITING PROGRAMS

Each time an entry point is specified in a called program, an external name is defined. 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. 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.

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 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).

```
        .
        .
        .
        ENTRY   CALLED          Establishes CALLED as an external name that can be
                                  referred to in another program.

        (Save Routine)

CALLED  SAVE    (14,10)         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 10, which are not actually used for
                                  linkage, are saved in subsequent words of the
                                  save area. The implication in not saving
                                  registers 11 and 12 is that they will not be used
                                  in this program. Consequently, there is no need
                                  to save them since their contents will be
                                  unchanged when control is returned. 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      2,13            Loads the address of the save area into register 2,
                                  which subsequently will be used to refer to the
                                  save area.

        LM      3,5,0(1)        Loads into registers 3, 4, and 5 the addresses of
                                  the three identifiers passed to the program. The
                                  address of the identifier list always is passed
                                  in register 1, which here is used as the base
                                  register to get the addresses. Subsequent
                                  references to the first identi- fier will use
                                  register 3 as the base register for that address.
                                  References to the second and third identifiers
                                  will use registers 4 and 5. (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.)

        (Return Routine)

        LR      13,2            Loads into register 13 the address of the save area
                                  that was passed to this program.

        RETURN  (14,10),T,RC=(15)  This RETURN macro instruction restores the saved
                                  registers (14, 15, and 0 through 10). 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 occurred. Specification of RC=(15)
                                  indicates that a return code has been placed in
                                  register 15.
```

Note: If the called program containing this coding also calls another program, it would contain the calling sequence, and it would establish linkage between the save area it had received and the save area it passes to the called program.

Figure 64. Sample Linkage Coding Used in a Called Assembler-Language Program

```
//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=IEUASM,PARM=(LOAD,NODECK),                X
//            COND=(9,LT,STEP1)[1]
//SYSGO         DD       DSN=&&TEMPLIB1,UNIT=SYSSQ,DISP=(MOD,PASS)
//SYSIN         DD       *
              (Source module for ASSMMAIN, a calling assembler-
                language program)
/*
//STEP3         EXEC     PGM=IEWL,PARM=(LIST,XREF,LET),                X
//            COND=((9,LT,STEP1),(5,LT,STEP2))
    .
    .
    .
//PROGLIB1      DD       DSN=&&TEMPLIB1,DISP=OLD
//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 65. Sample Coding Used for a Calling Assembler-Language Program and a Called COBOL Program

```
//JOBX      JOB
//STEP1     EXEC      PGM=IKFCBL00,PARM=LOAD
            .
            .
            .
//SYSLIN    DD        DSNAME=&&GOFILE,DISP=(MOD,PASS),UNIT=SYSSQ
//SYSIN     DD        *

            (Source module for MAIN, a calling program)

/*
//STEP2     EXEC      PGM=IKFCBL00,PARM=LOAD
            .
            .
            .

//SYSLIN    DD        DSNAME=*.STEP1.SYSLIN,DISP=(MOD,PASS)
//SYSIN     DD        *

            (Source module for ADD, a called program)

/*
//STEP3     EXEC      PGM=IKFCBL00,PARM=LOAD
            .
            .
            .

//SYSLIN    DD        DSNAME=*.STEP2.SYSLIN,DISP=(MOD,PASS)
//SYSIN     DD        *

            (Source module for SUBTRACT, a called program)

/*
//STEP4     EXEC      PGM=IEWL
            .
            .
            .

//SYSLIB    DD        DSNAME=SYS1.COBLIB,DISP=OLD
//SYSLMOD   DD        DSNAME=PROGLIB(CALC),DISP=OLD
//ADDLIB    DD        DSNAME=MYLIB,DISP=OLD
//SYSLIN    DD        DSNAME=&&GOFILE,DISP=OLD
//          DD        *
            INCLUDE   ADDLIB(A)
            LIBRARY   ADDLIB (X,Y,Z)
/*
```

Figure 66. Specifying Primary and Additional Input to the Linkage Editor

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 66.

Note: Additional input often contains members of libraries (see "Specifying Libraries as Additional Input" in "Libraries").

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. Its 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.

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.

---

[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.

Example of Linkage Editor Processing

Figure 66 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 library) to form one load module.

STEP1 compiles a source module called MAIN, STEP2 compiles a source module called ADD, and STEP3 compiles a source module called SUBTRACT. 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 ADD and SUBTRACT are placed in the data set behind the object module, MAIN.)

In STEP4, the linkage editor uses the &&GOFILE data set as primary input, and the cataloged libraries MYLIB 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 member A of MYLIB 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 members X, Y, and Z of MYLIB, to resolve these references. MYLIB is specified in the ADDLIB DD statement.

After linkage editor processing is completed, the load module CALC is added as a member to the existing, cataloged library PROGLIB. CALC now contains MAIN, SUBTRACT, ADD, A, and, possibly, COBOL subroutines, and X, Y, and Z.

OVERLAY STRUCTURES

If the called programs needed to execute one COBOL source program do not all fit into main storage at the same time, it is still possible to use them with the overlay technique or with the use of the segmentation feature. 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. The use of segmentation is discussed in "Using the Segmentation Feature."

Considerations for Overlay

Assume a COBOL main program, called COBMAIN, exists that calls at one or more points in its logic the COBOL subprograms: CSUB1, CSUB2, CSUB3, CSUB4, and CSUB5. Also assume that the load module sizes for the main program and the subprograms are given as follows:

Program	Module Size (in Bytes)
COBMAIN	20,000
CSUB1	4,000
CSUB2	5,000
CSUB3	6,000
CSUB4	3,000
CSUB5	4,000

Through the linkage mechanism, CALL SUB1..., all subprograms plus COBMAIN must be linkage 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 three techniques of overlay are available to the COBOL programmer:

- Preplanned overlay using the linkage editor
- Dynamic overlay using macro instructions during execution
- Segmentation Feature

Note: The largest load module that can be processed by Fetch is 524,248 bytes. If a load module exceeds this limit, it should be divided.

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 IBM System/360 Operating System: 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 67 shows the deck arrangement for an overlay structure using preplanned linkage editor overlay. The OVERLAY statements specify to the linkage editor that the overlay structure to be established is one in which SUBPROG A, SUBPROG B, and SUBPROG C overlay each other when called during execution.

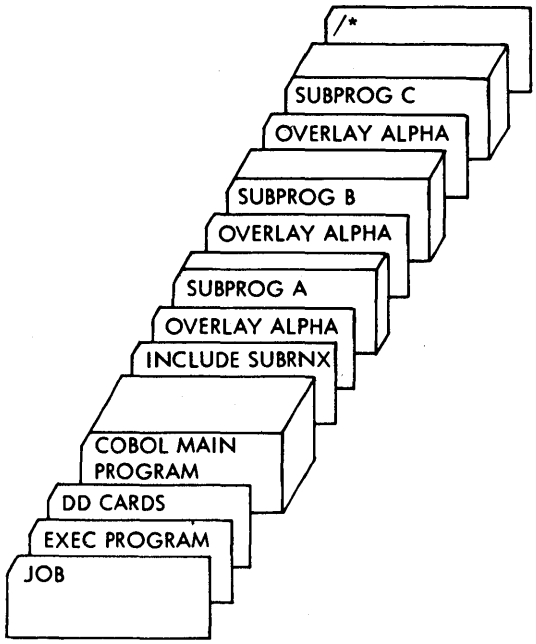


Figure 67. Sample Deck for Linkage Editor Overlay Structure

Dynamic Overlay Technique

In preparation for the dynamic overlay technique, each part of the program that is 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.

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 IBM System/360 Operating System: Supervisor and Data Management 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.

Beyond this, the programmer must ensure that the COBOL subprogram modules exist in a private library (PDS) which is defined by a //JOBLIB DD statement in the job control language for execution of the main program.

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). This requirement has no effect on the COBOL segmentation feature (see the chapter "Use of the Segmentation Feature"), since (1) all members of the object-time subroutine library are link-edited into the root segment, and (2) American National Standard COBOL subprograms may not be segmented.

In addition, the library routine ILBOSTP0 should always be link-edited into the root segment, or a segment higher than any which contain COBOL load modules. This is necessary because all calls (explicit or implicit) to this routine should reference the same copy.

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."

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.

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 called or calling 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: Neither the overlay facility nor the segmentation feature can 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 the data they contain may be specified by a user. Still other libraries have 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 always is 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 68 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. If additional space is required to store a member, allocation will be made in the amount specified by the secondary allocation in the SPACE parameter of the DD statement that was used when the library and its first member were created. 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=2311,                 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 until the MOVE or COPY control statement of the IEHMOVE utility program is used. The space previously occupied by the replaced or deleted member is thus made available. (For further details, see the publication IBM System/360 Operating System: 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.

LIBRARIES PROVIDED BY THE SYSTEM

Link Library

The link library is a partitioned data set that contains load modules to be executed. Unless 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.

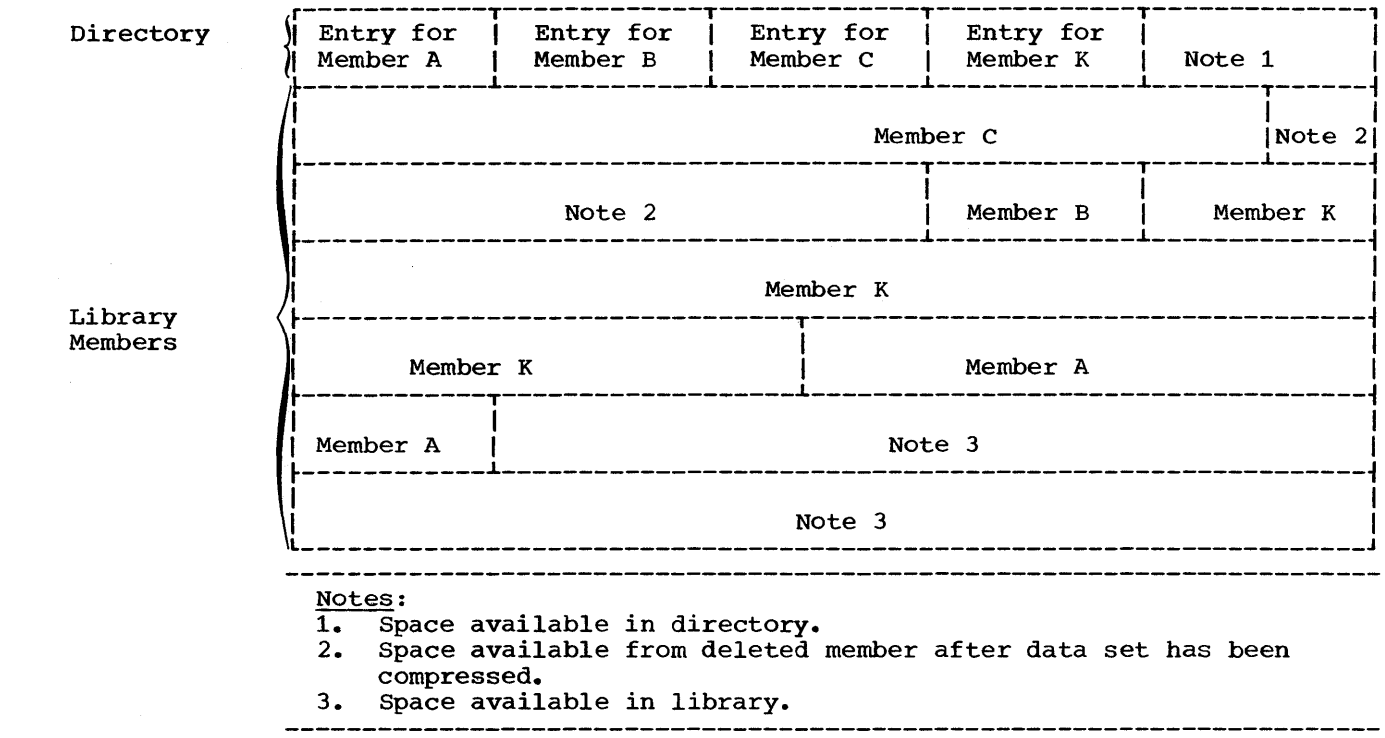


Figure 68. Format of a Library

The link library can be used by the programmer to store executable load modules at linkage editor time. The technique for doing this is described in "Linkage Editor Data Set Requirements."

The link library is identified in a job control statement as SYS1.LINKLIB.

Procedure Library

The procedure library is a partitioned data set whose members are the cataloged procedures at an installation. They 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 is identified by the name SYS1.SORTLIB (see "Using the Sort 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 are included in a COBOL load module to perform such functions as data conversion and double precision arithmetic. The COBOL programmer does not refer directly to these subroutines; calling sequences to them are generated at compile time from certain Procedure Division statements, and they are incorporated into the load module at linkage editor time. A listing of subroutine names, functions, entry points, and size is given in Appendix B.

The system name for the COBOL subroutine library is 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 linkage editor 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. 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=OLD
//         DD DSNAME=MYLIB,DISP=OLD
```

In this case, both the COBOL subroutine library and the partitioned data set named MYLIB are available to the automatic library call.

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 IBM System/360 Operating System: Utilities.

Entering Source Statements

Figure 69 illustrates the method to insert source statements into a copy library member.

The ./ ADD statement is a utility statement that copies CFILEA into the

```
//CATALOG          JOB
//                 EXEC   PGM=IEBUPDTE,PARM=(NEW)
//SYSUT2           DD     DSNAME=COPYLIB,UNIT=2311,                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 69. Entering Source Statements into the COPY Library

```
//UPDATE          JOB
//               EXEC   PGM=IEBUPDTE,PARM=(MOD)
//SYSUT1         DD     DSNAME=COPYLIB,UNIT=2311,                  X
//                          DISP=(OLD,KEEP),                       X
//                          VOLUME=SER=111111,                     X
//                          DCB=(RECFM=F,BLKSIZE=80)
//SYSUT2         DD     DSNAME=COPYLIB,UNIT=2311,                  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 70. Updating Source Statements in a COPY Library

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 10 and each succeeding statement is incremented by 5. The entries following the utility statements are the actual source statements to be cataloged. The ENDUP statement signals the end of the entries to be inserted.

This same procedure can be used to catalog entire source programs.

Updating Source Statements

Figure 70 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. 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 68 -- 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 SYSLIB DD statement, as follows:

```
//SYSLIB  DD   DSNAME=COPYLIB,                X
//               VOLUME=SER=111111,           X
//               DISP=OLD,UNIT=2311
```

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 cataloged source statements into the Data or Environment Divisions. If the programmer wishes to retrieve the member, CFILEA, cataloged 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 identifying 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 also permits the programmer to include previously cataloged source statements into the Procedure Division.

Assume a procedure named DOWORK was cataloged 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 cataloged 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.

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 may delete more than one statement with the first and last sequence numbers to be deleted separated by a hyphen.

Note: The COBOL sequence number is the six-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 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 $6600. The coding to accomplish this is shown in Figure 71.

Now, however, due to a change in the old age tax laws, tax is to be taken out until earnings exceed $7800 and a new percentage is to be placed. The programmer can code these changes as shown in Figure 72.

```
COBOL                                                                  IEBUPDTE
Sequence                                                               Sequence
Numbers                                                                Numbers
000730           IF ANNUAL-PAY GREATER THAN 6600 GO TO PAY-WRITE.      00000105
000735           IF ANNUAL-PAY GREATER THAN 6600 - 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 71. COBOL Statements To Deduct Old Age Tax

```
    .
    .
    .
BASIS PAYROLL
DELETE 000730-000740
000730              IF ANNUAL-PAY GREATER THAN 7800 GO TO PAY-WRITE.
000735              IF ANNUAL-PAY GREATER THAN 7800 - BASE-PAY GO TO LAST-TAX.
000740 TAX-PAYR.  COMPUTE TAX-PAY = BASE-PAY * .044.
```

Figure 72. Programmer Changes to Source Program

```
000730              IF ANNUAL-PAY GREATER THAN 7800 GO TO PAY-WRITE.
000735              IF ANNUAL-PAY GREATER THAN 7800 - 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 73. Changed COBOL Statements to Source COPY Library Statements

The altered program will contain the coding shown in Figure 73.

Note that changes made through use of the INSERT and DELETE statements remain in effect for the one run only.

Note: If both the BASIS card and the COPY statement 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: Check List for Job Control Procedures").

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 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 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."

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.

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 IBM System/360 Operating System: Linkage Editor and Loader.

USING THE CATALOGED PROCEDURES

A cataloged procedure is a set of job control statements that has been 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 four 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 IBM System/360 Operating System: Utilities.

In Figure 74, 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.

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 operating system, 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 providing compilation, linkage editing, and execution.
5. COBUCG providing compilation and loading.

These procedures may be used with any of the job schedulers released as part of the System/360 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 75, 76, 77, 78, and 79. (Space allocations in these procedures are in terms of record lengths on the 2311 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.

```
|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 74. Example of Adding Procedures to the Procedure Library

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 EXEC Statements" 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 three-step procedure wherein the first step compiles a program using COBOL, the second step linkage 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 linkage 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 2311 Disk Storage Drives might be assigned to the SYSSQ name. Then again, both 2400 Magnetic Tape Units and 2311 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 75 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)
```

If the DD * statement is used under MFT, the delimiter statement (/*) must follow the source module. Under MVT, the /* statement is not required.

COBUCL Procedure

The COBUCL procedure is a two-step procedure to compile and linkage edit using the COBOL compiler. Figure 76 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 linkage edit and execute the output of a COBOL compilation. Figure 77 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 and must be the last of the //GO, cards.

```
//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=86K
//SYSPRINT  DD SYSOUT=A
//SYSPUNCH  DD SYSOUT=B
//SYSUT1    DD DSNAME=&&SYSUT1,UNIT=SYSDA,SPACE=(460,(700,100))
//SYSUT2    DD DSNAME=&&SYSUT2,UNIT=SYSDA,SPACE=(460,(700,100))
//SYSUT3    DD DSNAME=&&SYSUT3,UNIT=SYSDA,SPACE=(460,(700,100))
//SYSUT4    DD DSNAME=&&SYSUT4,UNIT=SYSDA,SPACE=(460,(700,100))
```

Figure 75. Statements in the COBUC Procedure

```
//COB       EXEC PGM=IKFCBL00,REGION=86K
//SYSPRINT  DD SYSOUT=A
//SYSUT1    DD DSNAME=&&SYSUT1,UNIT=SYSDA,SPACE=(460,(700,100))
//SYSUT2    DD DSNAME=&&SYSUT2,UNIT=SYSDA,SPACE=(460,(700,100))
//SYSUT3    DD DSNAME=&&SYSUT3,UNIT=SYSDA,SPACE=(460,(700,100))
//SYSUT4    DD DSNAME=&&SYSUT4,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 76. Statements in the COBUCL Procedure

```
//LKED      EXEC PGM=IEWL,PARM='LIST,XREF,LET',REGION=96K
//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 DSNAME=&&SYSUT1,UNIT=(SYSDA,SEP=(SYSLIN,SYSLMOD)),     X
//              SPACE=(1024,(50,20))
//SYSPRINT  DD SYSOUT=A
//GO        EXEC PGM=*.LKED.SYSLMOD,COND=(5,LT,LKED)
```

Figure 77. Statements in the COBULG Procedure

```
//COB        EXEC PGM=IKFCBL00,PARM=SUPMAP,REGION=86K
//SYSPRINT   DD SYSOUT=A
//SYSUT1     DD DSNAME=&&SYSUT1,UNIT=SYSDA,SPACE=(460,(700,100))
//SYSUT2     DD DSNAME=&&SYSUT2,UNIT=SYSDA,SPACE=(460,(700,100))
//SYSUT3     DD DSNAME=&&SYSUT3,UNIT=SYSDA,SPACE=(460,(700,100))
//SYSUT4     DD DSNAME=&&SYSUT4,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 DSNAME=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 78. Statements in the COBUCLG Procedure

```
//COB        EXEC PGM=IKFCBL00,PARM='LOAD',REGION=86K
//SYSPRINT   DD SYSOUT=A
//SYSUT1     DD DSNAME=&&SYSUT1,UNIT=SYSDA,SPACE=(460,(700,100))
//SYSUT2     DD DSNAME=&&SYSUT2,UNIT=SYSDA,SPACE=(460,(700,100))
//SYSUT3     DD DSNAME=&&SYSUT3,UNIT=SYSDA,SPACE=(460,(700,100))
//SYSUT4     DD DSNAME=&&SYSUT4,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 79. Statements in the COBUCG Procedure

COBUCLG Procedure

The COBUCLG procedure is a three-step procedure to compile, linkage edit, and execute using the COBOL compiler. Figure 76 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 and load using the COBOL compiler. Figure 79 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 IBM System/360 Operating System: 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 75, 76, 78, and 79.

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,COND.LKED=       X
//                (9,LT,STEPA.COB),        X
//                PARM.LKED=(MAP,LIST),    X
//                ACCT=(1234)
//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 linkage 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 IBM System/360 Operating System: Job Control Language Reference.

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 75) into an in-stream procedure and execute it twice:

```
//CONVERT   JOB  1234,YOURNAME
//INSTREAM  PROC
//COB       EXEC PGM=IKFCBL00,PARM='DECK,  X
               NOLOAD,SUPMAP',REGION=86K
//SYSPRINT  DD   SYSOUT=A
//SYSPUNCH  DD   SYSOUT=B
//SYSUT1    DD   DSNAME=&&SYSUT1,          X
               UNIT=SYSDA,                 X
               SPACE=(460,(700,100))
//SYSUT2    DD   DSNAME=&&SYSUT2,          X
               UNIT=SYSDA,                 X
               SPACE=(460,(700,100))
//SYSUT3    DD   DSNAME=&&SYSUT3,          X
               UNIT=SYSDA,                 X
               SPACE=(460,(700,100))
//SYSUT4    DD   DSNAME=&&SYSUT4,          X
               UNIT=SYSDA,                 X
               SPACE=(460,(700,100))
//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.
- A DD statement with an * in the operand field terminates processing of subsequent DD statements in both the procedure and the input stream for the job step, but not necessarily for the job.

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 parameter. Within the DCB parameter, individual subparameters may be overridden.
- To nullify a keyword parameter (except the DCB parameter), 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 75, 77, 78, and 79.

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. Note that if the statement DD * or the statement DD DATA is used, it must be the last to appear in a series of DD statements.

```
//jobname       JOB  1234,J.SMITH
//stepname      EXEC COBUCLG,                X
//             PARMCOB=(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 any 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.

When using the DDNAME parameter, the following should also be kept 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. Without the DDNAME parameter, replacing the first DD statement with a DD * statement would terminate processing of subsequent statements in the job step. 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:

```
//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.

In order to use the IBM System/360 Operating System Sort/Merge program, Sort feature statements are written in the COBOL source program. These statements are described in the publication IBM System/360 Operating System: Full American National Standard COBOL. The Sort/Merge program itself is described in the publication IBM System/360 Operating System: Sort/Merge Program. In this publication, the system requirements when the Sort feature is used are discussed in "Machine Considerations."

DD statements must be written in the execution-time job steps of the procedure to describe the data sets used by the sort program. DD statements for data sets used during the sort process are described in the section "Sort DD Statements."

Note: The Sort/Merge Checkpoint Restart feature is available to the programmer who uses the COBOL SORT statement through the use of the RERUN statement.

SORT 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.

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 will generate 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 may be assigned to either magnetic tape or mass storage devices. All of the intermediate storage for one sort 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-track)
IBM 2400-series Magnetic Tape Unit (9-track)
IBM 2311 Disk Storage Drive
IBM 2314 Disk Storage Drive
IBM 2301 Drum Storage
IBM 3330 Disk Storage[1]

The publication IBM System/360 Operating System: Sort/Merge Program, 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 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

---

[1]Only the 5734-SM1 Sort/Merge Program Product supports this device.

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 on 7-track tape when the input data set is on 9-track tape.
2. May be on 7-track tape when the output data set is on 9-track tape.
3. Cannot use the data conversion feature if they are on 7-track tape. The TRTCH subparameter must reflect this.
4. 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=SYSDA,               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 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 modules.

Note: At system generation time, the programmer can designate that SORT diagnostic messages be printed either on the console or on the unit designated SYSOUT. If the system is generated to write SORT messages on SYSOUT, these messages may overprint any COBOL output assigned to SYSOUT. For example, if the programmer has selected SYSOUT on which to print a report in the output procedure associated with the execution of the COBOL SORT statement, any SORT messages will be interspersed within that report. If it is not possible to assign the SORT messages to the console, the programmer should assign his COBOL output to temporary files and print the reports at a later time.

In addition, since COBOL and the Sort/Merge Program use certain different DCB parameters (such as RECFM, LRECL, etc.), abnormal termination may result if SYSOUT is used jointly (usually an 001 or 60A ABEND).

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 STATEMENT IN A JOB

More than one SORT 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 80 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 System/360 Operating System: Full American National Standard COBOL.

```
//SORTEST     JOB   NY838670165,          X
//                  'J.SMITH',            X
//                  MSGLEVEL=1
//SORTJS3     EXEC  COBUCLG
//COB.SYSIN   DD    *
                    .
                    .
                    .

      (COBOL source program)
                    .
                    .
                    .
//GO.SORTWK01 DD    UNIT=2311,            X
//                  SPACE=(TRK,(200),     X
//                  ,CONTIG)
//GO.SORTWK02 DD    UNIT=2311,            X
//                  SPACE=(TRK,(200),     X
//                  ,CONTIG)
//GO.SORTWK03 DD    UNIT=2311,            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 80. 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 publication IBM System/360 Operating System: Sort/Merge).

CATALOGING SORT DD STATEMENTS

Since repeated use of the Sort feature often involves the same execution time DD statements, the user may wish to catalog them (see "Using the Cataloged Procedures").

When using the COBOL RERUN feature, all SORT messages are written on the console.

SORT DIAGNOSTIC MESSAGES

The messages generated by the Sort Feature are listed in the publications IBM System/360 Operating System: Sort/Merge, and IBM System/360 Operating System: Messages and Completion Codes. The identifying characters in a sort message are IER.

LINKAGE WITH THE SORT/MERGE PROGRAM

Communication between the Sort/Merge program and the COBOL program is maintained by the COBOL library subroutine ILBOSRT0.

ILBOSRT0 dynamically executes a LINK to the Sort/Merge program using the linkage name 'IERRCO00'.

If the INPUT PROCEDURE option of the SORT 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.

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, 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 publication IBM System/360 Operating System: Sort/Merge 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 DATE,
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 name
```

See "Appendix J: Diagnostic Messages" for a description of action to be taken.

A normal SORT operation will produce the following messages from the Sort/Merge Program:

```
IER036I   IER049I
IER037I   IER055I
IER038I   IER054I
IER045I   IER052I
```

Others may appear depending on COBOL options specified.

LOCATING SORT RECORD FIELDS

Records defined under a COBOL SD are assigned a BLL (Base Locator for Linkage Section), rather than a BL (Base Locator) as is done with other records. 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 BY AN INPUT PROCEDURE

For debugging purposes, it is sometimes useful to determine the last input record released to the Sort program. The following procedure should be used:

1. From the Data Division map, determine the BLL number of the SORT 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 core dump will point to the last record released to SORT.

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.

SORT/MERGE CHECKPOINT/RESTART

The CHECKPOINT/RESTART feature is available to the programmer using the COBOL SORT 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 program, during the execution of all SORT statements in the program. This RERUN clause is fully described in the publication IBM System/360 Operating System: 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 system generation time, the program assumes a maximum of 15,500 bytes. The sort program requests 12,000 bytes leaving 3,500 bytes for system functions. Performance usually improves as the program is given more main storage. Approximately 44K bytes of main storage are needed for efficient execution of the sort/merge program, and performance increases 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 core storage available in addition to the space needed for Data Management and the COBOL program. If this amount of core storage is not available, the program will terminate abnormally.

The programmer can override the amount of main storage specified at system generation time by using the SORT-CORE-SIZE special register.

The value in SORT-CORE-SIZE is the amount of main storage (in bytes) with which the programmer would want to operate. It cannot be less than 12,000, and at this value, some combinations of input/output devices and record lengths make a successful sort impossible. The main storage value is changed only for the current job step; afterwards, the value reverts to the one specified at system generation time.

Changing the main storage allocation is useful when a sort/merge application in a multiprogramming environment. By reducing the amount of main storage allocated to sort, so that other programs can have the storage they need to operate simultaneously, the performance of sort is impaired. By increasing the allocation, a large sort application runs more efficiently but the performance of other jobs sharing the multiprogramming environment is impaired.

Defining Variable-Length Records

If the input records used are 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 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 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 program would assume a low modal length and might not allocate a large enough record storage area to sort data.

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 occurrences of the item represents 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.

To avoid this problem, specify a dummy sort record of a fixed length, with no OCCURS DEPENDING ON clauses, with the size of the smallest sort record described in OCCURS DEPENDING ON clauses.

For a complete discussion, see the publication IBM System/360 Operating System: Sort/Merge Program.

Sorting Variable-Length Records

Figure 81 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.

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 never be used. There is no way for the user to set the correct length of the receiving field.

Figure 81. Sorting Variable-Length Records Whose File-Name Description and Sort-File-Name Description Correspond

USE OF SEGMENTATION FEATURE

Segmentation is a facility that provides a means of accomplishing object time overlay as a result of specifications made at the source language level. The programmer may divide the Procedure Division of a source program into sections. Through the use of a system of priority numbers, certain sections are designated as permanently resident in core and other sections as overlayable fixed segments and/or independent segments. Thus, a large program can be executed in a defined area of core storage by limiting the number of segments in the program that are permanently resident in core storage.

Note: The segmentation feature is not available when the loader is used.

Suppose that because of core storage limitations, the program SAVECORE is segmented as shown in Figure 82. Only those segments that have priority numbers less than or equal to the segment limit of 15 are designated as permanently resident.

Assuming that 12K is available for the program SAVECORE, Figure 83 shows that manner in which core storage would be utilized. Sections 3 and 6, and sections 5 and 7 are considered logical units since they have the same priority numbers. Sections 3 and 6 can be in core together but cannot have section 7 at the same time. Likewise, sections 5 and 7 can be in core together but cannot have section 3 at the same time.

Sections in the 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 segment limit.

Sections in the overlayable fixed segment are sections that are less frequently used. These sections are always made available in the state in which they were last used. They are distinguishable here by the fact that they have been assigned priority numbers greater than the segment limit but less than 50.

Sections in the independent segment can overlay, and be overlaid by, either an overlayable fixed segment or another independent segment. Independent segments are those assigned priority numbers greater than 49 and less than 100. These independent segments are returned to their initial state when they are brought into core storage.

```
IDENTIFICATION DIVISION.

PROGRAM-ID.  SAVECORE.
.
.
.

ENVIRONMENT DIVISION.

OBJECT-COMPUTER.  IBM-360-H50
     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 82. Segmentation of Program SAVECORE

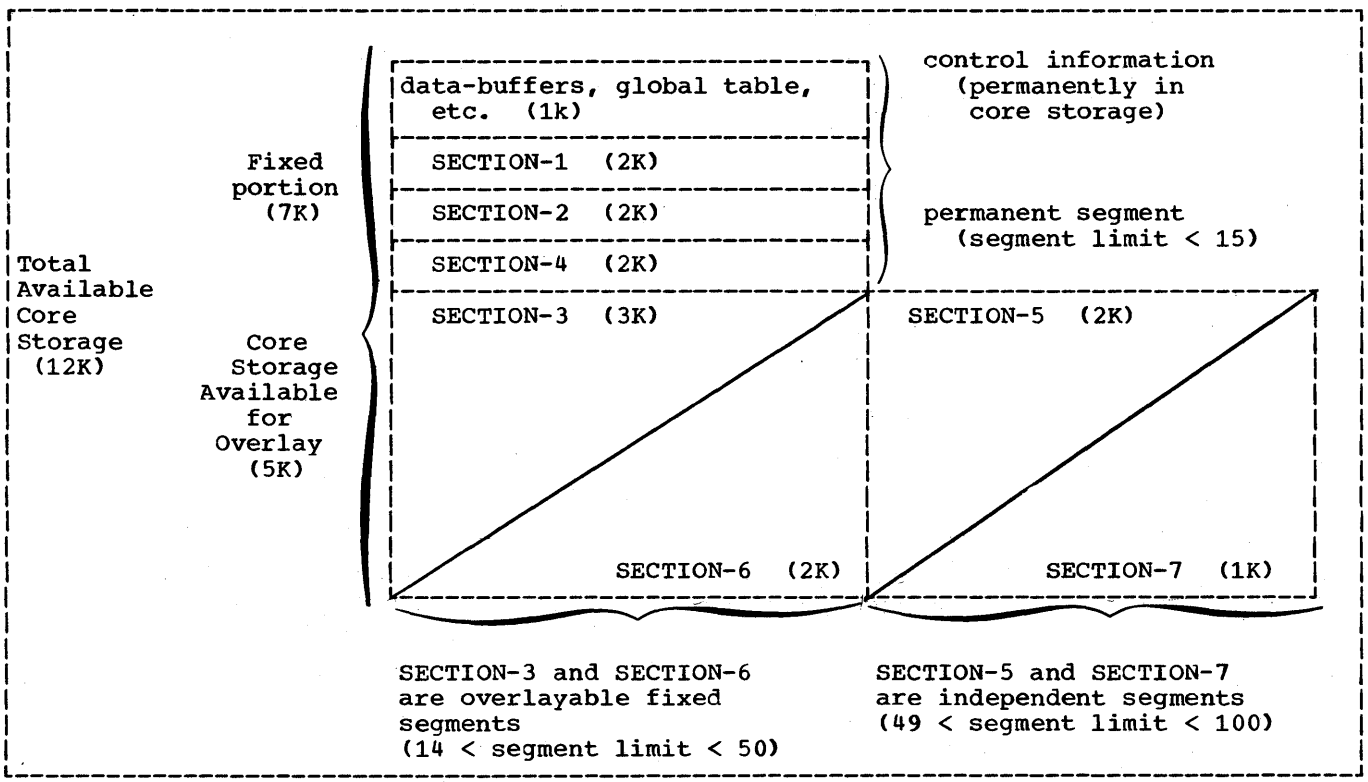


Figure 83. Storage Layout for SAVECORE

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 Full American National Standard 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 Full American National Standard 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 is 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 that will initiate the execution of the first overlayable or 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.

COMPILER OUTPUT

The output produced by the compiler is an overlay structure consisting of multiple object modules preceded by linkage editor control statements. 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 contains no procedure-name or verbs and is given in this sequence: all sections with priorities greater than the segment limit are listed first in ascending order by priority number, followed by the root segment.

If the CLIST option is specified for a condensed listing, the word VERB will be printed in place of the actual verb.

Figure 84 shows the output of a sample segmentation program.

JOB CONTROL CONSIDERATIONS

In order to execute a segmented program, the programmer must specify OVLY in the parameter field of the linkage editor EXEC statement. Note that when using the IBM-supplied cataloged procedures, the LIST and LET parameters must be respecified.

```
00001    000060 IDENTIFICATION DIVISION.
00002    000070 PROGRAM-ID. SEG-SAMPLE.
00003    000080 AUTHOR. PROGRAMMER-NAME.
00004    000090 REMARKS.
00005    000100     SPECIAL OPERATOR INSTRUCTIONS - NONE.
00006    000110     INPUT REQUIRED - NONE.
00007    000120     PURPOSE
00008    000130         TO CREATE A SINGLE FILE ON DISK USING
00009    000140         QSAM/DTFSD, AND READ  IT BACK.
00010    000150         PROGRAM USES SEGMENTATION
00011    000160         WITH FILE PROCESSING SPREAD OVER
00012    000170         THE PERMANENT, OVERLAYABLE FIXED,
00013    000180         AND INDEPENDENT SEGMENTS.
00014    000190     EXPECTED RESULTS
00015    000200         START TEST SEG-SAMPLE
00016    000210         (EACH SEGMENT DISPLAYS ITS SEGMENT NUMBER
00017    000220         AND FUNCTION)
00018    000230         END TEST SEG-SAMPLE SUCCESSFUL RUN.
00019    000240         SECTIONS WHILE WRITING APPEAR
00020    000250         IN ORDER 80, 20, 30, 60, 40.
00021    000260         SECTIONS WHILE READING APPEAR
00022    000270         IN ORDER 80, 60, 30, 40, 20.
00023    000280     ERROR INDICATIONS
00024    000290         **ERROR DISK SEQ I/O**
00025    000300         **ERROR END OF EXTENT WRITING AFTER (RECORD)**
00026    000310         **ERROR UNEXPECTED EOF READING AFTER
00027    000320             RECORD (RECNO)**
00028    000330         **ERROR EOF NOT FOUND**
00029    000340         **RECORD IS (RECNO)
00030    000350           SHOULD BE (RECNO)**
00031    000360         PROGRAM CONTAINS PERFORMS FROM BASE SECTION
00032    000390         TO PERMANENT, OVERLAYABLE FIXED, AND INDEPENDENT
00033    000400         SEGMENTS.
00034    000410         ALSO CONTAINS PERFORMS FROM INDEPENDENT TO PERMANENT
00035    000420         AND FROM OVERLAYABLE FIXED TO PERMANENT SEGMENTS.
00036    000430         ALSO CONTAINS PERFORMS ENTIRELY WITHIN A SEGMENT IN
00037    000440         IN EACH CATEGORY.
00038    000450 ENVIRONMENT DIVISION.
00039    000460 CONFIGURATION SECTION.
00040    000470 SOURCE-COMPUTER. IBM-360-40.
00041    000480 OBJECT-COMPUTER. IBM-360-40
00042    000490     MEMORY SIZE 64000 CHARACTERS
00043    000500     SEGMENT-LIMIT IS 25.
00044    000510 INPUT-OUTPUT SECTION.
00045    000520 FILE-CONTROL.
00046    000530     SELECT FILE-1 ASSIGN TO        DA-2311-S-DKSQ01A.
00047    000540 DATA DIVISION.
00048    000550 FILE SECTION.
00049    000560 FD  FILE-1
00050    000570     RECORDING MODE IS F
00051    000580     LABEL RECORDS OMITTED
00052    000590     DATA RECORD IS RECRD1.
00053    000600 01  RECRD1 PICTURE X(83).
00054    000610 WORKING-STORAGE SECTION.
```

Figure 84. Sample Segmentation Program (Part 1 of 5)

```
00055    000620 77  ERRORSW PIC A VALUE SPACE.
00056    000630 77  ERCTPL  PIC S99 VALUE ZERO.
00057    000640 77  MSGHDR  PIC X(22) VALUE '**ERROR DISK SEQ I/O**'.
00058    000650 77  MSGEOX PIC X(36)
00059    000660     VALUE '**ERROR END OF EXTENT WRITING AFTER '.
00060    000670 77  MSGECF  PIC X(37)
00061    000680     VALUE '**ERROR UNEXPECTED ECF READING AFTER '.
00062    000690 77  MSGNEF  PIC X(23) VALUE '**ERROR EOF NOT FOUND**'.
00063    000700 01  REC1.
00064    000710     02  REC-ID.
00065    000720     03  REC-HD  PIC X(4) VALUE 'RECD'.
00066    000730     03  REC-NO  PIC S9(4) VALUE ZERO.
00067    000740     02  FILLER  PIC A(75) VALUE SPACES.
00068    000750     66  RECID RENAMES REC-ID.
00069    000760 01  VER-REC.
00070    000770     02  VER-ID.
00071    000780     03  VER-HD  PIC X(4) VALUE 'RECD'.
00072    000790     03  VER-NO  PIC S9(4) VALUE ZERO.
00073    000800 PROCEDURE DIVISION.
00074    000810 BASE-SECTION SECTION 0.
00075    000820     DISPLAY 'START TEST SEG-SAMPLE'.
00076    000830     OPEN OUTPUT FILE-1.
00077    000840     PERFORM W-80-0 THRU W-80-9.
00078    000850     PERFORM W-30-0 THRU W-30-9.
00079    000860     PERFORM W-60-0 THRU W-60-9.
00080    000870     PERFORM W-40-0 THRU W-40-9.
00081    000880 BASE-50.
00082    000890     CLOSE FILE-1.
00083    000900     OPEN INPUT FILE-1.
00084    000910     PERFORM R-80-0 THRU R-80-9.
00085    000920     GO TO R-60-0.
00086    000930 BASE-60.
00087    000940     PERFORM R-40-0 THRU R-40-9.
00088    000950     READ FILE-1 INTO REC1 AT END GO TO BASE-70.
00089    000960     DISPLAY MSGHDR DISPLAY MSGNEF
00090    000970     MOVE 'E' TO ERRORSW.
00091    000980 BASE-70.
00092    000990     CLOSE FILE-1.
00093    001000 BASE-90.
00094    001010     IF ERRORSW IS EQUAL TO 'E'
00095    001020     DISPLAY 'END TEST SEG-SAMPLE UNSUCCESSFUL RUN' ELSE
00096    001030     DISPLAY 'END TEST SEG-SAMPLE SUCCESSFUL RUN'.
00097    001040     STOP RUN.
00098    001050 SECTION-20 SECTION 20.
00099    001060 W-20-0.
00100    001070     DISPLAY 'SECTION 20 WRITE'.
00101    001080     NOTE ENTERED BY PERFORM FROM W-80-0.
00102    001090     PERFORM W-21-0 THRU W-21-9 5 TIMES.
00103    001100 W-20-9.
00104    001110     EXIT.
00105    001120 W-21-0.
00106    001130     WRITE RECFD1 FROM REC1 INVALID KEY
00107    001140     DISPLAY MSGHDR
00108    001150     DISPLAY MSGEOX RECID
00109    001160     MOVE 'E' TO ERRORSW
00110    001170     GO TO BASE-50.
00111    001180     ADD 0001 TO REC-NO.
             .
             .
             .
```

Figure 84. Sample Segmentation Program (Part 2 of 5)

```
           .
           .
           .
00237    002440 SECTION-80 SECTION 80.
00238    002450 W-80-0.
00239    002460     DISPLAY 'SECTICN 80 WRITE'.
00240    002470     NOTE ENTERED BY PERFORM FROM BASE-SECTICN.
00241    002480     PERFORM  W-81-0  THRU W-81-9 7 TIMES.
00242    002490     PERFORM W-20-0 THRU W-20-9.
00243    002500 W-80-9.
00244    002510     EXIT.
00245    002520 W-81-0.
00246    002530     WRITE RECFD1 FROM REC1 INVALID KEY
00247    002540     DISPLAY MSGHDR
00248    002550     DISPLAY MSGEOX RECID
00249    002560     MOVE 'E' TC ERRORSW
00250    002570     GO TO BASE-50.
00251    002580     ADD 0001 TC REC-NC.
00252    002590 W-81-9.
00253    002600     EXIT.
00254    002610 R-80-0.
00255    002620     DISPLAY 'SECTION 80 READ'.
00256    002630     NCTE ENTERED BY PERFORM FRCM BASE-50.
00257    002640     PERFCRM R-81-0 THRU R-81-9 17 TIMES.
00258    002650 R-80-9.
00259    002660     EXIT.
00260    002670 R-81-0.
00261    002680     READ FILE-1 INTC REC1 AT END
00262    002690     DISPLAY MSGHDR DISPLAY MSGEOF
00263    002700     ADD 4 TO ERCTFL MOVE 'E' TC ERRCRSW
00264    002710     GO TC R-81-9.
00265    002720     IF REC-ID IS NOT EQUAL TC VER-ID
00266    002730     DISPLAY MSGHDR DISPLAY 'EXPECTED ' VER-ID ' FOUND ' REC-ID
00267    002740     ADD 1 TO ERCTFL MOVE 'E' TC ERRORSW
00268    002750     MCVE REC-ID TC VER-ID.
00269    002760     ADD 1 TC VER-NC.
00270    002770 R-81-8.
00271    002780     IF ERCTFL IS GREATER THAN 3
00272    002790     GO TO BASE-7C.
00273    002800 R-81-9.
00274    002810     EXIT.
```

Figure 84. Sample Segmentation Program (Part 3 of 5)

```
                                CROSS-REFERENCE DICTIONARY

DATA NAMES                      DEFN     REFERENCE

FILE-1                          00046    00076  00076  00082  00082  00083  00083  00088  00088  00092  00092  00106
                                         00106  00121  00121  00141  00141  00155  00155  00175  00175  00191  00191
                                         00211  00211  00225  00225  00246  00246  00261  00261
RECFD1                          00053    00088  00106  00121  00141  00155  00175  00191  00211  00225  00246  00261
ERRORSW                         00055    00090  00094  001C9  00123  00127  00144  00157  00161  00178  00193  00197
                                         00214  00227  00231  00249  00263  00267
ERCTFL                          00056    00123  00123  00127  00127  00131  00157  00157  00161  00161  00165  00193
                                         00193  00197  00197  00201  00227  00227  00231  00231  00235  00263  00263
                                         00267  00267  00271
MSGHDR                          00057    00089  00107  00122  00126  00142  00156  00160  00176  00192  00196  00212
                                         00226  00230  00247  00262  00266
MSGEOX                          00058    001C8  00143  00177  00213  00248
MSGEOF                          00060    00122  00156  00192  00226  00262

                                  .
                                  .
                                  .

PROCEDURE NAMES                 DEFN     REFERENCE

BASE-50                         00081    00110  00145  00179  00215  00250
BASE-60                         00086    00153
BASE-70                         00091    00088  00132  00166  00202  00236  00272
W-20-0                          00099    00242
W-21-0                          00105    00102
R-20-0                          00114    00187
R-21-0                          00120    00117
R-21-9                          00130    00124
SECTION-30                      00133    00132
W-30-0                          00134    00078
W-31-0                          00140    00137
R-30-0                          00149    00223
R-31-0                          00154    00152
R-31-9                          00164    00158

                                  .
                                  .
                                  .


 F88-LEVEL LINKAGE EDITOR OPTIONS SPECIFIED CVLY,XREF,LIST
         VARIABLE OPTIONS USED - SIZE=(153600,51200)              DEFAULT OPTION(S) USED
IEW0000      INSERT SEGOSAMP
IEW0000      OVERLAY A
IEW0000      INSERT SEGOSA30
IEW0000      OVERLAY A
IEW0000      INSERT SEGOSA40
IEW0000      OVERLAY A
IEW0000      INSERT SEGOSA60
IEW0000      OVERLAY A
IEW0000      INSERT SEGOSA80
IEW0000      ENTRY SEGOSAMP
```

Figure 84. Sample Segmentation Program (Part 4 of 5)

```
                                CROSS REFERENCE TABLE

  CONTROL SECTION                          ENTRY

    NAME     ORIGIN  LENGTH  SEG. NO.      NAME   LOCATION     NAME   LOCATION     NAME   LOCATION     NAME   LOCATION

 $SEGTAB         00      2C      1
  SEGOSAMP       30      DAE     1
  ILBODSP0*     DE0     5E2     1
  ILBOERR0*   13C8      20E     1

                                           ILBCERR1   13C8    ILBOERR2   140A    ILBOERR4   144C    ILBOERR3   146E
                                           ILBOERR5   14D2
  ILBOSGM0*   15D8        D4    1
                                           CURSEGM    16A9
  ILBOSTP0*   16E0        35    1
                                           ILBOSTP1   16C6
 $ENTAB       16F8        3C    1


  LOCATION  REFERS TO SYMBOL  IN CONTROL SECTION  SEG. NO.   LOCATION  REFERS TO SYMBOL  IN CONTROL SECTION  SEG. NO.

      4F0          SEGCSA30        SEGOSA30         2          4F8          SEGOSA40        SEGOSA40         3
      500          SEGOSA60        SEGCSA60         4          508          SEGCSA80        SEGOSAE0         5
      110          ILBOERR1        ILBCERR0         1          510          ILBCSTP0        ILBOSTP0         1
      514          ILBODSP0        ILBCDSP0         1          518          ILBOSGM0        ILBOSGM0         1
      51C          ILBCSTP1        ILBCSTP0         1           50          SEGOSA30        SEGOSA30         2

                          .
                          .
                          .


  CONTROL SECTION                          ENTRY

    NAME     ORIGIN  LENGTH  SEG. NO.      NAME   LOCATION     NAME   LOCATION     NAME   LOCATION     NAME   LOCATION

  SEGOSA80    1728      28A     5


  LOCATION  REFERS TO SYMBOL  IN CONTROL SECTION  SEG. NO.   LOCATION  REFERS TO SYMBOL  IN CONTROL SECTION  SEG. NO.

 ENTRY ADDRESS       30
 TOTAL LENGTH      19B8

****RUN       DOES NOT EXIST BUT HAS BEEN ADDED TO DATA SET
```

Figure 84. Sample Segmentation Program (Part 5 of 5)

USING THE CHECKPOINT/RESTART FEATURE

The IBM System/360 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 is available with both sequential and priority scheduling systems. 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 applies to the use of the RERUN clause.

The Checkpoint/Restart feature is fully described in the publication IBM System/360 Operating System: Supervisor Services.

TAKING A CHECKPOINT

In order to initiate a checkpoint, the programmer uses job control statements and 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 is processed or when end of volume is reached while processing a file. The programmer decides when he wants the checkpoints taken when 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.

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} ,PASS),        X
                    {MOD}
//           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} ,PASS),KEEP,   X
                    {MOD}
```

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 be sequential.

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 IBM System/360 Operating System: Supervisor Services.)
- 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. The HOLD option is applicable only in a multiprogramming environment.

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 if restart is to occur in an MVT environment.

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 IBM System/360 Operating System: Operator's Reference, Order No. GC28-6691.

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 85 illustrates a sequence of control statements for restarting a job. 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 84) is included in a job, a SYSCHK DD statement must not appear before the first EXEC statement for a job.

Figure 86 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.

```
//PAYROLL     JOB    MSGLEVEL=1,REGION=80K,RESTART=(STEP1,CHECKPT4)
//JOBLIB      DD     DSNAME=PRIV.LIB3,DISP=OLD
//SYSCHK      DD     DSNAME=CHKPTLIB,UNIT=2311,VOL=SER=456789,        X
//                   DISP=(OLD,KEEP)
//STEP1       EXEC   PGM=PROG4,TIME=5
```

Figure 85. Restarting a Job at a Specific Checkpoint Step

```
//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
```

Figure 86. 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
```

Figure 87. Modifying Control Statements Before Resubmitting for Step Restart

```
//J3412     JOB   386,SMITH,MSGLEVEL=1,RD=R,RESTART=(*.C0000002)
//S1        EXEC  MYPROG
//SYSCHK    DD    DSNAME=C91468,DISP=OLD
//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
```

Figure 88. Modifying Control Statements Before Resubmitting for Checkpoint Restart

Figure 87 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 87, 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 88 illustrates those modifications that might be made to control statements before resubmitting the job 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 88, are the DSNAME and DISP parameters.

This chapter contains information concerning system requirements for the COBOL compiler, execution time, and the sort feature. Additional information for use in estimating the main and auxiliary storage requirements is contained in the publication IBM System/360 Operating System: Storage Estimates.

MINIMUM MACHINE REQUIREMENTS FOR THE COBOL COMPILER

The basic system requirements for use of the COBOL compiler are:

- A System/360 (at least a Model 40) or a System/370 model, with a minimum of 80K (81,920) bytes of main storage available to the compiler, and 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.

  At least 80K (81,920) 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 80K. If more storage is allocated, the compiler will run more efficiently.

  Notes: Before deciding on a value for the SIZE option, the programmer should consider all of the following:

  1. The value of compiler data set SPACE parameters. Given limited storage under MFT, if the primary space allocation for compiler data sets is too small and secondary extents are needed, the system must often use the compiler linkage area for the respective data extent block. Such action often results in either an 80A abnormal termination, if the space limitations are encountered when an attempt is made to load a compiler phase, or diagnostic message IKF0020I-D, if more extensive core has to be allocated for table space for compiler processing.
  2. The size and/or complexity of the program to be compiled. A large or complex program requires more table space than a small or simple one. Accordingly, this table space must be relected in the SIZE parameter chosen. (For further discussion of table requirements, see "Table Handling Considerations.")
  3. The blocking factors used for compiler data sets. The SIZE parameter (and BUF parameter) reflect the increased buffer size needed to handle blocked compiler data sets (see "Appendix D: Compiler Optimization.")

- Compiler Work Files -- Four utility data sets named SYSUT1, SYSUT2, SYSUT3, and SYSUT4. At least one mass storage device, such as an IBM 2311 Disk Storage Drive, for residence of the operating system and SYSUT1. Both the operating system and SYSUT1 may reside on the same volume. SYSUT2, SYSUT3, and SYSUT4 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 1052 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.

MULTIPROGRAMMING WITH A VARIABLE NUMBER OF TASKS (MVT)

REGION Parameter

COMPILATION: If the compiler is being executed under the MVT control program of the System/360 Operating System, the REGION parameter, specified as 86K 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, rounded to the next highest 2K multiple, plus 6K.

The following examples illustrate both the default and the override cases:

Example 1

```
//JOB1   JOB   1234,J.SMITH
//STEP1  EXEC  COBUC
         .
         .
         .
```

In Example 1, the programmer accepts the REGION default value of 86K specified in the COBUC cataloged procedure.

Example 2

```
//JOB2   JOB   1234,J.SMITH
//STEP1  EXEC  COBUCLG,REGION=134K,   X
//             PARM.COB='SIZE=130000'
         .
         .
         .
```

In Example 2, the REGION default value is overridden. Rounding 130000 to the next highest 2K multiple, it becomes 131072, or 128K. Thus, the correct region size is 128K+6K=134K (where K=1024 bytes).

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 linkage edited with all of the necessary object time subroutines.
2. The size of the input/output buffers being used, multiplied by the blocking factor (standard 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 IBM System/360 Operating System: Storage Estimates).
4. Any GETMAIN macro instruction executed for USE LABELS, etc.
5. An additional 4K bytes.
6. If the Sort feature is used, 15,360 bytes plus any additional core storage assigned via the SORT-CORE-SIZE special register.

Intermediate Data Sets Under MVT

Except when the Direct SYSOUT Writer is used, SYSIN and SYSOUT data sets reside 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.

SYSIN-SYSOUT CHARACTERISTICS: The input and output data set characteristics are determined by the system, but can be altered by the programmer if necessary. The procedure used to alter the default values depends on whether the data set is used for input or output, as follows:

- For SYSIN data -- the programmer must request, at the time the job is submitted, that the operator use one of the several reader procedures available. Reader procedures are cataloged procedures that control the reader and vary according to the blocking factor specified.
- For SYSOUT data -- the programmer must use override statements as described in "Using the Cataloged Procedures."

When a job is being run in an MFT or MVT environment, the SPACE parameter assumes added importance. Output is placed in the SYSOUT intermediate data set, except when the Direct SYSOUT Writer is used. When the Direct SYSOUT Writer is used, output goes directly to the printer, punch, or tape as in systems with the primary control program. 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))
```

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 (2301, 2303, 2311, ..., etc.). To avoid this consideration, the average record-length subparameter can be used.

MULTIPLE OPEN AND CLOSE STATEMENTS: Under the MVT control program, input data following the DD * or DD DATA card becomes a single data set. Once a CLOSE statement is encountered. The data set is repositioned to the beginning of the data set. To avoid errors, the programmer should keep this in mind when using more than one OPEN and CLOSE statement for a data set assigned to SYSIN.

Note: Under MVT, a file must be closed before the STOP RUN or EXIT PROGRAM statement is executed. Failure to do this results in an abnormal termination.

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

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 FEATURE CONSIDERATIONS

The basic requirements for use of the Sort feature are:

- A System/360 model or System/370 with sufficient main storage to accommodate the load module to be executed plus a minimum of 15,360 bytes for execution of the sort program and any additional core storage assigned to the sort 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 following is a sample COBOL program and the output listing resulting from its compilation, linkage editing, and execution. The program creates a blocked, unlabeled, standard 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."

```
//TEST   JOB  NY16090101,'SCHOEN 1',MSGLEVEL=1,CLASS=C
//JOBLIB  DD   DSN=PRODTEST,DISP=SHR
//STEP1  EXEC  PGM=IKFCBL00,PARM='DMAP,PMAP,XREF,QUOTE',REGION=86K
//SYSUT1 DD    DSNAME=&&UT1,UNIT=SYSDA,SPACE=(TRK,(100,10))
//SYSUT2 DD    DSNAME=&&UT2,UNIT=SYSDA,SPACE=(TRK,(100,10))
//SYSUT3 DD    DSNAME=&&UT3,UNIT=SYSDA,SPACE=(TRK,(100,10))
//SYSUT4 DD    DSNAME=&&UT4,UNIT=SYSDA,SPACE=(TRK,(100,10))
//SYSLIN DD    DSNAME=&&PNCH,UNIT=SYSDA,SPACE=(TRK,(100,10)),          X
//             DISP=(NEW,PASS)
//SYSPRINT DD  SYSOUT=A
//SYSIN  DD    *
IEF236I ALLOC. FOR TEST     STEP1
IEF237I 234   ALLOCATED TO JOBLIB
IEF237I 230   ALLOCATED TO SYSUT1
IEF237I 190   ALLOCATED TO SYSUT2
IEF237I 235   ALLOCATED TO SYSUT3
IEF237I 190   ALLOCATED TO SYSUT4
IEF237I 230   ALLOCATED TO SYSLIN
IEF237I 00C   ALLOCATED TO SYSIN
```

```
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. JAN 27,1970
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   100100 ENVIRONMENT DIVISION.
00011   100110 CONFIGURATION SECTION.
00012   100120   SOURCE-COMPUTER. IBM-360-H50.
00013   100130   OBJECT-COMPUTER. IBM-360-H50.
00014   100140 INPUT-OUTPUT SECTION.
00015   100150 FILE-CONTROL.
00016   100160     SELECT FILE-1 ASSIGN TO UT-2400-S-SAMPLE.
00017   100170     SELECT FILE-2 ASSIGN TO UT-2400-S-SAMPLE.
```

```
00018   100180 DATA DIVISION.
00019   100190 FILE SECTION.
00020   100200 FD  FILE-1
00021   100210     LABEL RECORDS ARE OMITTED
00022   100220     BLOCK CONTAINS 100 CHARACTERS
00023   100225     RECORD CONTAINS 20 CHARACTERS
00024   100230     RECORDING MODE IS F
00025   100240     DATA RECORD IS RECORD-1.
00026   100250   01 RECORD-1.
00027   100260     02 FIELD-A PICTURE IS X(20).
00028   100270 FD  FILE-2
00029   100280     LABEL RECORDS ARE OMITTED
00030   100290     BLOCK CONTAINS 5 RECORDS
00031   100300     RECORD CONTAINS 20 CHARACTERS
00032   100310     RECORDING MODE IS F
00033   100320     DATA RECORD IS RECORD-2.
00034   100330   01 RECORD-2.
00035   100340     02 FIELD-A PICTURE IS X(20).
00036   100350 WORKING-STORAGE SECTION.
00037   100360     77 COUNT PICTURE S99 COMP SYNC.
00038   100370     77 NUMBER  PICTURE S99 COMP SYNC.
00039   100375   01 FILLER.
00040 ** 00380     02 ALPHABET PICTURE X(26) VALUE "ABCDEFGHIJKLMNOPQRSTUVWXYZ".
00041   100395     02 ALPHA REDEFINES ALPHABET PICTURE X OCCURS 26 TIMES.
00042   100405     02 DEPENDENTS PICTURE X(26)  VALUE "01234012340123401234012340
00043   100410-    "0".
00044   100420     02 DEPEND REDEFINES DEPENDENTS PICTURE X OCCURS 26 TIMES.
00045   100440   01 WORK-RECORD.
00046   100450     02 NAME-FIELD  PICTURE X.
00047   100460     02 FILLER PICTURE X VALUE SPACE.
00048   100470     02 RECORD-NO PICTURE 9999.
00049   100480     02 FILLER PICTURE X VALUE SPACE.
00050   100490     02 LOCATION PICTURE AAA VALUE "NYC".
00051   100500     02 FILLER PICTURE X VALUE SPACE.
00052   100510     02 NO-OF-DEPENDENTS PICTURE XX.
00053   100520     02 FILLER PICTURE X(7) VALUE SPACES.
00054   100530 PROCEDURE DIVISION.
00055   100540 BEGIN. READY TRACE.
00056   100550     NOTE THAT THE FOLLOWING OPENS THE OUTPUT FILE TO BE CREATED
00057   100560     AND INITIALIZES COUNTERS.
00058   100570 STEP-1. OPEN OUTPUT FILE-1. MOVE ZERO TO COUNT NUMBER.
00059   100580     NOTE THAT THE FOLLOWING CREATES INTERNALLY THE RECORDS TO BE
00060   100590     CONTAINED IN THE FILE, WRITES THEM ON TAPE, AND DISPLAYS
00061   100600     THEM ON THE CONSOLE.
00062   100610 STEP-2. ADD 1 TO COUNT, ADD 1 TO NUMBER, MOVE ALPHA (COUNT) TO
00063   100620     NAME-FIELD.
00064   100630     MOVE DEPEND (COUNT) TO NO-OF-DEPENDENTS.
00065   100640     MOVE NUMBER TO RECORD-NO.
00066   100650 STEP-3. DISPLAY WORK-RECORD UPON CONSOLE. WRITE RECORD-1 FROM
00067   100660     WORK-RECORD.
00068   100670 STEP-4. PERFORM STEP-2 THRU STEP-3 UNTIL COUNT IS EQUAL TO 26.
00069   100680     NOTE THAT THE FOLLOWING CLOSES OUTPUT AND REOPENS IT AS
00070   100690     INPUT.
00071   100700 STEP-5. CLOSE FILE-1. OPEN INPUT FILE-2.
00072   100710     NOTE THAT THE FOLLOWING READS BACK THE FILE AND SINGLES OUT
00073   100720     EMPLOYEES WITH NO DEPENDENTS.
00074   100730 STEP-6. READ FILE-2 RECORD INTO WORK-RECORD AT END GO TO STEP-8.
00075   100740 STEP-7. IF NO-OF-DEPENDENTS IS EQUAL TO "0" MOVE "Z" TO
00076   100750     NO-OF-DEPENDENTS. EXHIBIT NAMED WORK-RECORD. GO TO
00077   100760     STEP-6.
00078   100770 STEP-8. CLOSE FILE-2.
00079   100780     STOP RUN.
```

```
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=1-167     01  RECORD-1                  BL=1    000     DNM=1-167     DS 0CL20     GROUP
DNM=1-188     02  FIELD-A                   BL=1    000     DNM=1-188     DS 20C       DISP
DNM=1-205     FD  FILE-2                   DCB=02          DNM=1-205                  QSAM                  F
DNM=1-224     01  RECORD-2                  BL=2    000     DNM=1-224     DS 0CL20     GROUP
DNM=1-245     02  FIELD-A                   BL=2    000     DNM=1-245     DS 20C       DISP
DNM=1-265     77  COUNT                     BL=3    000     DNM=1-265     DS 1H        COMP
DNM=1-280     77  NUMBER                    BL=3    002     DNM=1-280     DS 1H        COMP
DNM=1-296     01  FILLER                    BL=3    008     DNM=1-296     DS 0CL52     GROUP
DNM=1-315     02  ALPHABET                  BL=3    008     DNM=1-315     DS 26C       DISP
DNM=1-333     02  ALPHA                     BL=3    008     DNM=1-333     DS 1C        DISP       R  O
DNM=1-351     02  DEPENDENTS                BL=3    022     DNM=1-351     DS 26C       DISP
DNM=1-371     02  DEPEND                    BL=3    022     DNM=1-371     DS 1C        DISP       F  O
DNM=1-387     01  WORK-RECORD               BL=3    040     DNM=1-387     DS 0CL20     GROUP
DNM=1-411     02  NAME-FIELD                BL=3    040     DNM=1-411     DS 1C        DISP
DNM=1-431     02  FILLER                    BL=3    041     DNM=1-431     DS 1C        DISP
DNM=1-450     02  RECORD-NO                 BL=3    042     DNM=1-450     DS 4C        DISP-NM
DNM=1-469     02  FILLER                    BL=3    046     DNM=1-469     DS 1C        DISP
DNM=1-488     02  LOCATION                  BL=3    047     DNM=1-488     DS 3C        DISP
DNM=2-000     02  FILLER                    BL=3    04A     DNM=2-000     DS 1C        DISP
DNM=2-019     02  NO-OF-DEPENDENTS          BL=3    04B     DNM=2-019     DS 2C        DISP
DNM=2-045     02  FILLER                    BL=3    04D     DNM=2-045     DS 7C        DISP
```

MEMORY MAP

```
TGT                          00230

SAVE AREA                    00230
SWITCH                       00278
TALLY                        0027C
SORT SAVE                    00280
ENTRY-SAVE                   00284
SORT CORE SIZE               00288
RET CODE                     0028C
SORT RET                     0028E
WORKING CELLS                00290
SORT FILE SIZE               003C0
SORT MODE SIZE               003C4
PGT-VN TBL                   003C8
TGT-VN TBL                   003CC
VCONPTR                      003D0
LENGTH OF VN TBL             003D4
LABEL RET                    003D6
UNUSED                       003D7
INIT1 ADCON                  003E0
UNUSED                       003E4
OVERFLOW CELLS               003EC
BL CELLS                     003EC
DECBADR CELLS                003F8
TEMP STORAGE                 003F8
TEMP STORAGE-2               00400
TEMP STORAGE-3               00430
TEMP STORAGE-4               00430
BIL CELLS                    00430
VLC CELLS                    00438
SBL CELLS                    00438
INDEX CELLS                  00438
SUBADR CELLS                 00438
ONCTL CELLS                  00438
PFMCTL CELLS                 00438
PFMSAV CELLS                 00438
VN CELLS                     0043C
SAVE AREA =2                 00440
SAVE AREA =3                 00440
XSASW CELLS                  00448
XSA CELLS                    00448
PARAM CELLS                  00448
RPTSAV AREA                  0044C
CHECKPT CTR                  0044C
VCON TBL                     00450
```

LITERAL POOL (HEX)

```
004A8 (LIT+0)     004805EF  00228000  C9D2C6F9  F9F9C940  F4D5F2E4  C3C3C5E2
004C0 (LIT+24)    E2C6E4D3  40D6D7C5  D540C6D6  D9400000  00000000  00000200
004D8 (LIT+48)    40200001  00000001  001A4800  F0E90000  C0000000
```

DISPLAY LITERALS (BCD)

```
004EC (LTL+68)    'WORK-RECORD'
```

```
                PGT                     00458

              OVERFLOW CELLS            00458
              VIRTUAL CELLS             00458
              PROCEDURE NAME CELLS      00464
              GENERATED NAME CELLS      00478
              DCB ADDRESS CELLS         00498
              VNI CELLS                 004A0
              LITERALS                  004A8
              DISPLAY LITERALS          004EC


 REGISTER ASSIGNMENT

   REG 6    BL =3
   REG 7    BL =1
   REG 8    BL =2

55      *BEGIN
                0004F8                          START   EQU   *
                0004F8  58 F0 C 004                     L     15,004(0,12)          V(IIBODSP0)
                0004FC  05 1F                           BALR  1,15
                0004FE  000140                          DC    X'000140'
                000501  04F5F5404040                    DC    X'04F5F5404040'
55       READY  000508  96 40 D 048                     OI    048(13),X'40'         SWT+0
58      *STEP-1
                00050C  58 F0 C 004                     L     15,004(0,12)          V(IIBODSP0)
                000510  05 1F                           BALR  1,15
                000512  000140                          DC    X'000140'
                000515  04F5F8404040                    DC    X'04F5F8404040'
58       OPEN   00051C  58 10 C 040                     L     1,040(0,12)           DCB=1
                000520  D2 01 1 032 C 050               MVC   032(2,1),050(12)                      LIT+0
                000526  D2 01 1 060 C 052               MVC   060(2,1),052(12)                      LIT+2
                00052C  50 10 D 210                     ST    1,210(0,13)           SAV3
                000530  92 8F D 210                     MVI   210(13),X'8F'         SAV3
                000534  41 10 D 210                     LA    1,210(0,13)           SAV3
                000538  0A 13                           SVC   19
                00053A  58 10 C 040                     L     1,040(0,12)           DCB=1
                00053E  58 20 C 034                     L     2,034(0,12)           GN=06
                000542  91 10 1 030                     TM    030(1),X'10'
                000546  07 12                           BCR   1,2
                000548  D2 2D D 1D0 C 054               MVC   1D0(46,13),054(12)    TS2=1           LIT+4
                00054E  D2 07 D 1F2 1 028               MVC   1F2(8,13),028(1)      TS2=35
                000554  41 10 D 1D0                     LA    1,1D0(0,13)           TS2=1
                000558  0A 23                           SVC   35
                00055A                          GN=06   EQU   *
                00055A  58 10 C 040                     L     1,040(0,12)           DCB=1
                00055E  18 21                           LR    2,1
                000560  58 F0 1 030                     L     15,030(0,1)
                000564  05 FF                           BALR  14,15
                000566  50 10 D 1BC                     ST    1,1BC(0,13)           BI =1
                00056A  58 70 D 1BC                     L     7,1BC(0,13)           BI =1
58       MOVE   00056E  D2 01 6 000 C 076               MVC   000(2,6),076(12)      DNM=1-265       LIT+38
                000574  D2 01 6 002 C 076               MVC   002(2,6),076(12)      DNM=1-280       LIT+38
62      *STEP-2
                00057A                          PN=01   EQU   *
                00057A  58 F0 C 004                     L     15,004(0,12)          V(IIBODSP0)
                00057E  05 1F                           BALR  1,15
                000580  000140                          DC    X'000140'
                000583  04F6F2404040                    DC    X'04F6F2404040'
62       ADD    00058A  48 30 C 082                     LH    3,082(0,12)           LIT+50
                00058E  4A 30 6 000                     AH    3,000(0,6)            DNM=1-265
                000592  40 30 6 000                     STH   3,000(0,6)            DNM=1-265
62       ADD    000596  48 30 C 082                     LH    3,082(0,12)           LIT+50
                00059A  4A 30 6 002                     AH    3,002(0,6)            DNM=1-280
                00059E  40 30 6 002                     STH   3,002(0,6)            DNM=1-280
62       MOVE   0005A2  41 40 6 008                     LA    4,008(0,6)            DNM=1-333
                0005A6  48 20 6 000                     LH    2,000(0,6)            DNM=1-265
                0005AA  4C 20 C 082                     MH    2,082(0,12)           LIT+50
                0005AE  1A 42                           AR    4,2
                0005B0  5B 40 C 084                     S     4,084(0,12)           LIT+52
                0005B4  D2 00 6 040 4 000               MVC   040(1,6),000(4)       DNM=1-411
```

```
         MOVE     0005BA  41 20 6 022                LA    2,022(0,6)            DNM=1-371
                  0005BE  48 10 6 000                LH    1,000(0,6)            DNM=1-265
                  0005C2  4C 10 C 082                MH    1,082(0,12)           LIT+50
                  0005C6  1A 21                      AR    2,1
                  0005C8  5B 20 C 084                S     2,084(0,12)           LIT+52
                  0005CC  D2 00 6 04B 2 000          MVC   04B(1,6),000(2)       DNM=2-19
                  0005D2  92 40 6 04C                MVI   04C(6),X'40'          DNM=2-19+1
65       MOVE     0005D6  48 30 6 002                LH    3,002(0,6)            DNM=1-280
                  0005DA  4E 30 D 1C8                CVD   3,1C8(0,13)           TS=01
                  0005DE  F3 31 6 042 D 1CE          UNPK  042(4,6),1CE(2,13)    DNM=1-450       TS=07
                  0005E4  96 F0 6 045                OI    045(6),X'F0'          DNM=1-450+3
66       *STEP-3
                  0005E8  58 F0 C 004                L     15,004(0,12)          V(ILBODSP0)
                  0005EC  05 1F                      BALR  1,15
                  0005EE  000140                     DC    X'000140'
                  0005F1  04F6F64C4040               DC    X'04F6F6404040'
66       DISPLAY  0005F8  58 F0 C 004                L     15,004(0,12)          V(ILBODSP0)
                  0005FC  05 1F                      BALR  1,15
                  0005FE  0002                       DC    X'0002'
                  000600  00                         DC    X'00'
                  000601  000014                     DC    X'000014'
                  000604  0D0001C4                   DC    X'0D0001C4'           BL =3
                  000608  0040                       DC    X'0040'
                  00060A  FFFF                       DC    X'FFFF'
66       WRITE    00060C  D2 13 7 000 6 040          MVC   000(20,7),040(6)      DNM=1-167       DNM=1-387
                  000612  58 10 C 040                L     1,040(0,12)           DCB=1
                  000616  18 21                      LR    2,1
                  000618  58 10 C 040                L     1,040(0,12)           DCB=1
                  00061C  58 00 1 04C                L     0,04C(0,1)
                  000620  58 F0 1 030                L     15,030(0,1)
                  000624  44 00 1 060                EX    0,060(0,1)
                  000628  50 10 D 1BC                ST    1,1BC(0,13)           BL =1
                  00062C  58 70 D 1BC                L     7,1BC(0,13)           BL =1
                  000630                       GN=01 EQU   *
                  000630  58 10 D 20C                L     1,20C(0,13)           VN=01
                  000634  07 F1                      BCR   15,1
68       *STEP-4
                  000636                       PN=02 EQU   *
                  000636  58 F0 C 004                L     15,004(0,12)          V(ILBODSP0)
                  00063A  05 1F                      BALR  1,15
                  00063C  000140                     DC    X'000140'
                  00063F  04F6F84C4040               DC    X'04F6F8404040'
68       PERFORM  000646  58 00 D 20C                L     0,20C(0,13)           VN=01
                  00064A  50 00 D 208                ST    0,208(0,13)           PSV=1
                  00064E  58 00 C 024                L     0,024(0,12)           GN=02
                  000652  50 00 D 20C                ST    0,20C(0,13)           VN=01
                  000656                       GN=02 EQU   *
                  000656  48 30 6 000                LH    3,000(0,6)            DNM=1-265
                  00065A  49 30 C 088                CH    3,088(0,12)           LIT+56
                  00065E  58 F0 C 028                L     15,028(0,12)          GN=03
                  000662  07 8F                      BCR   8,15
                  000664  58 10 C 00C                L     1,00C(0,12)           PN=01
                  000668  07 F1                      BCR   15,1
                  00066A                       GN=03 EQU   *
                  00066A  58 00 D 208                L     0,208(0,13)           PSV=1
                  00066E  50 00 D 20C                ST    0,20C(0,13)           VN=01
71       *STEP-5
                  000672  58 F0 C 004                L     15,004(0,12)          V(ILBODSP0)
                  000676  05 1F                      BALR  1,15
                  000678  000140                     DC    X'000140'
                  00067B  04F7F14C4040               DC    X'04F7F1404040'
71       CLOSE    000682  58 10 C 040                L     1,040(0,12)           DCB=1
                  000686  58 30 1 02C                L     3,02C(0,1)
                  00068A  91 0F 3 00C                TM    00C(3),X'0F'
                  00068E  05 50                      BALR  5,0
                  000690  47 F0 5 010                BC    14,010(0,5)
                  000694  58 20 1 04C                L     2,04C(0,1)
                  000698  4B 20 1 052                SH    2,052(0,1)
                  00069C  50 20 1 04C                ST    2,04C(0,1)
                  0006A0  58 10 C 040                L     1,040(0,12)           DCB=1
                  0006A4  50 10 D 210                ST    1,210(0,13)           SAV3
                  0006A8  92 C0 D 210                MVI   210(13),X'C0'         SAV3
                  0006AC  41 10 D 210                LA    1,210(0,13)           SAV3
```

```
                0006B0  0A 14                          SVC   20
                0006B2  58 20 C 040                    L     2,040(0,12)          DCB=1
                0006B6  58 10 2 014                    L     1,014(0,2)
                0006BA  96 01 2 017                    OI    017(2),X'01'
                0006BE  48 40 1 004                    LH    4,004(0,1)
                0006C2  4C 40 1 006                    MH    4,006(0,1)
                0006C6  41 00 4 008                    LA    0,008(0,4)
                0006CA  41 10 1 CC0                    LA    1,000(0,1)
                0006CE  0A 0A                          SVC   10
71      OPEN    0006D0  58 10 C 044                    L     1,044(0,12)          DCB=2
                0006D4  D2 01 1 032 C 08A              MVC   032(2,1),08A(12)                    LIT+58
                0006DA  50 10 D 210                    ST    1,210(0,13)          SAV3
                0006DE  92 80 D 210                    MVI   210(13),X'80'        SAV3
                0006E2  41 10 D 210                    LA    1,210(0,13)          SAV3
                0006E6  0A 13                          SVC   19
                0006E8  58 10 C 044                    L     1,044(0,12)          DCB=2
                0006EC  58 20 C 038                    L     2,038(0,12)          GN=07
                0006F0  91 10 1 030                    TM    030(1),X'10'
                0006F4  07 12                          BCR   1,2
                0006F6  D2 2D D 1D0 C 054              MVC   1D0(46,13),054(12)   TS2=1          LIT+4
                0006FC  D2 07 D 1F2 1 028              MVC   1F2(8,13),028(1)     TS2=35
                000702  41 10 D 1D0                    LA    1,1D0(0,13)          TS2=1
                000706  0A 23                          SVC   35
                000708                         GN=07   EQU   *
74      *STEP-6
                000708                         PN=03   EQU   *
                000708  58 F0 C 004                    L     15,004(0,12)         V(ILBODSP0)
                00070C  05 1F                          BALR  1,15
                00070E  000140                         DC    X'000140'
                000711  04F7F44C4040                   DC    X'04F7F4404040'
74      READ    000718  58 10 C 044                    L     1,044(0,12)          DCB=2
                00071C  18 21                          LR    2,1
                00071E  D2 02 2 021 C 02D              MVC   021(3,2),02D(12)                    GN=04+1
                000724  58 F0 1 030                    L     15,030(0,1)
                000728  05 EF                          BALR  14,15
                00072A  50 10 D 1C0                    ST    1,1C0(0,13)          BL =2
                00072E  58 80 D 1C0                    L     8,1C0(0,13)          BL =2
                000732  D2 13 6 040 8 000              MVC   040(20,6),000(8)     DNM=1-387      DNM=1-224
                000738  58 50 C 018                    L     5,018(0,12)          PN=04
                00073C  07 F5                          BCR   15,5
74      GO      00073E                         GN=04   EQU   *
                00073E  58 10 C 01C                    L     1,01C(0,12)          PN=05
                000742  07 F1                          BCR   15,1
75      *STEP-7
                000744                         PN=04   EQU   *
                000744  58 F0 C 004                    L     15,004(0,12)         V(IBODSP0)
                000748  05 1F                          BALR  1,15
                00074A  000140                         DC    X'000140'
                00074D  04F7F5404040                   DC    X'04F7F5404040'
75      IF      000754  58 10 C 03C                    L     1,03C(0,12)          GN=08
                000758  58 20 C 030                    L     2,030(0,12)          GN=05
                00075C  D5 00 C 08C 6 04B              CLC   08C(1,12),04B(6)     LIT+60         DNM=2-19
                000762  07 72                          BCR   7,2
                000764  95 40 6 04C                    CLI   04C(6),X'40'         DNM=2-19+1
                000768  07 72                          BCR   7,2
75      MOVE    00076A                         GN=08   EQU   *
                00076A  D2 00 6 04B C 08D              MVC   04B(1,6),08D(12)     DNM=2-19       LIT+61
                000770  92 40 6 04C                    MVI   04C(6),X'40'         DNM=2-19+1
76      EXHIBIT 000774                         GN=05   EQU   *
                000774  58 10 C 090                    L     1,090(0,12)          LIT+64
                000778  50 10 D 218                    ST    1,218(0,13)          PRM=1
                00077C  41 20 D 218                    LA    2,218(0,13)          PRM=1
                000780  58 F0 C 004                    L     15,004(0,12)         V(ILBODSP0)
                000784  05 1F                          BALR  1,15
                000786  8001                          DC    X'8001'
                000788  10                            DC    X'10'
                000789  00000B                        DC    X'00000B'
                00078C  0C000094                       DC    X'0C000094'          LIT+68
                000790  0000                          DC    X'0000'
                000792  00                            DC    X'00'
                000793  000014                        DC    X'000014'
                000796  0D0001C4                       DC    X'0D0001C4'          BL =3
                00079A  0040                          DC    X'0040'
                00079C  FFFF                          DC    X'FFFF'
```

```
76       GO       00079E  58 10 C 014                 I     1,014(0,12)          PN=03
                  0007A2  07 F1                       BCR   15,1
78       *STEP-8
                  0007A4                       PN=05  EQU   *
                  0007A4  58 F0 C 004                 I     15,004(0,12)         V(ILBODSP0)
                  0007A8  05 1F                       BALR  1,15
                  0007AA  000140                      DC    X'000140'
                  0007AD  04F7F8404040                DC    X'04F7F8404040'
78       CLOSE    0007B4  58 10 C 044                 L     1,044(0,12)          DCB=2
                  0007B8  58 30 1 02C                 L     3,02C(0,1)
                  0007BC  91 0F 3 00C                 TM    00C(3),X'0F'
                  0007C0  05 50                       BALR  5,0
                  0007C2  47 F0 5 010                 BC    14,010(0,5)
                  0007C6  58 20 1 04C                 L     2,04C(0,1)
                  0007CA  4B 20 1 052                 SH    2,052(0,1)
                  0007CE  50 20 1 04C                 ST    2,04C(0,1)
                  0007D2  58 10 C 044                 L     1,044(0,12)          DCB=2
                  0007D6  50 10 D 210                 ST    1,210(0,13)          SAV3
                  0007DA  92 C0 D 210                 MVI   210(13),X'C0'        SAV3
                  0007DE  41 10 D 210                 LA    1,210(0,13)          SAV3
                  0007E2  0A 14                       SVC   20
                  0007E4  58 20 C 044                 L     2,044(0,12)          DCB=2
                  0007E8  58 10 2 014                 L     1,014(0,2)
                  0007EC  96 01 2 017                 OI    017(2),X'01'
                  0007F0  48 40 1 004                 LH    4,004(0,1)
                  0007F4  4C 40 1 006                 MH    4,006(0,1)
                  0007F8  41 00 4 C08                 LA    0,008(0,4)
                  0007FC  41 10 1 000                 LA    1,000(0,1)
                  000800  0A 0A                       SVC   10
79       STOP     000802  58 F0 C 008                 L     15,008(0,12)         V(ILBOSTP1)
                  000806  07 FF                       BCR   15,15
                  000808  50 D0 5 0C8          INIT2  ST    13,008(0,5)
                  00080C  50 50 D 004                 ST    5,004(0,13)
                  000810  50 E0 D 054                 ST    14,054(0,13)
                  000814  58 F0 C 000                 L     15,000(0,12)         VIR=1
                  000818  05 EF                       BALR  14,15
                  00081A  12 00                       LTR   0,0
                  00081C  07 89                       BCR   8,9
                  00081E  96 10 D 048                 OI    048(13),X'10'        SWT+0
                  000822  05 F0                INIT3  BALR  15,0
                  000824  91 20 D 048                 TM    048(13),X'20'        SWT+0
                  000828  47 F0 F 016                 BC    14,016(0,15)
                  00082C  98 2D B 040                 LM    2,13,040(11)
                  000830  58 00 B 038                 L     0,038(0,11)
                  000834  58 E0 D 054                 L     14,054(0,13)
                  000838  07 FF                       BCR   15,14
                  00083A  96 20 D 048                 OI    048(13),X'20'        SWT+0
                  00083E  41 60 0 004                 LA    6,004(0,0)
                  000842  41 10 C 00C                 LA    1,00C(0,12)          PN=01
                  000846  41 70 C 050                 LA    7,050(0,12)          LIT+0
                  00084A  06 70                       BCTR  7,0
                  00084C  05 50                       BALR  5,0
                  00084E  58 40 1 000                 L     4,000(0,1)
                  000852  1E 4B                       ALR   4,11
                  000854  50 40 1 000                 ST    4,000(0,1)
                  000858  87 16 5 000                 BXLE  1,6,000(5)
                  00085C  41 80 D 1BC                 LA    8,1BC(0,13)          OVF=1
                  000860  41 70 D 1C7                 LA    7,1C7(0,13)          TS=01-1
                  000864  05 10                       BALR  1,0
                  000866  58 00 8 000                 L     0,000(0,8)
                  00086A  1E 0B                       ALR   0,11
                  00086C  50 00 8 000                 ST    0,000(0,8)
                  000870  87 86 1 000                 BXLE  8,6,000(1)
                  000874  D2 03 D 20C C 048           MVC   20C(4,13),048(12)    VN=01                VNI=1
                  00087A  58 60 D 1C4                 L     6,1C4(0,13)          EI =3
                  00087E  58 70 D 1BC                 L     7,1BC(0,13)          RL =1
                  000882  58 80 D 1C0                 L     8,1C0(0,13)          RL =2
                  000886  58 E0 D 054                 L     14,054(0,13)
                  00088A  07 FF                       BCR   15,14
```

```
000000  07 00              INIT1    BCR   0,0
000002  90 EC D 00C                 STM   14,12,00C(13)
000006  18 5D                      LR    5,13
000008  05 F0                      BALR  15,0
00000A  98 9F F 012                 LM    9,15,012(15)
00000E  07 FF                      BCR   15,15
000010  96 02 1 034                OI    034(1),X'02'
000014  07 FE                      BCR   15,14
000016  41 F0 0 001                LA    15,001(0,0)
00001A  07 FE                      BCR   15,14
00001C  00000822                   ADCON L4(INIT3)
000020  00000000                   ADCON L4(INIT1)
000024  00000000                   ADCON L4(INIT1)
000028  00000458                   ADCON L4(PGT)
00002C  00000230                   ADCON L4(TGT)
000030  000004F8                   ADCON L4(START)
000034  00000808                   ADCON L4(INIT2)
000038                             DS    15F
000074  FFFFFFFF                   DC    X'FFFFFFFF'
000078  E3C5E2E3D9E4D540           DC    X'E3C5E2E3D9E4D540'
```

```
*STATISTICS*     SOURCE RECORDS =    79     DATA DIVISION STATEMENTS =    22     PROCEDURE DIVISION STATEMENTS =    21
*OPTIONS IN EFFECT*     SIZE =   81920  BUF =    2768  LINECNT = 57  SPACE1, FLAGW,   SEQ,   SOURCE
*OPTIONS IN EFFECT*      DMAP,   PMAP, NOCLIST, NOSUPMAP,   XREF,   LOAD, NODECK, QUOTE, NOTRUNC, LIB, VERB, ZWB
```

CROSS-REFERENCE DICTIONARY

DATA NAMES	DEFN	REFERENCE					
FILE-1	00016	00058	00058	00066	00066	00071	00071
RECORD-1	00026	00066					
FIELD-A	00027						
FILE-2	00017	00071	00071	00074	00074	00078	00078
RECORD-2	00034	00074					
FIELD-A	00035						
COUNT	00037	00058	00062	00062	00062	00064	00068
NUMBER	00038	00058	00062	00062	00065		
FILLER	00039						
ALPHABET	00040						
ALPHA	00041	00062					
DEPENDENTS	00042						
DEPEND	00044	00064					
WORK-RECORD	00045	00066	00066	00074	00076		
NAME-FIELD	00046	00062					
FILLER	00047						
RECORD-NO	00048	00065	00065				
FILLER	00049						
LOCATION	00050						
FILLER	00051						
NO-OF-DEPENDENTS	00052	00064	00064	00075	00075	00075	00075
FILLER	00053						

PROCEDURE NAMES	DEFN	REFERENCE
BEGIN	00055	
STEP-1	00058	
STEP-2	00062	00068
STEP-3	00066	
STEP-4	00068	
STEP-5	00071	
STEP-6	00074	00076
STEP-7	00075	
STEP-8	00078	00074

```
CARD   ERROR MESSAGE

       IKF1100I-W      1 SEQUENCE ERROR IN SOURCE PROGRAM.
```

```
IEF285I   PRODTEST                                        PASSED
IEF285I   VOL SER NOS= USAS
IEF285I   SYS70027.T092817.RP002.TEST.UT1                 DELETED
IEF285I   VOL SER NOS= 231400.
IEF285I   SYS70027.T092817.RP002.TEST.UT2                 DELETED
IEF285I   VOL SER NOS= 231100.
IEF285I   SYS70027.T092817.RP002.TEST.UT3                 DELETED
IEF285I   VOL SER NOS= 231401.
IEF285I   SYS70027.T092817.RP002.TEST.UT4                 DELETED
IEF285I   VOL SER NOS= 231100.
IEF285I   SYS70027.T092817.RP002.TEST.PNCH                PASSED
IEF285I   VOL SER NOS= 231400.
IEF285I   SYSOUT                                          SYSOUT
IEF285I   VOL SER NOS=
STEP STEP1    TERMINATED. TIME 00.02 HR.HDRTH/HR * 00.01.35.22 HR.MIN.SEC.HDRTH/SEC*DATE 70.027
//STEP2  EXEC  PGM=IEWL,PARM='XREF',REGION=96K
//SYSUT1 DD    DSNAME=&&UT1,UNIT=SYSDA,SPACE=(TRK,(100,10))
//SYSLMOD DD   DSNAME=&&GJOB(GO),UNIT=SYSDA,SPACE=(TRK,(100,10,1)),     X
//             DISP=(NEW,PASS)
//SYSLIB DD DSN=SYS1.COBULIB,UNIT=2314,VOL=SER=USAS,DISP=SHR
//SYSPRINT DD  SYSOUT=A
//SYSLIN   DD  DSNAME=&&PNCH,DISP=(OLD,DELETE)
IEF236I ALLOC. FOR TEST     STEP2
IEF237I 234   ALLOCATED TO JOBLIB
IEF237I 190   ALLOCATED TO SYSUT1
IEF237I 190   ALLOCATED TO SYSLMOD
IEF237I 234   ALLOCATED TO SYSLIB
IEF237I 230   ALLOCATED TO SYSLIN
```

```
F88-LEVEL LINKAGE EDITOR OPTIONS SPECIFIED XREF
          VARIABLE OPTIONS USED - SIZE=(153600,51200)              DEFAULT OPTION(S) USED


                                        CROSS REFERENCE TABLE

  CONTROL SECTION                       ENTRY

    NAME    ORIGIN  LENGTH              NAME   LOCATION     NAME   LOCATION     NAME   LOCATION     NAME   LOCATION

  IFSTRUN       00     88C
  IILBODSP0*   890     69A
  ILBOSTP0*    F30     11C
                                       ILBOSTP1    1016    PDTSZE      104B


  LOCATION  REFERS TO SYMBOL  IN CONTROL SECTION            LOCATION  REFERS TO SYMBOL  IN CONTROL SECTION

     458            ILBOSTP0        ILBOSTP0                   45C            ILBODSP0        ILBODSP0
     460            ILBOSTP1        ILBOSTP0
 ENTRY ADDRESS       00
 TOTAL LENGTH      1050

****GO        DOES NOT EXIST BUT HAS BEEN ADDED TO DATA SET


IEF285I   PROD.TEST                                       PASSED
IEF285I   VOL SER NOS= USAS  .
IEF285I   SYS70027.T092817.RP002.TEST.UT1                 DELETED
IEF285I   VOL SER NOS= 231100.
IEF285I   SYS70027.T092817.RP002.TEST.GJOB                PASSED
IEF285I   VOL SER NOS= 231100.
IEF285I   SYS1.COBULIB                                    KEPT
IEF285I   VOL SER NOS= USAS  .
IEF285I   SYSOUT                                          SYSOUT
IEF285I   VOL SER NOS=       .
IEF285I   SYS70027.T092817.RP002.TEST.PNCH                DELETED
IEF285I   VOL SER NOS= 231400.
STEP STEP2    TERMINATED. TIME 00.00 HR.HDRTH/HR * 00.00.23.58 HR.MIN.SEC.HDRTH/SEC*DATE 70.027
//STEP3  EXEC  PGM=*.STEP2.SYSLMOD
//SYSOUT DD    SYSOUT=A
//SYSUDUMP DD SYSOUT=A
//SAMPLE DD    UNIT=2400,LABEL=(,NL)
IEF236I ALLOC. FOR TEST     STEP3
IEF237I 234   ALLOCATED TO JOBLIB
IEF237I 190   ALLOCATED TO PGM=*.DD
IEF237I 183   ALLOCATED TO SAMPLE

58
62
66
68
62
66
62
66
 .
 .
 .
```

(Repeat 21 times)

```
 .
 .
 .
62
66
62
66
71
74
75
```

```
WORK-RECORD = A 0001 NYC Z
74
75
WORK-RECORD = B 0002 NYC 1
74
75
WORK-RECORD = C 0003 NYC 2
74
75
WORK-RECORD = D 0004 NYC 3
74
75
WORK-RECORD = E 0005 NYC 4
74
75
WORK-RECORD = F 0006 NYC Z
74
75
WORK-RECORD = G 0007 NYC 1
74
75
WORK-RECORD = H 0008 NYC 2
74
75
WORK-RECORD = I 0009 NYC 3
74
75
WORK-RECORD = J 0010 NYC 4
74
75
WORK-RECORD = K 0011 NYC Z
74
75
WORK-RECORD = L 0012 NYC 1
74
75
WORK-RECORD = M 0013 NYC 2
74
75
WORK-RECORD = N 0014 NYC 3
74
75
WORK-RECORD = O 0015 NYC 4
74
75
WORK-RECORD = P 0016 NYC Z
74
75
WORK-RECORD = Q 0017 NYC 1
74
75
WORK-RECORD = R 0018 NYC 2
74
75
WORK-RECORD = S 0019 NYC 3
74
75
WORK-RECORD = T 0020 NYC 4
74
75
```

```
WORK-RECORD = U 0021 NYC 1
74
75
WORK-RECORD = V 0022 NYC 1
74
75
WORK-RECORD = W 0023 NYC 2
74
75
WORK-RECORD = X 0024 NYC 3
74
75
WORK-RECORD = Y 0025 NYC 4
74
75
WORK-RECORD = Z 0026 NYC Z
74
78
```

```
IEF285I   PRODTEST                                        PASSED
IEF285I   VOL SER NOS= USAS
IEF285I   SYS70027.T092817.RP002.TEST.GJOB                PASSED
IEF285I   VOL SER NOS= 231100.
IEF285I   SYSOUT                                          SYSOUT
IEF285I   VOL SER NOS=        .
IEF285I   SYS70027.T092817.RP002.TEST.R0000006            DELETED
IEF285I   VOL SER NOS= L00101.
STEP STEP3    TERMINATED. TIME 00.02 HR.HDRTH/HR * 00.01.15.77 HR.MIN.SEC.HDRTH/SEC*DATE 70.027
IEF285I   PRODTEST                                        KEPT
IEF285I   VOL SER NOS= USAS  .
IEF285I   SYS70027.T092817.RP002.TEST.GJOB                DELETED
IEF285I   VOL SER NOS= 231100.
JOB  TEST     TERMINATED. TIME 00.05 HR.HDRTH/HR * 00.03.19.57 HR.MIN.SEC.HDRTH/SEC*DATE 70.027* START 09.47 END 09.52
******    THIS JOB WAS RUN ON MODEL 50D    ******
```

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:

- Checkpoint feature
- Conversion routines
- Segmentation feature
- Arithmetic verb routines
- Input/output verb routines
- Sort feature interface routines
- Other verb routines

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.

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: (1) external or zoned decimal, (2) external floating-point, (3) sterling display, (4) numeric edited, and (5) sterling report. The three internal formats are: (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 Table 28.

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 Table 29.

COBOL LIBRARY INPUT/OUTPUT SUBROUTINES

The input/output subroutines are for the verbs DISPLAY (TRACE and EXHIBIT), ACCEPT, WRITE and CLOSE, I/O errors, printer spacing, 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.

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.

If end-of-file is reached on SYSIN, no further activity is performed on the file, even if further ACCEPT statements are issued.

Table 28. Functions of COBOL Library Conversion Subroutine (Part 1 of 2)

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

[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.

Table 28. Functions of COBOL Library Conversion Subroutines (Part 2 of 2)

Subroutine Name and Entry Points	Conversion	
	From	To
ILBOIBD0	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
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

[3]This entry point is used for calls from other COBOL library subroutines.

Table 29. Functions of COBOL Library Arithmetic Subroutines

Subroutine Name	Function
ILBOXMU0	Internal Decimal Multiplication (30 digits * 30 digits = 60 digits)
ILBOXDI0	Internal Decimal Division (60 digits/30 digits = 60 digits)
ILBOSPR0	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.

BSAM 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 Subroutine (ILBOSAMR0)

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.

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	Standard Sequential Files
ILBOERR2	Direct and Relative Files Accessed Sequentially
ILBOERR3	Indexed Files Accessed Sequentially
ILBOERR4	Direct and Relative Files Accessed Randomly
ILBOERR5	Indexed Files Accessed Randomly

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.

SORT FEATURE SUBROUTINE (ILBOSRT0)

The ILBOSRT0 routine acts as an interface between the COBOL calling program and the Sort/Merge program.

COBOL LIBRARY SUBROUTINES

There are also COBOL library subroutines for the comparisons, the verbs MOVE and TRANSFORM, and other features of the COBOL language.

COMPARE Subroutine (ILBOVCO0)

The ILBOVCO0 subroutine compares two operands, one or both of which is variable in length. They may exceed 256 bytes.

MOVE Subroutine (ILBOVMO0 and ILBOVMO1)

The MOVE subroutine is used when one or both operands is variable in length. They may exceed 256 bytes. The MOVE 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: ILBOVMO0 (left-justified) and ILBOVMO1 (right-justified).

TRANSFORM Subroutine (ILBOVTR0)

The ILBOVTR0 subroutine translates variable-length items.

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.

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

Segmentation Subroutine (ILBOSGM0)

The ILBOSGM0 subroutine is used to load segments of a program that are not in core storage and to pass control from one segment to the other.

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.

STOP RUN Subroutine (ILBOSTP0)

The ILBOSTP0 subroutine controls exiting from the program and is entered when a program receives initial control. ILBOSTP1 is entered at the termination of a program.

Date Subroutine (ILBODTE0)

The ILBODTE0 subroutine retrieves the time of day and the date from the system and stores the information in the receiving field of the specified MOVE statement.

Compare Figurative Constant Greater Than One Character Subroutine (ILBOIVL0)

The ILBOIVL0 subroutine is used in comparisons of the ALL 'literal' where literal is a figurative constant greater than one character in length.

MOVE Data-name, Literal, or Figurative Constant Subroutine (ILBOANE0)

The ILBOANE0 subroutine moves a data-name, literal, or figurative constant into a right or left adjusted alphanumeric edited field.

MOVE Figurative Constant of More Than One Character Subroutine (ILBOANF0)

The ILBOANF0 subroutine moves a figurative constant of more than one character into a nonnumeric receiving field. The result may be right or left justified.

Checkpoint Subroutine (ILBOCKP0)

The ILBOCKP0 subroutine is used when checkpoints are taken in the program.

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. Tables 30 through 34 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 IBM System/360 Operating System: Supervisor and Data Management Macro Instructions.

Note: The DCB subparameters are discussed under "User Defined Files" in the chapter "User File Processing."

Table 30. Data Control Block Fields for Standard Sequential Files

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)
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	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	
MACRF	Type of macro instruction	OPEN INPUT, READ OPEN OUTPUT, WRITE OPEN I-O, READ, WRITE REWRITE	
OPTCD	Optional service provided by control program		(OPTCD=[W|C|WC|T])
RECFM	Characteristics of records in data set	RECORDING MODE Record description ADVANCING POSITIONING BLOCK CONTAINS APPLY RECORD-OVERFLOW	
SYNAD	Address of error exit routine	Used by compiler for INVALID KEY and USE AFTER ERROR	

Table 31. Data Control Block Fields for Direct and Relative Files Accessed Sequentially

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	
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.

Table 32. Data Control Block Fields for Direct and Relative Files Accessed Randomly

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.

Table 33. Data Control Block Fields for Indexed Files Accessed Sequentially

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	
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	

Table 34. Data Control Block Fields for Indexed Files Accessed Randomly

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	

In general, compilation is faster when:

1. Options in the EXEC statement are specified to:
   a. make more main storage available (the SIZE option)
   b. optimize the space available for buffers (the BUF option)
   c. suppress output (the NOSOURCE, NODECK, NOLOAD, and the SUPMAP options)
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 both MFT and MVT.

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 subparameter 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.

The logical record size for SYSPRINT is 121 bytes. The logical record size for SYSIN, SYSLIN, SYSPUNCH, and SYSLIB is 80 bytes.

Note: For compile, linkage 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. The 44K version of the linkage editor supports input data sets with a blocking factor of up to 40 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 utility data sets 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 on 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 that is 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 for buffers.

Let A represent the 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 2768 bytes.

2. If the SIZE option is specified, but BUF is not, A equals (SIZE - 80K) / 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 (that is, 81920 bytes - 2768 bytes).

4. If BUF is smaller than 2768 bytes (the minimum value), 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. If the computed value is greater than the maximum allowable value, a diagnostic message is printed and compilation is abandoned.

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	121

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 a utility data set is 255 bytes.

The programmer may, at times, want to determine minimum buffer, SIZE, and REGION parameters from the block sizes of input/output data sets.

Assume that the block sizes of the input/output data sets are as follows:

Data Set	Block Size (Bytes)
SYSIN	80
SYSLIN	240
SYSPUNCH	320
SYSLIB	160
SYSPRINT	484

Since B equals two times the block size of SYSPRINT + SYSIN + SYSLIB or two times the block size of SYSPRINT + SYSPUNCH + SYSLIN, whichever is greater, by substituting the block sizes of the appropriate files for this example we have

B = 2(484 + 80 + 160)

or

B = 2(484 + 320 + 240)

The greater value is 2088 bytes, therefore the minimum buffer parameter in this case is B + 1280 or 3368.

Referring to the note in item number 3 above, we can calculate a suitable SIZE parameter as follows:

SIZE - BUF = SIZE - 3368 = 79152

SIZE = 79152 + 3368 = 82520 (Minimum)

The minimum REGION is 6K greater than the SIZE parameter. In this example, the minimum REGION size is approximately 87K.

Minimum values with respect to SIZE do not always yield the most efficient processing. Not every program will compile successfully in minimum core. Additional code may have to be generated in the case where core is limited. The amount of physical I/O might increase because of small utility data set buffers. The above example must be used with this consideration in mind.

A guideline for calculating the SIZE parameter once the buffer parameter is known is to increase the minimum SIZE parameter value by the amount that the buffer parameter has exceeded its minimum. If the SIZE and buffer parameters are known for a particular program and the user wishes to increase his buffer parameter value, the SIZE parameter should be increased by the same amount.

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 LINK macro instruction, i.e., dynamic invocation. Dynamic invocation of COBOL compiled programs can be accomplished through the use of the LINK 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 word 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 "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 (SYSUT1/2/3/4) passed to the compiler must be physical sequential; that is, 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

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 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 "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.

Limitations on the size of a COBOL source program should be considered in relation to the capacities of both the COBOL compiler and the various linkage editors. 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 contiguous 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 are 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 1500 card program in the minimum machine configuration (80K). Within an 80K byte environment, the user should not specify buffer size for the compiler files. 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 SYSOUT for which 120 bytes are allocated. Double buffering will be assumed.

Within this configuration, assuming no REPORT SECTION, the compiler will accept:

- 300 procedure references assuming an average procedure-name length of 12 characters
- 25 OCCURS clauses with the DEPENDING ON option
- Ten files assuming an average of three subordinate record entries
- 400 literals assuming an average of 8 bytes

EFFECTIVE STORAGE CONSIDERATIONS

The amount of core 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 core 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:

1. ADCON Table

   Each entry occupies 8 bytes. This table is not limited to the maximum size of 32,767 bytes. Entries are based on:

   - 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, 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 concerning each RD such as controls, sums, headings, footings, routines to be generated, etc. The contents of the table is increased by the existence of 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 existence of the following items causes entries to be made into tables that impact 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

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 IBM System/360 Operating System: Linkage Editor and Loader, for additional information on linkage editor capacities and processing.

1. All COBOL object programs, with the exception of segmented 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 + number of nonresident segments

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 may not be sorted prior to the linkage editor step.

APPENDIX G: INPUT/OUTPUT ERROR CONDITIONS

This appendix contains a brief summary of input/output (I/O) error conditions for each of the file processing techniques. More detailed information on error conditions can be found in the publications IBM System/360 Operating System: Supervisor and Data Management Macro Instructions, and IBM System/360 Operating System: System Control Blocks.

Standard Sequential, Direct, and Relative File Processing Technique (Sequential Access)

Register 1 contains error bits detailing the exact cause of an error. Conditions causing input/output errors and suggested user responses are as follows:

I/O Error Condition:

1. Input Error
2. Output Error
3. Invalid Request (BSAM only)

Suggested User Response:

For BSAM, display the error message. Processing of the file is limited to CLOSE. For QSAM, display the error message and then execute the EROPT option in the DD statement. Note that the EROPT option gives the user three choices:

ACC - accept the error block and continue processing

SKP - skip to the next block.

ABE - terminate the job.

Direct and Relative File Processing Technique (Random Access)

The DECB contains two error condition bytes at location DECB + 4. Conditions causing input/output errors and suggested user responses are as follows:

I/O Error Condition:

1. Invalid Request
   a. Requested block outside data set.
   b. Attempt to add fixed-length record with key beginning with hexadecimal FF.

Suggested User Response:

Condition caused by invalid key. Processing of the file may be continued.

I/O Error Condition:

1. Space Not Found.
2. Record not found.

Suggested User Response:

Processing of the file may be continued. CLOSE, READ, or REWRITE statements may be executed for the file.

I/O Error Condition:

1. Uncorrectable I/O Error
2. Uncorrectable Error, Not I/O

Suggested User Response:

Processing of the file is limited to CLOSE.

Indexed File Processing Technique (Sequential Access)

The DCB contains two error condition bytes named EXCD1 and EXCD2, at location DCB + 80. Conditions causing I/O errors and suggested user responses are as follows:

I/O Error Condition:

1. Sequence Check
2. Duplicate Record

Suggested User Response:

Condition caused by invalid key. Processing of the file may be continued.

I/O Error Condition:

1. Space Not Found
2. Uncorrectable Output Error
3. Unreachable Block (Input)
4. Unreachable Block (Update)

Suggested User Response:

Processing of the file is limited to CLOSE.

I/O Error Condition:

Uncorrectable Input Error

Suggested User Response:

The user may attempt to bypass the block containing the error. If, in reading the next block, the error does not recur, he may continue processing without closing the file. If the error persists, processing of the file is limited to CLOSE.

Indexed File Processing Technique (Random Access)

The DECB contains an error condition byte at location DECB + 24. Conditions causing I/O errors and suggested user responses are as follows:

I/O Error Condition:

1. Record Not Found
2. Duplicate Record

Suggested User Response:

Condition caused by invalid key. Processing of the file may be continued.

I/O Error Condition:

Space Not Found

Suggested User Response:

Processing of the file may be continued. The record may be written after changing the keys and executing a WRITE statement if a cylinder overflow area is available for the new value of the keys. CLOSE or READ may be executed for the file.

I/O Error Condition:

Invalid Request

Suggested User Response:

Processing of the file is limited to CLOSE.

I/O Error Condition:

1. Uncorrectable I/O Error.
2. Unreachable Block--Index Cannot Be Read.

Suggested User Response:

Processing of the file is limited to CLOSE. The user can try to execute the instruction again. If the error persists, he can close the file or perform file recovery procedures.

Determining Causes of I-O Errors

To determine the cause of an I-O error by means of the information presented in the error declarative, specify the GIVING option of the USE sentence. The system will insert a 136-byte formatted message into data-name-1 for inspection, printing, etc. This is not a COBOL function. The contents and format of this area are entirely system-defined.

The portion of data-name-1 referred to as error description (bytes 91-105) may be inspected, since it contains a short prose description of the error. The exact wording of this description is somewhat Release-dependent, and is contained in a module in SYS1.SVCLIB. A good technique for keeping up to date on the possible contents of this area is to maintain a copy of the microfiche of IGC0006H (up to and including Release 20.0) or IGC0006H, IGC106H, IGC206H, IGC306H, IGC406H, IGC506H, IGC606H, IGC706H, IGC806H, or IGC906H (Release 20.1 and later) for this purpose. Alternatively, IMASPZAP may be used to obtain a current dump of the modules. Inspection of these modules will provide the necessary information.

APPENDIX H: CREATING AND RETRIEVING INDEXED SEQUENTIAL DATA SETS

Indexed data sets 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. DD statements must define the areas in a specific sequence:

1. Index area
2. Prime area
3. Overflow area

This order must be maintained if one of the statements is absent. The first or only DD statement defining the data set can contain a name field. Other statements 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. The remaining DD statement parameters -- DSNAME, UNIT, VOLUME, LABEL, DCB, DISP, SPACE, and 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), 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 password protection (PASSWORD).

DCB: Required. Can be used to complete the data control block if it has not been completed by the processing program. 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), catalog it (CATALG), or 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 IBM System/360 Operating System: Utilities.

SPACE: Required. Space must be requested using either the recommended nonspecific allocation technique or the more restricted absolute track (ABSTR) technique. If more than one DD statement is used to define the data set, all 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 on 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 that defines 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 89 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=2301,DCB=(DSORG=IS,OPTCD=1),      X
//            SPACE=(CYL,10,,CONTIG),DISP=(,KEEP)
//
//           DD  DSNAME=MHB(PRIME),DCB=(DSORG=IS,OPTCD=1),UNIT=(2311,2), X
//            VOLUME=SER=(334,335),DISP=(,KEEP),                        X
//            SPACE=(CYL,25,,CONTIG)
//
//           DD  DSNAME=MHB(OVFLOW),DCB=(DSORG=IS,OPTCD=1),UNIT=2311,    X
//            VOLUME=SER=336,SPACE=(CYL,25,,CONTIG),DISP=(,KEEP)
```

Figure 89. Creating an Indexed Data Set

Table 35. Area Arrangement for Indexed Data Sets

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.

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.

Table 35 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 90 shows how to retrieve the indexed data set created by the illustration in Figure 86.

```
//INPUT    DD  DSNAME=MHB,DCB=DSORG=IS,UNIT=2301,DISP=OLD
//         DD  DSNAME=MHB,DCB=DSORG=IS,UNIT=(2311,3),DISP=OLD,        X
//             VOLUME=SER=(334,335,336)
```

Figure 90. Retrieving an Indexed Data Set

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 91 shows a general job control procedure for a compilation job step. The following cases show how to add to or modify the general procedure to obtain various processing options.

Case 1: Compile Only -- No Object Module 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 Card Reader

Modify the end of the procedure as follows:

```
//SYSIN  DD  *
    (source module)
/*
```

If the DD * convention is used, the SYSIN DD statement must be the last DD statement for the job step, and the source module must follow. If another job step follows the compilation, the EXEC statement for that step follows the /* 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)
```

```
//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,DELETE)
```

Figure 91. General Job Control Procedure for Compilation

Note: If LOAD is not the installation default condition, it must be specified in the PARM parameter of the EXEC statement.

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 follows:

A. Cataloging

```
//SYSLIN   DD  DSNAME=dsname,                    X
//             DISP=({NEW},CATLG),               X
//                   {MOD}
//             VOLUME=(subparms),                X
//             LABEL=(subparms),                 X
//             UNIT={SYSDA},                     X
//                  {SYSSQ}
//            {SPACE   }
//            {SPLIT   } =(subparms)
//            {SUBALLOC}
```

B. Keeping

```
//SYSLIN   DD  DSNAME=dsname,                    X
//             DISP=({NEW},KEEP),                X
//                   {MOD}
//             VOLUME=(subparms),                X
//             LABEL=(subparms),                 X
//             UNIT={SYSDA},                     X
//                  {SYSSQ}
//            {SPACE   }
//            {SPLIT   } =(subparms)
//            {SUBALLOC}
```

C. Adding a Member to an Existing 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 DD card(s), as follows:

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 92 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 Card Reader

Change SYSLIN statement and the end of the procedure as follows:

```
//SYSLIN   DD *
    (object module(s))
/*
```

If the DD * convention is used, the SYSLIN DD statement must be the last DD statement in the job step. If another job step follows the link edit step, the EXEC statement for that job step follows the /* statement.

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 92. 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" 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 93 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

The JOBLIB 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
                 .
                 .
                 .
```

```
//stepname EXEC  PGM=progname
//ddname   DD    (parameters for user-specified data sets)
           .
           .
           .
```

Figure 93. General Job Control Procedure for an Execution-Time Job Step

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. If only the first step executes a load module from the library, the disposition PASS on the JOBLIB statement need not be included.

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
```

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

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 discussion 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.

This section describes diagnostic messages generated by the system. Compiler messages and object time messages are discussed.

COMPILER DIAGNOSTIC MESSAGES

The COBOL compile-time message that follows serves as an example of the format of COBOL compiler messages.

```
005  IKF4046I-E  ILLEGAL TO *****/***** FILE *****.  STATEMENT
                 DISCARDED..
```

The code "005" at the very left is the card number of the statement in which the error has occurred. Some errors may not be discovered until information from various sections of the program is combined. For this reason, the source card number in the error message may not be exact. IKF identifies this as a COBOL compiler message. 4046 is the identifying number of the message. The symbol "I" means that this is a message to the programmer for his action. E is a level of severity in the error codes that follow. The five consecutive asterisks that appear in some of the messages (including the example above) indicate a message that varies with the program being compiled.

- W Warning -- indicates that an error was made in the source program. However, it is not serious enough to hinder the execution of the program.

- C Conditional -- indicates that an error was made but that the compiler usually makes a corrective assumption. The statement containing the error is retained. Execution can be attempted for the debugging value.

- E Error -- indicates that a serious error was made. Usually the compiler makes no corrective assumption. The statement containing the error is dropped. Execution of program should not be attempted.

- D Disaster -- indicates that a serious error was made. Compilation is not completed. Results are unpredictable.

The severity level of the messages is correlated with the return code issued by the compiler at the end of compilation.

Severity Level	Return Code
(No messages)	0
W	4
C	8
E	12
D	16

The return code indicated is the most severe level occurring in the messages. This code can be tested by the programmer with the COND parameter in the EXEC statement.

The message text is usually composed of two sentences. The first describes the error; the second describes what the compiler has done as a result of the error. Thus, most of the messages are self-explanatory, except in two situations:

1. When no compiler action is given. These messages are numbered in the 3000 series. They appear in combination with other messages that do have the compiler action described.

2. When messages describe errors that require an explanation too long to include in a message. These explanations appear in text under the messages.

Words in a message that must vary according to the program being compiled are denoted by five asterisks (*****) in the messages printed below. When diagnostic messages are printed, the programmer will find that wherever there are three asterisks (***) in the printout the compiler has encountered unrecognizable information.

Unless otherwise indicated, all these messages appear in the compiler output listing. Messages IKF0001I through IKF0015I and IKF6001I through IKF6007I may be interspersed in the compiler output listing. In addition, message IKF0003I appears on the chief operator's console. Messages IKF1001I through IKF5017I are grouped together following the compiler output listing.

IKF0001I-D SIZE PARAMETER TOO SMALL FOR THIS PROGRAM.

Explanation: Compiler was unable to allocate sufficient table space for the source program.

Programmer Response: Probable user error. Specify a larger amount of main storage in the SIZE option of the PARM parameter in the EXEC statement, or divide the program into smaller segments and utilize the overlay feature of the linkage editor.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF0002I-D INVALID BASIS LIBRARY NAME OR LIB OPTION NOT SPECIFIED. COMPILATION ABANDONED.

Explanation: Library name given on the BASIS card or in the COPY statement does not exist, or the LIB option is not in effect.

Programmer Response: Probable user error. Check that a SYSLIB DD statement is present that defines a library which contains the member whose name is specified in the BASIS statement.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF0003I- Jobname, stepname, unit address, device type, ddname, operation attempted, error description. Each of the following fields if printed will vary in length:

UNIT RECORD--Access method
MAGNETIC TAPE--Relative block number (decimal), access method
DIRECT ACCESS--Actual track address and block number

Programmer Response: Provide the additional control information required; 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.

IKF0005I-W ***** INVALID BLOCKSIZE. DEFAULT USED.

Explanation: Blocksize specified on DD card is not an integral multiple of record length, or blocking factor is too large.

Programmer Response: Probable user error. Change blocksize parameter on DD card or accept default action.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF0006I-D ***** DATASET NOT USABLE. JOB STEP CANNOT EXECUTE.

Explanation: Data set required by the compiler cannot be opened. Program cannot be compiled.

Programmer Response: Probable user error. Check DD card.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF0007I-W SYSLIN NOT USABLE. LOAD OPTION CANCELLED.

Explanation: SYSLIN cannot be opened.

Programmer Response: Probable user error. Check DD card.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF0008I-W SYSPUNCH NOT USABLE. DECK OPTION CANCELLED.

Explanation: SYSPUNCH cannot be opened.

Programmer Response: Probable user error. Check DD card.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF0009I-W SIZE / BUF PARAMETER IGNORED. DEFAULT USED.

Explanation: Value of parameters is inadequate to calculate table and dictionary space for compilation purposes, or BUF parameter is less than minimum required.

Programmer Response: Probable user error. If the default value is unsuitable, increase the value for the SIZE parameter or decrease the BUF parameter.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF0010I-D A TABLE HAS EXCEEDED THE MAXIMUM PERMISSIBLE SIZE.

Programmer Response: Probable user error. Check all tables for features that significantly affect storage usage before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF0015I-D BUF SUBPARAMETER IN PARM FIELD TOO SMALL FOR DD CARD BLKSIZE SUBPARAMETERS. COMPILATION ABANDONED.

Explanation: The size of the buffer used for SYSPUNCH, SYSLIN, SYSIN, SYSPRINT, and SYSLIB exceeds BUF-1280 bytes.

Programmer Response: Probable user error. Increase the BUF parameter in the EXEC card or decrease the BLKSIZE values specified in the DD cards for SYSPRINT, SYSLIN, SYSLIB, SYSPUNCH, or SYSIN.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF0020I-D COMPILER ERROR. COMPILATION WILL NOT BE COMPLETE.

Explanation: A logic error or a machine error occurred in table handling.

Programmer Response: The compiler has reached a point in its processing where it is unable to continue. If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF0030I-D FRAGMENTED CORE. RUN IN LARGER REGION SIZE.

Programmer Response: Probable user error. Increase the amount of main storage requested.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1001I-C NUMERIC LITERAL NOT RECOGNIZED AS LEVEL NUMBER BECAUSE ***** ILLEGAL AS USED. SKIPPING TO NEXT LEVEL, SECTION OR DIVISION.

Programmer Response: Probable user error. Check the item indicated, as well as the statements preceding it, for possible error; 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.

IKF1002I-W ***** SECTION HEADER MISSING. ASSUMED PRESENT.

Programmer Response: Probable user error. Supply section header, or if present, correct its syntax (check for a margin error or a misspelling) before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1003I-W ***** PARAGRAPH NAME MISSING. ASSUMED PRESENT.

Programmer Response: Probable user error. Supply paragraph name, or if present, correct its syntax (check for a margin error or a misspelling) before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1004I-E INVALID WORD *****. SKIPPING TO NEXT RECOGNIZABLE WORD.

Programmer Response: Probable user error. Correct invalid word or syntax error before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1005I-E INVALID ORDER IN ENVIRONMENT DIVISION. SKIPPING TO NEXT DIVISION.

Programmer Response: Probable user error. Correct the order of sections and/or paragraphs in the Environment Division before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1006I-E DECLARATIVES SECTION WITHOUT USE SENTENCE. SECTION CAN NEVER BE EXECUTED.

Programmer Response: Probable user error. Supply USE sentence before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1007I-W ***** NOT PRECEDED BY A SPACE. ASSUME SPACE.

Programmer Response: Probable user error. Check syntax, supply space where needed, and recompile if necessary.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1008I-W RIGHT PAREN SHOULD NOT BE PRECEDED BY SPACE.

Programmer Response: Probable user error. Delete space preceding right parenthesis and recompile if necessary.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1009I-E COPY MUST BE PRECEDED BY PROCEDURE-NAME. IGNORED.

Programmer Response: Probable user error. Supply procedure-name, or if present, correct its syntax (check for a margin error or a misspelling) before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1010I-W LEFT PAREN SHOULD NOT BE FOLLOWED BY SPACE.

Programmer Response: Probable user error. Delete space following left parenthesis and recompile if necessary.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1011I-C RECORDING MODE SPECIFICATION IS INVALID. ASSUMED VARIABLE.

Programmer Response: Probable user error. Correct RECORDING MODE specification to agree with record description before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1012I-E FILE-NAME NOT UNIQUE. USING FIRST DEFINITION.

Programmer Response: Probable user error. Correct the duplication either by deleting a redundant SELECT sentence or by replacing a misspelled file-name in a SELECT sentence before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1013I-E CHARACTER LENGTH IN SPECIAL-NAMES MUST BE ONE.

Programmer Response: Probable user error. Change length of nonnumeric literal to one before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1014I-W 'FILE' NOT PRESENT IN MULTIPLE FILE CLAUSE. ASSUMED PRESENT.

Programmer Response: Probable user error. Put the key word 'FILE' in the MULTIPLE FILE TAPE clause and recompile if necessary.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1015I-E EXTERNAL NAME IN SYSTEM-NAME ***** INVALID. SYSTEM-NAME IGNORED.

Programmer Response: Probable user error. Correct external-name in system-name.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1016I-E MORE THAN ONE ***** CLAUSE. SKIPPING TO NEXT CLAUSE.

Programmer Response: Probable user error. Delete multiple occurrence of clause from entry before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1017I-E ***** INVALID IN ***** CLAUSE. SKIPPING TO NEXT CLAUSE.

Programmer Response: Probable user error. Replace or delete the invalid specification before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1018I-E COPY CLAUSE INVALID IN A COPY LIBRARY OR LIB OPTION NOT SPECIFIED. IGNORED.

Programmer Response: Probable user error. Correct library member or specify the LIB option before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1019I-E NO LIBRARY NAME. COPY CLAUSE IGNORED.

Programmer Response: Probable user error. Supply library-name before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1020I-E ***** MUST BE PROCEDURE-NAME FOLLOWING DEBUG. *****.

Programmer Response: Probable user error. Add or correct word following DEBUG to conform to rules for a valid procedure-name before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1021I-E ***** DOES NOT BELONG ON DEBUG CARD. SKIPPING TO NEXT CARD.

Programmer Response: Probable user error. Delete invalid specification from DEBUG card before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1022I-W PERIOD DOES NOT BELONG ON DEBUG CARD. DELETED.

Programmer Response: Probable user error. Delete period from DEBUG card and recompile if necessary.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1023I-E INVALID FILE-NAME. USE IGNORED.

Programmer Response: Probable user error. Supply valid file-name before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1024I-E UNDEFINED FILE-NAME. USE IGNORED.

Programmer Response: Probable user error. Supply valid SELECT sentence for file-name before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1025I-C REDEFINES CLAUSE NOT FIRST CLAUSE FOLLOWING DATA-NAME. ASSUMED FIRST.

Programmer Response: Probable user error. Make REDEFINES clause first clause following data-name before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1026I-W TOTALED AND TOTALING AREA CLAUSES MUST BOTH BE SPECIFIED.

Programmer Response: Probable user error. Amend the source statement to include either TOTALED or TOTALING, whichever has been omitted from the LABEL RECORDS clause. 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.

IKF1027I-W FILE WITH TOTALED AREA CLAUSE AND NONSTANDARD LABELS MAY NOT BE OPENED OUTPUT.

Programmer Response: Probable user error. When the TOTALED AREA clause is associated with a file opened as output, you must be processing user labels (i.e., LABEL RECORDS are data-name-1....). Change the source statement(s) as warranted; 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.

IKF1028I-E ***** SENTENCE IMPROPERLY WRITTEN. SENTENCE IGNORED.

Programmer Response: Probable user error. Correct syntax of sentence before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1029I-E ***** IN ***** SENTENCE NOT DEFINED AS FILE-NAME. NAME IGNORED.

Programmer Response: Probable user error. Make sure that file-name is validly defined in a SELECT sentence before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1030I-E ***** IN ***** SENTENCE IS INVALID. WORD IGNORED.

Programmer Response: Probable user error. Supply valid word before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1031I-C USE SENTENCE NOT PRECEDED BY SECTION-NAME. SECTION-NAME ASSUMED.

Programmer Response: Probable user error. Supply section-name before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1032I-E ***** INCORRECTLY USED IN USE SENTENCE. SENTENCE IGNORED.

Programmer Response: Probable user error. Correct syntax of USE sentence before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1033I-W ***** FILE-NAME ALREADY ASSIGNED THIS 'SAME' CLAUSE OPTION. USING FIRST ONE.

Programmer Response: Probable user error. Delete duplicate SAME clause, specify correct SAME clause option, or correct a misspelled file-name before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF11034I-W ***** CLAUSE IN ***** LEVEL IS TREATED AS COMMENTS IN OS. NEXT CLAUSE.

Programmer Response: Probable user error. Correct data item description entry before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1035I-W INTEGER NOT PRESENT IN MULTIPLE FILE CLAUSE.

Programmer Response: Probable user error. Indicate position of file by specifying the "POSITION integer-n" option.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1036I-C QUALIFIED NAME INVALID AFTER LEVEL NUMBER. USING LOWEST NAME.

Programmer Response: Probable user error. Correct data-name following level number before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1037I-E ***** INVALID IN DATA DESCRIPTION. SKIPPING TO NEXT CLAUSE.

Programmer Response: Probable user error. Correct or delete invalid clause in data description entry before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1038I-E ***** INVALID AFTER LEVEL NUMBER. SKIPPING TO NEXT LEVEL.

Programmer Response: Probable user error. Correct data-name following level number before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1039I-W DATA-NAME IN ***** CLAUSE NEED NOT BE QUALIFIED. USING LOWEST NAME.

Programmer Response: Probable user error. Remove qualification of data-name and recompile if necessary.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1040I-E IMPROPER LEVEL NUMBER FOR FILE-SECTION.

Programmer Response: Probable user error. Correct invalid level numbers (77's) in the File Section before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1041I-E ***** INVALID AS USED IN ***** SECTION. SKIPPING TO NEXT LEVEL, SECTION OR DIVISION.

Programmer Response: Probable user error. Correct invalid specification by deleting it or moving it to its proper place in source program before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1042I-E ASSIGN CLAUSE MISSING IN SELECT. CONTINUING.

Programmer Response: Probable user error. Supply ASSIGN clause for file before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1043I-W END OF SENTENCE SHOULD PRECEDE *****. ASSUMED PRESENT.

Programmer Response: Probable user error. Supply period to terminate sentence and recompile if necessary.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1044I-E INVALID WORD *****. SKIPPING TO NEXT LEVEL, SECTION OR DIVISION.

Programmer Response: Probable user error. Check the syntax of the source statement for a possible misspelling or transposition or an invalidly formed word before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1045I-E INVALID ORDER IN ***** SECTION.

Programmer Response: Probable user error. Correct order of paragraphs before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1046I-E MEMBER NOT FOUND IN LIBRARY. IGNORING COPY.

Programmer Response: Probable user error. Correct misspelled library-name or make sure member is in the library before recompiling.

If the problem recurs, do the following before calling IBM for programming support. Have source deck, control cards, compiler output, and listing of source statement library available.

IKF1047I-E LIBRARY NOT FOUND ON SYSTEM. IGNORING COPY.

Programmer Response: Probable user error. Ensure that the source statement library is assigned before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1048I-E LIBRARY MEMBER HAS BAD TRACK. IGNORING REST OF COPY.

Programmer Response: Probable user error. Recreate library on a good device before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1049I-W ***** FILE-NAME ALREADY ASSIGNED THIS MULTIPLE FILE CLAUSE OPTION. USING FIRST ONE.

Programmer Response: Probable user error. Delete duplicate mention of file-name or correct a misspelled file-name before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1050I-C ***** FILE ALREADY ASSIGNED THIS APPLY OPTION. FILE-NAME IGNORED.

Programmer Response: Probable user error. Delete duplicate APPLY option for file-name or correct a misspelled file-name before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1051I-E NO DATA-NAME IN USE SENTENCE. SENTENCE IGNORED.

Programmer Response: Probable user error. Include data-name in USE sentence before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1052I-E ***** ILLEGALLY USED IN USE SENTENCE. END SENTENCE, RESCANNING AT NEXT RECOGNIZABLE WORD.

Programmer Response: Probable user error. Check syntax of USE statement and supply a valid data-name before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1053I-E ***** CLAUSE INVALID. CLAUSE IGNORED.

Programmer Response: Probable user error. Correct invalid clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1054I-E OPERAND FOR INITIATE NOT FOUND OR ILLEGAL. OPERAND DROPPED.

Programmer Response: Probable user error. Supply valid operand for INITIATE statement before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1055I-E VALID FILE-NAME NOT PRESENT. DESCRIPTION IGNORED.

Programmer Response: Probable user error. Supply valid file-name before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1056I-E FILE-NAME NOT DEFINED IN A SELECT. DESCRIPTION IGNORED.

Programmer Response: Probable user error. Check that the SELECT sentence has not been discarded due to a syntax error or correct a misspelled file-name before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1057I-E FIRST WORD IN REPORT SECTION NOT RD. IGNORED.

Programmer Response: Probable user error. Correct syntax of Report Section before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1058I-E NO REPORTS CLAUSE IN FILE SECTION. REPORT SECTION IGNORED.

Programmer Response: Probable user error. Ensure that a valid REPORT clause is included in File Section before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1059I-E NO REPORT CLAUSE FOR RD. RD IGNORED.

Programmer Response: Probable user error. Ensure that the report-name is specified in a REPORT clause in the File Section for the file on which the report is to be written before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1060I-E INVALID WORD IN REPORT WRITER STATEMENT. IGNORED.

Programmer Response: Probable user error. Correct invalid word before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1061I-E DUPLICATE CLAUSE. DROPPED.

Programmer Response: Probable user error. Delete one of duplicate clauses before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1062I-E ***** IN COPY REPLACING STATEMENT INVALID AS BCD NAME.

Programmer Response: Probable user error. Replace indicated word with valid configuration before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1063I-E DUPLICATE ENTRY IN PAGE CLAUSE. DROPPED.

Programmer Response: Probable user error. Remove one of duplicate entries before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1064I-E NO TYPE CLAUSE SPECIFIED. SKIPPING TO NEXT 01.

Programmer Response: Probable user error. Supply TYPE clause for this report group before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1065I-E INTEGER MISSING IN PAGE CLAUSE. ENTRY IGNORED.

Programmer Response: Probable user error. Ensure that an integer is specified for each PAGE clause entry before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1066I-E INVALID WORD IN PAGE CLAUSE. SKIPPING TO NEXT RECOGNIZABLE WORD.

Programmer Response: Probable user error. Correct spelling of PAGE clause entries before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1067I-E INVALID HEADER. SKIPPING TO NEXT RECOGNIZABLE WORD.

Programmer Response: Probable user error. Supply valid header before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1068I-E OPERAND FOR GENERATE NOT FOUND. CLAUSE DROPPED.

Programmer Response: Probable user error. Supply GENERATE clause operand before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1069I-E INVALID TYPE CLAUSE. SKIPPING TO NEXT 01.

Programmer Response: Probable user error. Correct TYPE clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1070I-C FLT-PT LIT MANTISSA EXCEEDS 16 DIGITS. TRUNCATED TO 16.

Programmer Response: Probable user error. Specify a mantissa of no more than 16 digits before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1071I-C FLT-PT LIT EXPONENT EXCEEDS 2 DIGITS. TRUNCATED TO 2. RESCANNING.

Programmer Response: Probable user error. Specify an exponent of no more than 2 digits before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1072I-C FLT-PT LIT EXPONENT FOLLOWED BY NON-BLANK. RESCANNING AT NON-BLANK.

Programmer Response: Probable user error. Ensure that a blank follows exponent before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1073I-C FLT-PT LIT E FOLLOWED BY INVALID CHARACTER. RESCANNING AT E.

Programmer Response: Probable user error. Supply valid character to follow E before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1074I-C FLT-PT LIT SIGN FOLLOWED BY INVALID CHARACTER. RESCANNING AT E.

Programmer Response: Probable user error. Supply valid character to follow sign before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1075I-C FLT-PT LIT EXCEEDS LIMIT. ASSUME MAX OR MIN PER SIGN OF EXPONENT.

Programmer Response: Probable user error. Respecify valid literal before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1076I-C ALPHANUMERIC LIT EXCEEDS 120 CHARACTERS. TRUNCATED TO 120.

Programmer Response: Probable user error. Ensure that alphanumeric literal contains no more than 120 characters before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1077I-C ALPHANUMERIC LIT CONTINUES IN A-MARGIN. ASSUME B-MARGIN.

Programmer Response: Probable user error. Correct syntax of continuation line and recompile if necessary.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1078I-W ALPHA-LITERAL CONTINUED WITH MISSING HYPHEN OR QUOTE. ASSUMED.

Programmer Response: Probable user error. Supply continuation character and recompile if necessary.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1079I-C ALPHANUMERIC LIT HAS ZERO LENGTH. ASSUME ONE SPACE.

Programmer Response: Probable user error. Specify valid alphanumeric literal before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1080I-W PERIOD PRECEDED BY SPACE. ASSUME END OF SENTENCE.

Programmer Response: Probable user error. Ensure that no spaces precede period and recompile if necessary.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1081I-W PERIOD NOT FOLLOWED BY SPACE. ASSUME END OF SENTENCE.

Programmer Response: Probable user error. Either delete period or follow it with a space, whichever is appropriate, before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1082I-C NUMERIC LIT EXCEEDS 18 DIGITS. TRUNCATED TO 18.

Programmer Response: Probable user error. Supply a numeric literal of no more than 18 digits before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1083I-C ILLEGAL CHARACTER. SCAN RESUMED AT NEXT VALID CHARACTER.

Programmer Response: Probable user error. Remove or replace invalid character before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1084I-W COMMA SHOULD NOT BE PRECEDED BY SPACE.

Programmer Response: Probable user error. Ensure that no spaces precede comma and recompile if necessary.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1085I-C WORD OR PICTURE EXCEEDS 30 CHARACTERS. TRUNCATED TO 30 CHARACTERS.

Programmer Response: Probable user error. Supply a word or PICTURE of no more than 30 characters before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1086I-W ***** SHOULD BEGIN A-MARGIN.

Programmer Response: Probable user error. Begin indicated word in A-margin before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1087I-W '*****' SHOULD NOT BEGIN A-MARGIN.

Programmer Response: Probable user error. Begin indicated word in B-margin before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1088I-E MISSING FIRST INSERT OR DELETE CARD. PASS CARDS UNTIL FOUND. *****.

Programmer Response: Probable user error. Supply INSERT or DELETE card before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1089I-E INSERT OR DELETE NUMBER OUT OF SEQUENCE. SKIPPING TO NEXT INSERT OR DELETE NUMBER. *****.

Programmer Response: Probable user error. Correct sequence of inserted or deleted numbers before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1090I-E DELETE THRU NUMBER OUT OF SEQUENCE. PASS CARDS UNTIL NEXT INSERT OR DELETE. *****.

Programmer Response: Probable user error. Resequence statement numbers of cards following DELETE and ensure that sequence numbers specified on DELETE card appear in source program before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1091I-C ***** IN A-MARGIN NOT VALID AS PROC-NM. ASSUME B-MARGIN.

Programmer Response: Probable user error. If indicated name is a procedure-name, correct formation before recompiling. If indicated name is not a procedure-name, ensure that it begins in B-margin and recompile if necessary.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1092I-E DECLARATIVES DO NOT FOLLOW PROCEDURE DIVISION. IGNORED.

Programmer Response: Probable user error. Ensure that Declaratives Section immediately follows Procedure Division header before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1093I-E NO DECLARATIVES SECTION. END DECLARATIVES IGNORED.

Programmer Response: Probable user error. Depending upon the logic of the program, either remove END DECLARATIVES statement and recompile if necessary, or add a Declaratives Section before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1094I-E INTEGER IN 'NEXT GROUP' CLAUSE DOES NOT CONFORM TO PAGE CLAUSE SPECIFICATIONS. CONTINUING.

Programmer Response: Probable user error. Supply an "integer" in NEXT GROUP clause that is compatible with PAGE clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1095I-W WORD 'SECTION' OR 'DIVISION' MISSING. ASSUMED PRESENT.

Programmer Response: Probable user error. Add missing word and recompile if necessary.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1096I-E THE DATA-NAME IN THE UPON CLAUSE HAS NOT BEEN SPECIFIED AS A DATA-NAME FOR A TYPE DETAIL REPORT GROUP. UPON OPTION IGNORED.

Programmer Response: Probable user error. Ensure that the TYPE DETAIL clause is specified for the data-name at the 01-level before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1097I-E PROGRAM-ID MISSING OR MISPLACED. IF PROGRAM-ID DOES NOT IMMEDIATELY FOLLOW IDENTIFICATION DIVISION, IT WILL BE IGNORED.

Programmer Response: Probable user error. Ensure that the PROGRAM-ID clause immediately follows IDENTIFICATION DIVISION header before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1098I-E ALPHA LITERAL NOT CONTINUED WITH HYPHEN AND QUOTE. END LITERAL ON LAST CARD.

Programmer Response: Probable user error. Insert a hyphen in column 7 of the continuation line, and a quotation mark preceding the continuation of the literal anywhere in Column B before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1099I-E ***** IS INVALID AS USED. IGNORED.

Programmer Response: Probable user error. Correct indicated item before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1100I-W ***** SEQUENCE ERRORS IN SOURCE PROGRAM.

Programmer Response: Probable user error. Correct sequence errors and recompile if necessary.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1101I-E NEXT PAGE NOT IN FIRST LINE CLAUSE. IGNORED.

Programmer Response: Probable user error. Correct placement of NEXT PAGE option before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1102I-C INCOMPLETE ELEMENTARY ITEM. ASSUME VALUE SPACES.

Programmer Response: Probable user error. Correct description of elementary item before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1103I-E GROUP TYPE ALLOWED ONCE FOR RD. IGNORED.

Programmer Response: Probable user error. Remove duplicate TYPE option before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1104I-E CONTROL NAME NOT SPECIFIED IN RD. SKIPPING TO NEXT 01.

Programmer Response: Probable user error. Ensure that identifier is specified in a CONTROL clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1105I-W ELEMENTARY ITEM EXPECTED. ASSUMED.

Programmer Response: Probable user error. Supply elementary item before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1106I-E OPERAND FOR TERMINATE NOT FOUND OR ILLEGAL. OPERAND DROPPED.

Programmer Response: Probable user error. Supply missing operand or correct invalid one.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1107I-C 'NEXT GROUP' CLAUSE IS ILLEGAL FOR THIS REPORT GROUP. IGNORED.

Programmer Response: Probable user error. Remove NEXT GROUP clause from PH, PF, or CF report group entry before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1108I-E ***** IS NOT A POSITIVE INTEGRAL NUMBER. ASSUMED ONE.

Programmer Response: Probable user error. Supply a valid positive integer before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1109I-E DUPLICATE USE OF CONTROL NAME. SKIPPING TO NEXT 01.

Programmer Response: Probable user error. Eliminate duplication before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1110I-E INVALID USE OF SUM CLAUSE. CLAUSE IGNORED.

Programmer Response: Probable user error. Correct invalid use of SUM clause and recompile if necessary.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1111I-W ELEMENTARY LEVEL WITHOUT COLUMN OR SUM CLAUSE.

Programmer Response: Probable user error. If entry is not to be suppressed, supply COLUMN clause before recompiling. If sum counter is to be referenced elsewhere in the program, supply SUM clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1112I-E ***** ALREADY SPECIFIED IN TWO FILE DESCRIPTION ENTRIES. IGNORED.

Programmer Response: Probable user error. Ensure that a given report-name appears in no more than two REPORT clauses before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1113I-E EXPECTING 6-DIGIT SEQUENCE NUMBER. SKIPPING TO NEXT INSERT OR DELETE NUMBER. *****.

Programmer Response: Probable user error. Correct sequence-number-field before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1114I-C EXTRANEOUS COMMA OR HYPHEN ON DELETE CARD. IGNORED.

Programmer Response: Probable user error. Correct syntax of INSERT or DELETE card and recompile if necessary.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1115I-E NO BLANK, COMMA OR HYPHEN FOLLOWING SEQUENCE NUMBER. ASSUME BLANK. *****.

Programmer Response: Probable user error. Provide valid sequence number separator before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1116I-E EXPECTING 6-DIGIT SEQUENCE NUMBER AFTER HYPHEN. IGNORING DELETE FROM THRU NUMBER. *****.

Programmer Response: Probable user error. Supply six-digit sequence number before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1117I-E DELETE NUMBER GREATER THAN LAST SEQUENCE NUMBER. STOP INSERT AND DELETE. *****.

Programmer Response: Probable user error. Ensure that DELETE sequence number is within library entry before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1118I-E INSERT NUMBER GREATER THAN LAST SEQUENCE NUMBER. STOP INSERT AND DELETE. *****.

Programmer Response: Probable user error. Ensure that INSERT sequence number is within library entry before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1119I-E INTEGER IN 'LINE CLAUSE' DOES NOT CONFORM TO PAGE CLAUSE SPECIFICATIONS. STATEMENT ACCEPTED AS WRITTEN.

Programmer Response: Probable user error. Ensure that LINE clause is compatible with PAGE clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1120I-W COMMA NOT FOLLOWED BY SPACE. ASSUMED SPACE.

Programmer Response: Probable user error. Insert a space after comma and recompile if necessary.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1121I-W PERIOD OR COMMA INVALID AS USED IN PICTURE CLAUSE.

Programmer Response: Probable user error. Supply valid PICTURE clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1122I-E EXTERNAL-NAME IN RERUN CLAUSE MUST NOT BE THE SAME AS SYSTEM NAME USED IN ASSIGN CLAUSE. SENTENCE IGNORED.

Programmer Response: Probable user error. Correct duplicate use of name before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1123I-E NUMBER IS ZERO OR NEGATIVE. SENTENCE IGNORED.

Programmer Response: Probable user error. Supply valid positive integer before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1124I-E NUMBER TOO LARGE FOR RERUN. SENTENCE IGNORED.

Programmer Response: Probable user error. Provide a number no larger than allowable maximum before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1125I-C '*****' FILE-NAME USED IN PREVIOUS RERUN. USING FIRST ONE.

Programmer Response: Probable user error. Ensure that a given file-name appears in only one RERUN clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1126I-E '*****' FILE-NAME SPECIFIED IN BOTH RERUN AND USING OR GIVING OPTION. RERUN IGNORED.

Programmer Response: Probable user error. Specify file-name only in using or giving option.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

ILF1127I-E ***** INVALID IN ***** SENTENCE. REST OF SENTENCE IGNORED.

Programmer Response: Probable user error. Correct invalid entry in indicated sentence before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1128I-W FOUND *****. EXPECTING ENVIRONMENT. ALL ENV. DIV. STATEMENTS IGNORED.

Programmer Response: Probable user error. Ensure that ENVIRONMENT DIVISION header precedes Environment Division statements before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1129I-C ID DIV. HEADER MISSING OR MISPLACED. ASSUMED PRESENT.

Programmer Response: Probable user error. Ensure that an IDENTIFICATION DIVISION header appears as first source statement in program and recompile if necessary.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1130I-E ***** DIV. HEADER MISSING. WORDS IN ***** STATEMENT ARE INVALID.

Programmer Response: Probable user error. Supply indicated division header before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1131I-W INVALID PRIORITY NUMBER. ZERO ASSUMED.

Programmer Response: Probable user error. Supply a valid priority number before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1132I-E INVALID SYSTEM-NAME. SKIPPING TO NEXT CLAUSE.

Programmer Response: Probable user error. Correct system-name before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1133I-C MORE THAN 1 USE ON STANDARD ERROR SPECIFIED FOR SAME FILE ON OPEN OPTION. USE IGNORED.

Programmer Response: Probable user error. Ensure that a given file-name is not referred to implicitly or explicitly in more than one USE AFTER STANDARD ERROR procedure before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1134I-E USE SPECIFIED FOR '*****' WITH LABEL RECORDS OMITTED OR STANDARD. SENTENCE IGNORED.

Programmer Response: Probable user error. Either specify the LABEL RECORDS clause with the data-name option or remove USE procedure for labels before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1135I-W INTEGER-1 OUTSIDE OF ALLOWABLE LIMITS. 1 ASSUMED.

Programmer Response: Probable user error. Correct integer-1 specification before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1136I-E DATA-NAME ALREADY SPECIFIED FOR A TYPE DETAIL REPORT GROUP. SKIPPING TO NEXT 01, RD, OR SECTION.

Programmer Response: Probable user error. Ensure that each DETAIL report group has a unique data-name at level-01 in the report before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1137I-W MINIMUM NUMBER OF OCCURRENCES IN OCCURS CLAUSE NOT LESS THAN MAXIMUM NUMBER. CONTINUING.

Programmer Response: Probable user error. Correct the OCCURS clause to ensure that integer-1 is less than integer-2 before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1138I-W APPLY ***** IS A FUNCTION OF JCL IN OS.

Programmer Response: Probable user error. Ensure that the APPLY option is handled via JCL before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1139I-E ***** DUPLICATELY DEFINED SECTION. SECTION NAME IGNORED.

Programmer Response: Probable user error. Remove duplication of indicated section before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1140I-C NUMERIC LITERAL ***** EXCEEDS MAXIMUM. SUBSTITUTING 32768.

Programmer Response: Probable user error. Supply a literal no larger than 32768 before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1141I-C FILE ORGANIZATION FIELD INVALID IN SYSTEM-NAME. SEQUENTIAL ASSUMED.

Programmer Response: Probable user error. Supply a valid organization field in system-name of ASSIGN clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1142I-E USE FOR STANDARD ERROR OR LABEL PROCEDURE SPECIFIED FOR FILE AND OPEN OPTION. USE FOR OPEN OPTION IGNORED.

Programmer Response: Probable user error. Ensure that a given file-name is not referred to, implicitly or explicitly, in more than one USE for error or label processing declarative before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1143I-E USE STATEMENTS IMPLY STANDARD AND NON-STANDARD LABELS. USE IGNORED.

Programmer Response: Probable user error. Ensure that if a USE BEFORE label procedure is specified for the file, a USE AFTER is not also specified for the same file, and vice versa, before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1144I-W WRITE AFTER POSITIONING AND WRITE BEFORE ADVANCING ILLEGALLY USED FOR 1 FILE.

Programmer Response: Probable user error. Ensure that the ADVANCING and POSITIONING options are not both specified for the same file before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1145I-C ***** DUPLICATELY DEFINED IN SPECIAL NAMES PARAGRAPH. SENTENCE IGNORED.

Programmer Response: Probable user error. Eliminate duplicate definition of indicated item before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1147I-W SD FILE ILLEGALLY SPECIFIED IN SAME AREA CLAUSE. CLAUSE FOR SD IGNORED.

Programmer Response: Probable user error. Ensure that a sort-file-name does not appear in a SAME AREA clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1148I-C INVALID SEGMENT LIMIT. FIFTY ASSUMED.

Programmer Response: Probable user error. Supply a valid segment limit before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1149I-W FILES IN SAME AREA CLAUSE DO NOT ALL APPEAR IN THE SAME SORT/RECORD AREA CLAUSE. '*****' NOT GIVEN SAME AREA NUMBERS.

Programmer Response: Probable user error. Ensure that if one or more file-names of a SAME AREA clause appear in a SAME SORT/RECORD AREA clause, all file-names in former clause appear in latter clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1150I-W FILES IN SAME SORT/RECORD AREA CLAUSE ARE NOT ALL SPECIFIED IN THE SAME 'SAME AREA' CLAUSE. '*****' NOT GIVEN SAME RECORD SORT NUMBER.

Programmer Response: Probable user error. Simply verify that for your purposes there is no need to specify in the SAME AREA clause all the file-names given in the SAME RECORD AREA clause. Otherwise, change the source statement as necessary and recompile.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1151I-E ILLEGAL CHARACTER USE IN CURRENCY SIGN CLAUSE. CLAUSE IGNORED.

Programmer Response: Probable user error. Correct literal in CURRENCY SIGN clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1154I-W 2 DIFFERENT LABEL PROCEDURES FOR EOF AND EOV WITH 'BEFORE' OPTION. EOV LABEL PROCEDURE IGNORED.

Programmer Response: Probable user error. Ensure that a file is not referenced, implicitly or explicitly, in more than one USE statement with the BEFORE option before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1155I-W DEVICE CLASS INVALID IN SYSTEM-NAME. SKIPPING TO NEXT FIELD.

Programmer Response: Probable user error. Supply a valid device-class field in system-name of ASSIGN clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1156I-W DEVICE NUMBER INVALID IN SYSTEM-NAME. '*****' ASSUMED.

Programmer Response: Probable user error. Supply a valid device-number field in system-name of ASSIGN clause and recompile if necessary.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1157I-E EXTERNAL NAME NOT PRESENT IN SYSTEM-NAME.

Programmer Response: Probable user error. Include in the system-name a name field of one to eight characters specifying the external-name by which the file is known to the system. 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.

IKF1158I-W '*****' IN ENTRY STATEMENT IS SAME AS PROGRAM-ID. '*****' IGNORED FOR ENTRY VERB.

Programmer Response: Probable user error. Ensure that the literal specified in the ENTRY statement is not the same name as the PROGRAM-ID before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1159I-W PAGE LIMIT INTEGER-1 NOT SPECIFIED OR INVALID. ASSUME HIGH VALUE.

Programmer Response: Probable user error. Specify integer-1 if other than relative LINE NUMBERS are to be used before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1160I-E CONTINUATION OF WORD FOUND IN A-MARGIN.

Programmer Response: Probable user error. Begin continued word in B-margin before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1161I-W RESERVED WORD MISSING. ASSUMED PRESENT.

Programmer Response: Probable user error. Correct syntax of clause and recompile if necessary.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1162I-E INTEGER IN LINE CLAUSE IS LESS THAN PREVIOUS VALUE. IGNORED.

Programmer Response: Probable user error. Ensure that LINE NUMBER entries are given in ascending order before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1163I-E ABSOLUTE LINE NUMBER IS PRECEDED BY A RELATIVE LINE NUMBER. IGNORED.

Programmer Response: Probable user error. Check the source statement to be sure that the absolute LINE NUMBER entries appear in ascending order and that no relative LINE NUMBER precedes an absolute LINE NUMBER. Change the source statement if necessary and recompile.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1164I-E NO PAGE CLAUSE SPECIFIED. ALL LINE CLAUSES MUST BE 'LINE PLUS INTEGER'. IGNORED.

Programmer Response: Probable user error. Specify the PAGE LIMIT clause if other than relative LINE NUMBER entries are desired before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1165I-E 'HEADING' EQUALS 'FIRST DETAIL' IN PAGE CLAUSE. PAGE HEADING IS ILLEGAL. CONTINUING.

Programmer Response: Probable user error. Correct PAGE LIMIT clause so that FIRST DETAIL integer is greater than HEADING integer before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1166I-E 'FOOTING' EQUALS 'PAGE LIMIT' IN PAGE CLAUSE. PAGE FOOTING IS ILLEGAL. CONTINUING.

Programmer Response: Probable user error. Ensure that the line number of the FOOTING is less than the integer specified in the PAGE LIMIT clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1167I-E 'LINE NEXT PAGE' CLAUSE IS ILLEGAL FOR THIS REPORT GROUP. IGNORED.

Programmer Response: Probable user error. Ensure that LINE NEXT PAGE is not specified for RH, PH, or PF report groups, or for report groups within reports with no PAGE LIMIT clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1168I-E DUPLICATE REPORT-NAME. SKIPPING TO NEW RD.

Programmer Response: Probable user error. Ensure that each report-name is unique before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1170I-E DETAIL REPORT GROUP SPECIFIED WITH NO DATA-NAME. CONTINUING.

Programmer Response: Probable user error. Ensure that SUM clause operand appears as a source item in indicated DETAIL before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1171I-E INTEGERS IN PAGE CLAUSE ARE NOT IN ASCENDING ORDER. CONTINUING.

Programmer Response: Probable user error. Ensure that PAGE LIMIT integers (integer-2 through integer-5) are in ascending order before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1172I-E WORD INVALID AS REPORT NAME. RD IGNORED.

Programmer Response: Probable user error. Correct formation of report-name before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1173I-E GROUP INDICATE IS ILLEGAL FOR THIS REPORT GROUP. IGNORED.

Programmer Response: Probable user error. Remove GROUP INDICATE clause from all report groups except DETAIL before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1174I-E NO LINE CLAUSE SPECIFIED IN PRECEDING REPORT GROUP. NO OUTPUT GENERATED.

Programmer Response: Probable user error. For each report group, specify a LINE clause either at the report group level or prior to or for the first elementary item in the line before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1175I-E DATA-NAME FOR THIS REPORT GROUP IS NOT UNIQUE. SKIPPING TO NEW 01, RD, OR SECTION.

Programmer Response: Probable user error. Ensure that each report group data-name is a unique 01-level item before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1178I-E RESET CLAUSE SPECIFIED, AND IS EITHER ILLEGAL FOR THIS REPORT GROUP, OR ELEMENTARY ITEM DOES NOT CONTAIN A SUM CLAUSE. CLAUSE IGNORED.

Programmer Response: Probable user error. Ensure that the RESET clause is used in conjunction with the SUM clause and is associated with a CF report group before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1179I-E COLUMN NUMBER ILLEGAL. ASSUME COLUMN 1.

Programmer Response: Probable user error. Ensure that the column number does not exceed the record size before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1180I-W SYNTAX OF COMMENT IS INCORRECT. SKIPPING TO NEXT CLAUSE.

Programmer Response: Probable user error. Correct syntax of invalid comment before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1189-W 'TO' PORTION OF APPLY CORE-INDEX CLAUSE IGNORED.

Programmer Response: Probable user error. Correct syntax of clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF1190-E 'OF FILENAME' PORTION OF RERUN CLAUSE MISSING. RERUN IGNORED.

Programmer Response: Probable user error. Supply missing portion of clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2001I-W OPEN OPTION IS ILLEGAL FOR THIS ACCESS METHOD.

Programmer Response: Probable user error. Ensure that I-O option is not specified for direct or relative files with sequential access before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2002I-W DEVICE MUST BE DIRECT-ACCESS FOR OPEN I-O.

Programmer Response: Probable user error. If the OPEN statement specifies the I-O option, you must specify either UT or DA in the system-name portion of the ASSIGN clause. Change the source statements as necessary, and then recompile.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2003I-W DEVICE MUST BE DIRECT-ACCESS FOR THIS ACCESS METHOD.

Programmer Response: Probable user error. Ensure that device-class field of system-name in ASSIGN is DA before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2004I-C RECORDING MODE OPTION ILLEGAL FOR THIS ACCESS METHOD. CLAUSE IGNORED.

Programmer Response: Probable user error. Change recording mode statement to comply with sequential or random access method before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2005I-W 'APPLY RECORD OVERFLOW' CLAUSE ILLEGAL FOR THIS ACCESS METHOD. CLAUSE IGNORED.

Programmer Response: Probable user error. Check to see that a sequential access method or a direct access method with fixed-length records is specified.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2006I-E BLOCK CONTAINS CLAUSE WITH RECORDING MODE 'U' ILLEGAL. CLAUSE IGNORED.

Programmer Response: Probable user error. Either change recording mode or delete the BLOCK CONTAINS clause from the source program before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2007I-W 'APPLY REORG-CRITERIA' CLAUSE ILLEGAL FOR THIS ACCESS METHOD CLAUSE IGNORED.

Programmer Response: Probable user error. Ensure that the APPLY REORG-CRITERIA option is specified only for indexed files accessed randomly before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2008I-E APPLY WRITE-ONLY ILLEGAL FOR THIS ACCESS METHOD. IGNORED.

Programmer Response: Probable user error. If the APPLY WRITE-ONLY clause is specified, check to see that a standard sequential file with blocked V-mode records is being created. If not, change the source statement(s) necessary; then recompile.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2009I-W LABEL RECORDS OPTION INCOMPATIBLE WITH DEVICE TYPE.

Programmer Response: Probable user error. If you indicated UR as the device class in the ASSIGN statement, you must

specify the OMITTED option in the LABEL RECORDS clause. Change the source statement(s) as necessary, and then recompile.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2010I-E OBJECT OF REDEFINES CLAUSE IS OCCURS DEPENDING ON SUBJECT. REDEFINES IGNORED.

Programmer Response: Probable user error. Delete DEPENDING ON option from OCCURS clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2011I-E AN INDEX DATA ITEM MAY NOT BE A CONDITIONAL VARIABLE. 88 DISCARDED.

Programmer Response: Probable user error. Depending on the logic of your program, either supply appropriate level numbers or remove level-88 items before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2012I-E INDEX NAMES AND/OR KEYS IGNORED FOR TABLE WITH ILLEGAL SUBJECT.

Programmer Response: Probable user error. Supply valid subject for table before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2013I-E TRACK-LIMIT CLAUSE ILLEGAL FOR THIS ACCESS METHOD. CLAUSE IGNORED.

Programmer Response: Probable user error. Either delete the TRACK-LIMIT clause from the statement indicated or specify another access method; 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.

IKF2014I-W TRACK-LIMIT SPEC'D AND FILE NOT OPENED AS OUTPUT. CLAUSE IGNORED.

Programmer Response: Probable user error. For the TRACK-LIMIT clause to be valid, you must have opened the file as OUTPUT. Change the source statement(s) as necessary, and then recompile.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2015I-W 'RESERVE ALTERNATE AREA' CLAUSE TREATED AS COMMENTS FOR THIS ACCESS METHOD.

Programmer Response: Probable user error. For the 'RESERVE ALTERNATE AREA' clause to be valid, the access method must

be sequential. Correct the access method or remove the RESERVE clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2016I-W DATA RECORD SIZE IS VARIABLE, BUT 'RECORDING MODE IS F'

Programmer Response: Probable user error. Ensure that the record is associated with a valid FD; then correct the RECORDING MODE clause or record description before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2017I-E IF THE SUBJECT OF AN INDEXED BY CLAUSE IS AN ELEMENTARY ITEM ONLY THAT ITEM ITSELF MAY BE SPECIFIED IN THE KEY CLAUSE. REST OF KEYS DISCARDED.

Programmer Response: Probable user error. Remove all keys from ASCENDING/DESCENDING KEY option except subject of INDEXED BY clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2018I-E OBJECT OF RENAMES CLAUSE WAS NOT FOUND OR NON-UNIQUE IN LOGICAL RECORD.

Programmer Response: Probable user error. Supply a data description entry for the data-name being used as the object of the RENAMES clause, delete or qualify a duplicate use of the same data-name, or correct a misspelled data-name before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2019I-W BLOCK CONTAINS CLAUSE TREATED AS COMMENTS FOR THIS ACCESS METHOD. CLAUSE IGNORED.

Programmer Response: Probable user error. Ensure that if BLOCK CONTAINS 0 has been specified, the access method is sequential before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2020I-C TRACK-AREA CLAUSE ILLEGAL FOR THIS ACCESS METHOD. CLAUSE IGNORED.

Programmer Response: Probable user error. Delete TRACK-AREA clause if the desired access method for the file is sequential, and recompile if necessary. Otherwise, specify ACCESS MODE IS RANDOM before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2021I-C PICTURE DUPLICATION FACTOR TRUNCATED TO 5 SIGNIFICANT DIGITS.

Programmer Response: Probable user error. Correct picture duplication factor before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2022I-E THE OBJECT OF THE RENAMES OR RENAMES THRU CLAUSE CANNOT BE AN 01, 66, 77 OR 88. STATEMENT DISCARDED.

Programmer Response: Probable user error. Correct the object of the RENAMES or RENAMES THRU clause or its level number before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2023I-E AN ***** KEY WAS NOT SPECIFIED FOR THIS FILE.

Programmer Response: Probable user error. Supply indicated key for file or check that the desired combination of organization and access method has been specified before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2024I-E ***** KEY IS ILLEGAL FOR THIS ACCESS METHOD. CLAUSE IGNORED.

Programmer Response: Probable user error. Remove indicated key clause or check that the desired file organization has been specified in the ASSIGN clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2025I-E WRITE ADVANCING OR POSITIONING OPTION WAS SPEC'D AND FILE WAS NOT OPENED OUTPUT OR ACCESS METHOD NOT STANDARD SEQUENTIAL. WRITES WILL NOT HAVE AFTER ADVANCING OPTION FOR THIS FILE.

Programmer Response: Probable user error. Remove AFTER ADVANCING option from write statement.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2026I-W TOTALING AREA, TOTALED AREA IGNORED FOR FILE WITH RECORDING MODE S.

Programmer Response: Probable user error. Check to be sure that TOTALING and TOTALED AREA has not been specified in the LABEL RECORDS clause for a file with S-mode records. Correct the LABEL RECORDS clause or recording mode and recompile.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2027I-C APPLY CORE-INDEX ILLEGAL FOR THIS ACCESS METHOD. CLAUSE IGNORED.

Programmer Response: Probable user error. Remove APPLY CORE-INDEX clause or, if random access is desired, correct ACCESS MODE clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2028I-C REORG-CRITERIA DATA-NAME NOT FOUND OR NON-UNIQUE. CLAUSE IGNORED.

Programmer Response: Probable user error. Check the source statement to see if the data-name has been omitted or misspelled. Ensure that a unique data-name is present before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2029I-C FIRST NON 77, 88 ITEM IN SECTION IS NOT AN 01. THIS ITEM WAS CHANGED TO 01.

Programmer Response: Probable user error. Correct entry to ensure that a level-01 entry precedes subsequent levels of data description before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2030I-C 77 ITEM PRECEDED BY AN 01-49 ITEM OR 77 IN FILE SECTION. 77 CHANGED TO 01.

Programmer Response: Probable user error. Change 77 to a valid level number or rearrange items in Working-Storage Section before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2031I-C 88 ITEM MUST BE PRECEDED BY 01-49 OR 77 ITEM. 88 CHANGED TO 01.

Programmer Response: Probable user error. Correct entry so that condition-name (88) is subordinate to a conditional variable with a valid level number before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2032I-E 88 ITEM CONTAINED A CLAUSE OTHER THAN VALUE CLAUSE. CLAUSE DELETED.

Programmer Response: Probable user error. Delete clauses other than VALUE from condition-name (88) entry or correct level-number of entry before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2033I-C ITEM'S USAGE INCOMPATIBLE WITH USAGE OF GROUP IT BELONGS TO. USAGE CHANGED TO GROUP'S USAGE.

Programmer Response: Probable user error. Correct USAGE clause on group or elementary level before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2034I-E GROUP ITEM HAS PICTURE CLAUSE. CLAUSE DELETED.

Programmer Response: Probable user error. Delete PICTURE clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2035I-E GROUP ITEM HAS BLANK WHEN ZERO CLAUSE. CLAUSE DELETED.

Programmer Response: Probable user error. Delete BLANK WHEN ZERO clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2036I-E GROUP ITEM HAS JUSTIFIED CLAUSE. CLAUSE DELETED.

Programmer Response: Probable user error. Delete JUSTIFIED clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2037I-E BLANK WHEN ZERO CLAUSE USED INCORRECTLY. CLAUSE IGNORED.

Programmer Response: Probable user error. Correct syntax of BLANK WHEN ZERO clause or check compatibility of clause with data type of item being described before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2038I-W TOTALING AREA, TOTALED AREA IGNORED FOR THIS ACCESS METHOD.

Programmer Response: Probable user error. Check to be sure that TOTALING and TOTALED AREA are used only for standard sequential files. Correct the LABEL RECORDS clause or the organization field of system-name before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2039I-C PICTURE CONFIGURATION ILLEGAL. PICTURE CHANGED TO 9 UNLESS USAGE IS 'DISPLAY-ST', THEN L(6)BDZ9BDZ9.

Programmer Response: Probable user error. Correct PICTURE configuration before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2040I-E JUSTIFIED CLAUSE SPEC'D FOR NON-ALPHABETIC OR NON-ALPHANUMERIC ITEM. CLAUSE DELETED.

Programmer Response: Probable user error. Delete JUSTIFIED clause or change PICTURE to alphabetic or alphanumeric before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2041I-E CONDITION NAME UNDER GROUP HAS VALUE CLAUSE THAT IS NUMERIC. 88 DISCARDED.

Programmer Response: Probable user error. Change group item's usage to ensure that values associated with condition-names refer to a conditional variable whose USAGE IS DISPLAY before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2042I-E THIS ITEM CAUSES OVER 3 LEVELS OF SUBSCRIPTING. OCCURS CLAUSE DROPPED FOR THIS ITEM.

Programmer Response: Probable user error. Delete OCCURS clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2043I-E 01 OR 77 LEVEL HAS AN OCCURS CLAUSE. CLAUSE DELETED.

Programmer Response: Probable user error. Delete OCCURS clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2044I-E DUPLICATE SD. IGNORED.

Programmer Response: Probable user error. Delete duplicate or correct misspelled sort-file-name before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2045I-E REPORT CONTROL NAME UNDEFINED.

Programmer Response: Probable user error. Define the identifier specified in the CONTROL clause in the File, Working-Storage, or Linkage Section before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2046I-E REPORT CONTROL NAME NOT FIXED LENGTH.

Programmer Response: Probable user error. Correct the data description entry for the report control name before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2047I-E MORE THAN 12 INDEX NAMES SPECIFIED FOR TABLE. FIRST 12 ACCEPTED.

Programmer Response: Probable user error. Delete all index-names in excess of 12 before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2048I-E NONSTANDARD USER LABEL SIZE GREATER THAN 4095 BYTES.

Programmer Response: Probable user error. Ensure that the description of the nonstandard label (data-name-1 in the LABEL RECORDS clause) does not exceed a length of 4096 bytes. Correct the data description as appropriate and recompile.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2049I-C NO OPEN CLAUSE FOUND FOR FILE.

Programmer Response: Probable user error. Supply valid or missing OPEN statement for file before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2050I-W BLOCK SIZE GREATER THAN 32760. WARNING - CANNOT WRITE BLOCK OF THIS SIZE.

Programmer Response: Probable user error. Check the block size specified in either the BLOCK CONTAINS clause or the record description. Ensure that the block size is less than 32,760 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.

IKF2051I-W TRACK-AREA DATA-NAME IS NON-UNIQUE OR UNDEFINED. CLAUSE IGNORED.

Programmer Response: Probable user error. Check the source program to be sure that the data-name has been defined once and only once. 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.

IKF2052I-E MORE THAN 12 KEYS SPECIFIED FOR TABLE. FIRST 12 ACCEPTED.

Programmer Response: Probable user error. Adjust block size for file to a size compatible with the device type before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2053I-E NOMINAL KEY IS NON-UNIQUE OR UNDEFINED. CLAUSE IGNORED.

Programmer Response: Probable user error. Check the source program to see if the NOMINAL KEY has been defined once and only once. 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.

IKF2054I-W NOMINAL KEY PICTURE OR USAGE NOT LEGAL.

Programmer Response: Probable user error. Ensure that the USAGE of the NOMINAL KEY is neither DISPLAY, COMP-1 nor COMP-2. If an indexed file is being processed, the PICTURE clause can not represent a length greater than 225. If a relative file is being processed, the PICTURE clause must specify a binary length of 8 integers. Make necessary corrections before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2055I-C STERLING NON-REPORT PICTURE - SIGN IN POUND FIELD MUST BE ON HI OR LO ORDER DIGIT. PICTURE REPLACED BY 9D8D7.

Programmer Response: Probable user error. Correct PICTURE clause for sterling non-report item before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2056I-C STERLING NON-REPORT PICTURE - 9 IN ILLEGAL POSITION. PICTURE REPLACED BY 9D8D7.

Programmer Response: Probable user error. Correct PICTURE clause for sterling non-report item before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2057I-C STERLING NON-REPORT PICTURE - SIGN IN SHILLING FIELD ILLEGAL. PICTURE REPLACED BY 9D8D7.

Programmer Response: Probable user error. Correct PICTURE clause for sterling non-report before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2058I-C STERLING NON-REPORT PICTURE - 8 IN ILLEGAL POSITION. PICTURE REPLACED BY 9D8D7.

Programmer Response: Probable user error. Correct PICTURE clause for sterling non-report item before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2059I-C STERLING NON-REPORT PICTURE - SIGN IN PENCE FIELD ILLEGAL. PICTURE REPLACED BY 9D8D7.

Programmer Response: Probable user error. Correct PICTURE clause for sterling non-report item before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2060I-C STERLING NON-REPORT PICTURE - 6 OR 7 IN ILLEGAL POSITION. PICTURE REPLACED BY 9D8D7.

Programmer Response: Probable user error. Correct PICTURE clause for sterling non-report item before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2061I-C STERLING NON-REPORT PICTURE - USAGE NOT DISPLAY-ST. PICTURE REPLACED BY 9(1).

Programmer Response: Probable user error. Specify USAGE IS DISPLAY-ST before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2062I-C STERLING NON-REPORT PICTURE - V IN ILLEGAL POSITION. PICTURE REPLACED BY 9D8D7.

Programmer Response: Probable user error. Correct PICTURE clause for sterling non-report item before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2063I-C STERLING NON-REPORT PICTURE - S IN ILLEGAL POSITION. PICTURE REPLACED BY 9D8D7.

Programmer Response: Probable user error. Correct PICTURE clause for sterling non-report item before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2064I-C STERLING NON-REPORT PICTURE - DIGIT LENGTH GT 18. PICTURE REPLACED BY 9D8D7.

Programmer Response: Probable user error. Correct PICTURE clause for sterling non-report item before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2065I-C STERLING NON-REPORT PICTURE - SHILLING FIELD GT 2. PICTURE REPLACED BY 9D8D7.

Programmer Response: Probable user error. Check the programmer-defined shilling field. If it includes more than two characters, redefine this field. 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.

IKF2066I-C STERLING NON-REPORT PICTURE - PENCE FIELD GT 2. PICTURE REPLACED BY 9D8D7.

Programmer Response: Probable user error. Correct PICTURE clause for sterling non-report item before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2067I-C STERLING NON-REPORT PICTURE - NO POUND SEPARATOR. PICTURE REPLACED BY 9D8D7.

Programmer Response: Probable user error. Correct PICTURE clause for sterling non-report item before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2068I-C ONLY THE RENAMES CLAUSE MAY BE SPECIFIED FOR A LEVEL 66 ENTRY. CLAUSE IGNORED.

Programmer Response: Probable user error. Delete all clauses except RENAMES clause for level-66 item and recompile if necessary.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2069I-C NUMERIC PICTURE - SIGN IN ILLEGAL POSITION. PICTURE REPLACED BY 9(1).

Programmer Response: Probable user error. Correct PICTURE clause for item before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2070I-C NUMERIC PICTURE - P IN ILLEGAL POSITION. PICTURE REPLACED BY 9(1).

Programmer Response: Probable user error. Correct PICTURE clause for item before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2071I-C NUMERIC PICTURE - V IN ILLEGAL POSITION. PICTURE REPLACED BY 9(1).

Programmer Response: Probable user error. Correct PICTURE clause for item before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2072I-C NUMERIC PICTURE - NO 9 IN PICTURE. PICTURE REPLACED BY 9(1).

Programmer Response: Probable user error. Correct PICTURE clause for item or change USAGE clause to match PICTURE clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2073I-C NUMERIC PICTURE - P ENCLOSED BY 9'S. PICTURE REPLACED BY 9(1).

Programmer Response: Probable user error. Correct PICTURE clause for item before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2074I-D COMPILER ERROR. COMPILATION WILL NOT BE COMPLETE.

Explanation: Minor code for RENAMES entry is invalid.

Programmer Response: The compiler has reached a point in its processing where it is unable to continue. If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2075I-C NUMERIC PICTURE - DIGIT LENGTH GT 18. PICTURE REPLACED BY 9(1).

Programmer Response: Probable user error. Correct PICTURE clause for item before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2076I-C NUMERIC PICTURE - DIGIT LENGTH + SCALE GT 18. PICTURE REPLACED BY 9(1).

Programmer Response: Probable user error. Correct PICTURE clause for item before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2077I-C EXTERNAL FLOATING-POINT PICTURE - USAGE NOT DISPLAY. PICTURE CHANGED TO 9.

Programmer Response: Probable user error. Supply USAGE IS DISPLAY clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2078I-W EXTERNAL FLOATING-POINT PICTURE - MORE THAN 1 SIGN. CHANGED TO 1.

Programmer Response: Probable user error. Correct PICTURE clause for item and recompile if necessary.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2079I-C EXTERNAL FLOATING-POINT PICTURE - SIGN IN ILLEGAL POSITION. PICTURE CHANGED TO +9.E+99.

Programmer Response: Probable user error. Correct PICTURE clause for item before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2080I-C EXTERNAL FLOATING-POINT PICTURE - SIGN MISSING. ASSUME MINUS SIGN.

Programmer Response: Probable user error. Correct PICTURE clause for item before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2081I-C EXTERNAL FLOATING-POINT PICTURE - REQUIRED CHARACTER BEFORE EXPONENT MISSING. PICTURE CHANGED TO +9.E+99.

Programmer Response: Probable user error. Correct PICTURE clause for item before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2082I-W EXTERNAL FLOATING-POINT PICTURE - NO DECIMAL-POINT IN MANTISSA. ASSUME IMPLIED V.

Programmer Response: Probable user error. Correct PICTURE clause for item and recompile if necessary.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2083I-C EXTERNAL FLOATING-POINT PICTURE - MANTISSA LENGTH GT 16. PICTURE CHANGED TO +9.E+99.

Programmer Response: Probable user error. Correct PICTURE clause for item before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2084I-C EXTERNAL FLOATING-POINT PICTURE - TOTAL LENGTH GT 21. PICTURE CHANGED TO +9.E+99.

Programmer Response: Probable user error. Correct PICTURE clause for item before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2085I-C EXTERNAL FLOATING-POINT PICTURE - EXPONENT LENGTH NOT 2 DIGITS. ASSUME 2 DIGITS.

Programmer Response: Probable user error. Correct PICTURE clause for item before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2086I-C NUMERIC EDITED PICTURE - TWO FIXED DOLLAR SIGNS, +, - OR FIXED AND FLOATING DOLLAR SIGN. PICTURE REPLACED BY 9(1).

Programmer Response: Probable user error. Correct PICTURE clause for item before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2087I-W TOTALING AREA SIZE GREATER THAN 4095 BYTES. 4095 BYTES ASSUMED.

Programmer Response: Probable user error. Ensure TOTAL AREA size does not exceed 4095 bytes.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2089I-C NUMERIC EDITED PICTURE - 9, Z OR * PRECEDES FLOATING STRING. PICTURE REPLACED BY 9(1).

Programmer Response: Probable user error. Correct PICTURE clause for item before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2090I-C NUMERIC EDITED PICTURE - P IN ILLEGAL POSITION. PICTURE REPLACED BY 9(1).

Programmer Response: Probable user error. Correct PICTURE clause for item before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2091I-C NUMERIC EDITED PICTURE - TWO DIFFERENT FLOATING STRING CHARACTERS. PICTURE REPLACED BY 9(1).

Programmer Response: Probable user error. Check the floating string characters specified. If more than one character is indicated, correct the source statement to include only the one desired. Then recompile.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2092I-C NUMERIC EDITED PICTURE - Z AND * IN PICTURE. PICTURE REPLACED BY 9(1).

Programmer Response: Probable user error. Correct PICTURE clause for item before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2093I-C NUMERIC EDITED PICTURE - 9 PRECEDES * OR Z. PICTURE REPLACED BY 9(1).

Programmer Response: Probable user error. Correct the order in the PICTURE clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2094I-C NUMERIC EDITED PICTURE - FLOATING STRING PRECEDES * OR Z. PICTURE REPLACED BY 9(1).

Programmer Response: Probable user error. Correct order of PICTURE clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2095I-W NUMBER OF CHARACTERS IN BLOCK CONTAINS CLAUSE ON FORMAT F FILE NOT MULTIPLE OF RECORD LENGTH. ASSUMING BLOCK SIZE TO BE CLOSEST MULTIPLE.

Programmer Response: Probable user error. Check the value in the BLOCK CONTAINS clause for a multiple of the record length. If necessary, respecify this value and recompile.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2096I-C DECIMAL POINT MAY ONLY APPEAR ONCE IN A PICTURE CHARACTER STRING. PICTURE REPLACED BY 9(1).

Programmer Response: Probable user error. Delete all but one decimal point from PICTURE clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2097I-C NUMERIC EDITED PICTURE - DECIMAL POINT OR V CONTRADICTORY TO P. PICTURE REPLACED BY 9(1).

Programmer Response: Probable user error. Correct PICTURE clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2098I-C INDEXED BY AND/OR KEY CLAUSE IS ILLEGAL FOR ITEM SUBORDINATE TO GROUP THAT HAS OCCURS BUT NO INDEXED BY CLAUSE. CLAUSE IGNORED.

Programmer Response: Probable user error. Supply INDEXED BY clause on group item before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2099I-C NUMERIC EDITED PICTURE - CR OR DB AND SIGN BOTH USED. PICTURE REPLACED BY 9(1).

Programmer Response: Probable user error. Delete duplicate sign symbol before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2100I-C NUMERIC EDITED PICTURE - CR OR DB NOT LAST TWO CHARACTERS IN PICTURE. PICTURE REPLACED BY 9(1).

Programmer Response: Probable user error. Correct PICTURE clause so that either CR or DB are the last two characters in the string before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2101I-C NUMERIC EDITED PICTURE - SIGN IS NOT FIRST OR LAST CHARACTER IN PICTURE. PICTURE REPLACED BY 9(1).

Programmer Response: Probable user error. Make sign leftmost or rightmost character in PICTURE clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2102I-C NUMERIC EDITED PICTURE - NUMERIC CHARACTERS AFTER DECIMAL POINT ARE NOT THE SAME. PICTURE REPLACED BY 9(1).

Programmer Response: Probable user error. Supply valid numeric characters as suppression symbols after the decimal point before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2103I-C NUMERIC EDITED PICTURE - TOTAL LENGTH GT 127. PICTURE REPLACED BY 9(1).

Programmer Response: Probable user error. Reduce total length to 127 or less before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2104I-C NUMERIC EDITED PICTURE - NUMERIC LENGTH GT 18. PICTURE REPLACED BY 9(1).

Programmer Response: Probable user error. Reduce the number of digit positions represented to 18 or less before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2105I-E ONLY ONE KEY MAY BE SPECIFIED IF SUBJECT OF TABLE IS A KEY. REST OF KEYS DISCARDED.

Programmer Response: Probable user error. Eliminate all keys except table subject before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2106I-E THE RENAMES CLAUSE MUST BE THE LAST ENTRY IN A LOGICAL RECORD. SKIPPING TO NEXT LEVEL, SECTION OR DIVISION.

Programmer Response: Probable user error. Correct placement of level-66 item before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2107I-W EDITED PICTURE - USAGE NOT DISPLAY. PICTURE CHANGED TO 9.

Programmer Response: Probable user error. Remove USAGE clause from item or from group to which it belongs before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2108I-E KEYS IGNORED FOR ITEM WITH NO INDEXED BY CLAUSE.

Programmer Response: Probable user error. Supply INDEXED BY clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2109I-C TRACK-AREA INTEGER NOT MULTIPLE OF 8. ROUNDED DOWN TO MULTIPLE OF 8.

Programmer Response: Probable user error. Ensure TRACK-AREA integer is multiple of 8.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2110I-C APPLY WRITE-ONLY CLAUSE ILLEGALLY USED. IGNORED.

Programmer Response: Probable user error. Check the source statement identified for any of the following error possibilities:

- That the recording mode is not V.
- That the files are not standard sequential.
- That at least one WRITE statement associated with the file does not use the FROM identifier option.
- That either at least one subfield of the identifier specified in the FROM option is referred to in a procedure statement or the identifier is the object of an OCCURS DEPENDING ON clause.
- That the WRITE statement is used when the same file is opened for I-O.

If any of these errors are present, correct the source statement and recompile.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2111I-C TOTALING AREA SIZE VARIABLE. MAXIMUM SIZE ASSUMED.

Programmer Response: Probable user error. Ensure that the TOTALING AREA data-name represents a fixed-length item and then recompile.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2112I-C ***** IN KEY CLAUSE NOT DEFINED.

Programmer Response: Probable user error. Ensure that the data-name specified in the key clause is defined in the appropriate section of the Data Division, and 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.

IKF2113I-W ITEM WITH USAGE OF COMPUTATIONAL-1 OR COMPUTATIONAL-2 HAS PICTURE CLAUSE. CLAUSE IGNORED.

Programmer Response: Probable user error. Delete PICTURE clause and recompile if necessary, or correct USAGE clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2114I-E ONLY THE SYNCHRONIZED CLAUSE IS ALLOWED FOR A USAGE IS INDEX ITEM. CLAUSE IGNORED.

Programmer Response: Probable user error. Delete illegal clause(s) before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2115I-E GROUP ITEM SIZE IS GT 32K.

Programmer Response: Probable user error. Ensure that total length of group item does not exceed 32K, and then recompile.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2116I-E FIXED LENGTH GROUP ITEM IN WORKING-STORAGE OR LINKAGE SECTION IS GT 131K.

Programmer Response: Probable user error. Ensure that fixed-length group item does not exceed 131K, and recompile the program.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2117I-E INVALID REPORT CHARACTER. PICTURE CHANGED TO 9.

Programmer Response: Probable user error. Supply valid numeric edited character before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2118I-C LENGTH OF REDEFINES SUBJECT GREATER THAN LENGTH OF REDEFINES OBJECT. SUBJECT LENGTH USED.

Programmer Response: Probable user error. Ensure that length of redefined item is greater than or equal to length of item that redefines it before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2119I-E VALUE CLAUSE SPECIFIED FOR AN ITEM IN A REDEFINES GROUP. CLAUSE IGNORED.

Programmer Response: Probable user error. Delete VALUE clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2120I-E OBJECT OF REDEFINES CLAUSE UNDEFINED OR ILLEGAL. CLAUSE IGNORED.

Programmer Response: Probable user error. Correct object of REDEFINES clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2121I-W SUBJECT OF REDEFINES IS VARIABLE LENGTH.

Programmer Response: Probable user error. Remove DEPENDING ON option from OCCURS clause which describes subject of REDEFINES clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2122I-E REDEFINES SUBJECT LEVEL NUMBER NOT EQUAL TO REDEFINES OBJECT LEVEL NUMBER OR OBJECT NOT IMMEDIATELY PRECEDING SUBJECT. CLAUSE IGNORED.

Programmer Response: Probable user error. Correct placement or level number of entry containing REDEFINES clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2123I-W OBJECT OF REDEFINES IS SUBSCRIPTED.

Programmer Response: Probable user error. Remove OCCURS clause from description of object of REDEFINES or use method other than REDEFINES statement to achieve desired purpose.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2124I-E OBJECT OF REDEFINES IS VARIABLE LENGTH GROUP ITEM. REDEFINES CLAUSE IGNORED.

Programmer Response: Probable user error. Correct definition of REDEFINES clause or use method other than REDEFINES statement to achieve desired purpose.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2125I-W VALUE CLAUSE TREATED AS COMMENTS FOR ITEMS IN FILE SECTION AND LINKAGE SECTION.

Programmer Response: Probable user error. Remove VALUE clause from items other than level-88 in File or Linkage Section and recompile if necessary.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2126I-C VALUE CLAUSE LITERAL TOO LONG. TRUNCATED TO PICTURE SIZE.

Programmer Response: Probable user error. Depending upon the logic of the program, either correct length of PICTURE or of the literal specified in the VALUE clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2127I-C NUMERIC VALUE CLAUSE SPECIFIED FOR GROUP ITEM. CLAUSE IGNORED.

Programmer Response: Probable user error. Specify a nonnumeric literal or a figurative constant in the VALUE clause or remove VALUE clause from group level and define initial values at the elementary level before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2128I-C VALUE CLAUSE LITERAL DOES NOT CONFORM TO PICTURE. CHANGED TO BLANKS.

Programmer Response: Probable user error. Correct PICTURE clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2129I-C VALUE CLAUSE LITERAL DOES NOT CONFORM TO PICTURE. CHANGED TO ZERO.

Programmer Response: Probable user error. Correct PICTURE clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2130I-E ITEM CANNOT HAVE VALUE CLAUSE. CLAUSE IGNORED.

Programmer Response: Probable user error. Delete VALUE clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2131I-E RECORD KEY UNDEFINED OR NON-UNIQUE. KEY IGNORED.

Programmer Response: Probable user error. Check the source program for the possibility that either no data-name has been specified for the RECORD KEY or the data-name has been defined more than once. Correct the source statement(s) as necessary, and then recompile.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2132I-E RECORD KEY LENGTH GREATER THAN 255 BYTES. USING FIRST 255 BYTES.

Programmer Response: Probable user error. Ensure that RECORD KEY length does not exceed 255 bytes.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2133I-W LABEL RECORDS CLAUSE MISSING. DD CARD OPTION WILL BE TAKEN.

Programmer Response: Probable user error. Check to see whether the LABEL RECORDS clause has been left out or if another compiler error has caused the compiler to ignore this card. Correct the source program and recompile.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2134I-C VALUE FOR SCALING CHARACTER SHOULD BE ZERO. CHANGED TO ZERO.

Programmer Response: Probable user error. Change value to zero before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2135I-C RECORD DESCRIPTION VARIABLE FOR INDEXED OR RELATIVE FILE. ASSUMED FIXED SIZE TAKEN FROM MAXIMUM RECORD SIZE.

Programmer Response: Probable user error. Correct record descriptions associated with the file so that each is the same length before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2136I-E NOMINAL KEY LENGTH FOR INDEXED FILES GREATER THAN 255 BYTES. KEY IGNORED.

Programmer Response: Probable user error. Correct PICTURE clause of data-name specified in the NOMINAL KEY clause to reflect a length of 255 bytes or less before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2137I-E DATANAME-3 IS SUBORDINATE TO DATANAME-2 IN THE RENAMES THRU CLAUSE. STATEMENT DISCARDED.

Programmer Response: Probable user error. Correct RENAMES THRU clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2138I-W ITEM LENGTH GREATER THAN 32K. TRUNCATED TO 32K.

Programmer Response: Probable user error. Ensure that item length does not exceed 32K.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2139I-W APPLY OVERFLOW CLAUSE ILLEGAL FOR V OR U MODE DIRECT FILES. CLAUSE IGNORED.

Programmer Response: Probable user error. Delete APPLY OVERFLOW clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2140I-E VALUE CLAUSE SPECIFIED ON BOTH GROUP AND ELEMENTARY ITEM OR SUBORDINATE GROUP. SECOND ITEM'S VALUE CLAUSE IGNORED.

Programmer Response: Probable user error. Correct record description to ensure that a VALUE clause does not appear both on the group level and on a level subordinate to the group level before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2141I-C LENGTH OF LITERAL IS MORE OR LESS THAN LENGTH OF GROUP. LENGTH OF LITERAL ASSUMED.

Programmer Response: Probable user error. Either alter the length of the group by correcting PICTURE clause specifications or respecify VALUE clause for group before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2142I-E ALPHABETIC OR ALPHANUMERIC ITEM HAS ILLEGAL USAGE. PICTURE CHANGED TO 9.

Programmer Response: Probable user error. Correct either PICTURE or USAGE clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2143I-W STERLING NON-REPORT PICTURE - MORE THAN ONE V OR S, ASSUMED ONE.

Programmer Response: Probable user error. Correct PICTURE clause for sterling non-report item and recompile if necessary.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2144I-W NUMERIC PICTURE - MORE THAN ONE V OR S. ASSUMED ONE.

Programmer Response: Probable user error. Correct PICTURE clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2145I-E ALPHABETIC OR ALPHANUMERIC ITEM LENGTH GREATER THAN 32767. TRUNCATED TO 32767.

Programmer Response: Probable user error. Correct PICTURE clause for item before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2146I-W RECORD SIZE IN RECORD-CONTAINS CLAUSE DISAGREES WITH COMPUTED RECORD SIZE. USING MAXIMUM COMPUTED SIZE.

Programmer Response: Probable user error. Correct RECORD CONTAINS clause and recompile if necessary.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2147I-W 'RECORD CONTAINS INTEGER-1' IS NOT MINIMUM.

Programmer Response: Probable user error. Ensure 'RECORD CONTAINS integer-1' is minimum before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2148I-W ON AN 01(77) COPY LIBRARY-NAME CLAUSE, LIBRARY DID NOT HAVE AN 01(77) AS FIRST CARD.

Programmer Response: Probable user error. Correct first entry of library member before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2149I-E VALUE CLAUSE SPECIFIED FOR ITEM WITH OCCURS OR FOR ITEM SUBORDINATE TO AN ITEM WITH OCCURS. CLAUSE IGNORED.

Programmer Response: Probable user error. Delete VALUE clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2150I-E VALUE CLAUSE SPECIFIED FOR ITEM IN VARIABLE LENGTH PORTION OF A WORKING-STORAGE RECORD. CLAUSE IGNORED.

Programmer Response: Probable user error. Delete VALUE clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2151I-C ELEMENTARY ITEMS NOT INTERNAL FLOATING-POINT MUST HAVE PICTURE. PICTURE ASSUMED 9.

Programmer Response: Probable user error. Supply PICTURE clause for item before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2152I-D COMPILER ERROR. COMPILATION WILL NOT BE COMPLETE.

Explanation: Phase 2 input is unrecognizable. Skipping to next phase.

Programmer Response: The compiler has reached a point in its processing where it is unable to continue. If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2153I-W DATA-NAME OF TRACK-AREA CLAUSE IS NOT AN ITEM OF LEVEL 01 OR 77 IN WORKING-STORAGE. CLAUSE IGNORED.

Programmer Response: Probable user error. Ensure data-name of track-area clause is a level-77 or -01 item.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2154I-C THE AREA BEING REDEFINED IS NOT IMMEDIATELY PRECEDING THE ENTRY WHICH REDEFINES IT OR THE LEVEL NUMBERS OF THE SUBJECT AND OBJECT OF THE REDEFINES ARE NOT THE SAME. THE OBJECT OF THE REDEFINES IS ASSUMED TO BE THE LAST ENTRY WITH SAME LEVEL NUMBER AS SUBJECT OF REDEFINES.

Programmer Response: Probable user error. Correct level number of subject and/or object of the REDEFINES clause, or, if correct, check placement of object of REDEFINES to ensure that it and its subordinate entries immediately precede the subject before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2155I-C ILLEGAL STERLING NON-REPORT PICTURE CHARACTER. PICTURE REPLACED BY 9D8D7.

Programmer Response: Probable user error. Correct PICTURE of sterling non-report item before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2156I-W PICTURE DOES NOT CONTAIN A SIGN. SIGN DROPPED FROM VALUE CLAUSE LITERAL.

Programmer Response: Probable user error. Include sign in PICTURE clause before recompiling, or remove sign from literal and recompile if necessary.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2157I-W NOMINAL KEY FOR A RELATIVE ORGANIZATION FILE MUST BE S9(8) COMPUTATIONAL.

Programmer Response: Probable user error. Ensure nominal key is S9(8) computational before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2158I-D OCCURS DEPENDING ON VARIABLE IS IN VARIABLE PORTION OF A RECORD. PROGRAM MAY NOT EXECUTE.

Programmer Response: Probable user error. Ensure that DEPENDING ON variable is not in variable portion of record before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2159I-C OBJECT OF REDEFINES CLAUSE NOT DEFINED. PREVIOUS 01 ASSUMED TO BE OBJECT.

Programmer Response: Probable user error. Define object of REDEFINES clause and ensure that it and its subordinate fields immediately precede the subject before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2160I-E THE OBJECT OF THE RENAMES OR RENAMES THRU CLAUSE CANNOT CONTAIN AN OCCURS OR OCCURS DEPENDING ON CLAUSE NOR MAY IT BE SUBORDINATE TO AN ITEM THAT HAS ONE OF THESE CLAUSES. STATEMENT DISCARDED.

Programmer Response: Probable user error. Supply a valid RENAMES or RENAMES THRU object before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2161I-C PICTURE INVALID. ADJACENT C DELIMITERS. ASSUMED PICTURE L(6)9BDZ9BDZ9.

Programmer Response: Probable user error. Correct PICTURE clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2162I-C PICTURE INVALID. ADJACENT D DELIMITERS. ASSUMED PICTURE L(6)9BDZ9BDZ9.

Programmer Response: Probable user error. Correct PICTURE clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2163I-C PICTURE INVALID. MORE THAN 2 DELIMITERS. ASSUMED PICTURE L(6)9BDZ9BDZ9.

Programmer Response: Probable user error. Correct PICTURE clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2164I-C PICTURE INVALID. NO STERLING DELIMITERS. ASSUMED PICTURE L(6)9BDZ9BDZ9.

Programmer Response: Probable user error. Correct PICTURE clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2165I-C PICTURE INVALID. ONLY 1 STERLING DELIMITER. ASSUME PICTURE L(6)9BDZ9BDZ9.

Programmer Response: Probable user error. Correct PICTURE clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2166I-C PICTURE INVALID. ERROR IN SHILLING FIELD. ASSUMED SHILLING PICTURE Z9B.

Programmer Response: Probable user error. Correct PICTURE clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2167I-C PICTURE INVALID. NUMBER OF POUND DIGITS EXCEEDS 15. ASSUMED PICTURE L(6)9BD.

Programmer Response: Probable user error. Correct PICTURE clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2168I-C PICTURE INVALID. ERROR IN WHOLE PENCE FIELD. ASSUMED PENCE PICTURE Z9.

Programmer Response: Probable user error. Correct PICTURE clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2169I-C PICTURE INVALID. ERROR IN DECIMAL PENCE FIELD. DECIMAL FIELD TRUNCATED.

Programmer Response: Probable user error. Correct PICTURE clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2170I-C PICTURE INVALID. ERROR IN POUND FIELD. ASSUMED POUND PICTURE L(6)9B.

Programmer Response: Probable user error. Correct PICTURE clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2171I-C PICTURE INVALID. NUMBER OF POUND DIGITS PLUS NUMBER OF PENCE DECIMAL EXCEEDS 15. DECIMAL PENCE DROPPED.

Programmer Response: Probable user error. Correct PICTURE clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2172I-C PICTURE INVALID. SIZE OF REPORT FIELD EXCEEDS 127 BYTES. ASSUMED PICTURE L(6)9BDZ9BDZ9.

Programmer Response: Probable user error. Correct PICTURE clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2173I-C PICTURE INVALID. CR OR DB NOT VALID WITH LEADING SIGN. DECIMAL FIELD TRUNCATED.

Programmer Response: Probable user error. Correct PICTURE clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2174I-C PICTURE INVALID. SIGN IN DECIMAL PENCE FIELD NOT VALID WITH LEADING SIGN. DECIMAL FIELD TRUNCATED.

Programmer Response: Probable user error. Correct PICTURE clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2175I-W TRACK-AREA EXCEEDS AND IS REDUCED TO 32,760 BYTES.

Programmer Response: Probable user error. Correct integer specification in TRACK-AREA clause and recompile if necessary.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2176I-W DATA-NAME OF TRACK-AREA CLAUSE EXCEEDS 32,767 BYTES IN LENGTH. CLAUSE IGNORED.

Programmer Response: Probable user error. Ensure data-name does not exceed 32,767 bytes before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2177I-W DATA-NAME OF TRACK-AREA CLAUSE IS NOT FIXED-LENGTH. CLAUSE IGNORED.

Programmer Response: Probable user error. Ensure data-name is fixed length before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2178I-E RECORD KEY IS NOT WITHIN FILE RECORD.

Programmer Response: Probable user error. Ensure that the data-name specified in the RECORD KEY clause is defined within the file record before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2179I-E RECORD KEY IS NOT FIXED LENGTH.

Programmer Response: Probable user error. Define the data-name specified in the RECORD KEY clause as a fixed length item before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2180I-E RECORD KEY FOR UNBLOCKED FILE INCLUDES FIRST BYTE OF RECORD.

Programmer Response: Probable user error. For an unblocked file, correct the placement of the description of the data-name specified in the RECORD CONTAINS clause so that it excludes the first byte of the record before recompiling. If blocked records are desired, add or correct the BLOCK CONTAINS clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2181I-E NOMINAL OR ACTUAL KEY IS DEFINED WITHIN THE FILE.

Programmer Response: Probable user error. Define the data-name specified in the NOMINAL KEY clause in the Working-Storage Section before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2182I-W APPLY WRITE-ONLY IS MEANINGLESS WHEN RECORDING MODE IS F OR U. CLAUSE IGNORED.

Programmer Response: Probable user error. Delete APPLY WRITE-ONLY and recompile.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2183I-W NO 01 LEVEL SD OR FD.

Programmer Response: Probable user error. Supply a valid record description entry for the FD or SD before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2184I-E VALUE CLAUSE LITERAL DOES NOT CONFORM TO PICTURE. CLAUSE IGNORED.

Programmer Response: Probable user error. Supply valid literal before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2185I-E DATA-NAME-3 EITHER PRECEDES DATA-NAME-2 OR IS DATA-NAME-2 IN THE RENAMES THRU CLAUSE. STATEMENT DISCARDED.

Programmer Response: Probable user error. Supply valid objects of RENAMES clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2186I-C PICTURE DUPLICATION FACTOR IS ZERO. ASSUMING ONE OCCURRENCE OF PICTURE CHARACTER.

Programmer Response: Probable user error. If a PICTURE duplications factor is required, supply a non-zero integer before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2187I-E OBJECT OF RENAMES CLAUSE OR RENAMES THRU CLAUSE IS NOT IN THE SAME LOGICAL RECORD. STATEMENT DISCARDED.

Programmer Response: Probable user error. Supply valid objects of the RENAMES clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2188I-C EXTERNAL FLOATING-POINT PICTURE ILLEGAL WHEN CURRENCY SIGN IS E. PICTURE CHANGED TO 9.

Programmer Response: Probable user error. Supply valid PICTURE clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2189I-W 'BLOCK CONTAINS 0' OPTION ILLEGAL FOR BISAM OR WITH 'SAME AREA' CLAUSE. 'BLOCK CONTAINS' CLAUSE IGNORED.

Programmer Response: Probable user error. Delete 'BLOCK CONTAINS 0 RECORDS' or 'SAME AREA' clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2190I-W PICTURE CLAUSE IS SIGNED, VALUE CLAUSE UNSIGNED. ASSUMED POSITIVE.

Programmer Response: Probable user error. If a negative value is intended, respecify the VALUE clause before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2191I-C THE SYNCHRONIZED CLAUSE SHOULD NOT BE SPECIFIED WHEN 88'S ARE UNDER GROUP. STATEMENT ACCEPTED AS WRITTEN.

Programmer Response: Probable user error. Remove SYNCHRONIZED clause from group with which condition-names (88's) are associated before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2192I-C ONLY USAGE IS DISPLAY SHOULD BE SPECIFIED WHEN A VALUE CLAUSE IS ASSOCIATED WITH A GROUP ITEM.

Programmer Response: Probable user error. Remove clauses other than USAGE IS DISPLAY and VALUE from group level before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2195I-E UNIT RECORD DEVICE ILLEGAL FOR RECORDING MODE S. RECORDING MODE V ASSUMED.

Programmer Response: Probable user error. Ensure unit record device is compatible with recording mode and recompile.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2199I-W ZERO SUPPRESSION CHARACTER WILL OVERRIDE BLANK WHEN ZERO CLAUSE. CLAUSE IGNORED.

Programmer Response: Probable user error. Delete 'BLANK WHEN ZERO' clause and recompile.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2200I-W INVALID ALPHANUMERIC EDITED CHARACTER. STATEMENT ACCEPTED AS WRITTEN.

Programmer Response: Probable user error. Check the source statement to be sure that use of the PICTURE character string is restricted to combinations of the symbols 'A', 'X', '9', 'B', and '0'. 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.

IKF2201I-C SYNCHRONIZED ITEM NOT ON PROPER BOUNDARY. NO ALIGNMENT PERFORMED BECAUSE STARTING ADDRESS OF THE REDEFINING ITEM WOULD HAVE TO BE CHANGED.

Programmer Response: Probable user error. Ensure synchronized item is on proper boundary before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2203I-C NUMERIC EDITED PICTURE - NO NUMERIC CHARACTERS IN PICTURE. PICTURE CHANGED TO 9(1).

Programmer Response: Probable user error. Ensure PICTURE is correct and recompile.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2223I-C PICTURE LENGTH WOULD CAUSE OVERFLOW FROM REPORT LINE AT SPECIFIED COLUMN. TRUNCATED TO AVAILABLE SIZE.

Programmer Response: Probable user error. Redefine PICTURE length for the item indicated, and 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.

IKF2224I-C 'OCCURS 0 TIMES' (WITH OR WITHOUT THE 'DEPENDING ON' OPTION) IS ILLEGAL. 1 OCCURRENCE ASSUMED.

Programmer Response: Probable user error. Correct the source so that the integer representing the number of occurrences is greater than zero and recompile.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2226I-E TOTALING AREA DATA NAME NOT FOUND.

Programmer Response: Probable user error. Ensure that the TOTALING AREA data name is defined in the COBOL source statements.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF2227I-E TOTALING AREA DATA NAME NOT DEFINED IN WORKING-STORAGE SECTION.

Programmer Response: Probable user error. Ensure that the data name associated with the TOTALING AREA option is defined in the WORKING-STORAGE Section of the COBOL source statements.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF3001I-E ***** NOT DEFINED.

Explanation: This message always appears in conjunction with another message.

Programmer Response: Probable user error. Define the indicated name before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF3002I-E ***** NOT UNIQUE.

Explanation: This message always appears in conjunction with another message.

Programmer Response: Probable user error. Remove duplication by qualification or by substituting another name for the indicated item before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF3003I-E HIGHEST LEVEL QUALIFIER ***** NOT DEFINED.

Explanation: This message always appears in conjunction with another message.

Programmer Response: Probable user error. Define the qualifier indicated before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF3004I-W QUALIFYING NAME NOT UNIQUE. DISCARDED.

Explanation: This message always appears in conjunction with another message.

Programmer Response: Probable user error. Remove duplication by substituting another name for indicated item before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF3005I-E ***** NOT A VALID QUALIFIER.

Explanation: This message always appears in conjunction with another message.

Programmer Response: Probable user error. Correct indicated name before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF3006I-E ***** NOT DEFINED AS PART OF *****.

Explanation: This message always appears in conjunction with another message.

Programmer Response: Probable user error. Define indicated name before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF3007I-W ***** NOT UNIQUELY QUALIFIED BY *****.

Explanation: This message always appears in conjunction with another message.

Programmer Response: Probable user error. Correctly qualify indicated name before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF3008I-E ***** NOT VALID AS IDENTIFIER-1 IN ***** CORRESPONDING STATEMENT.

Programmer Response: Probable user error. Correct invalid item in the indicated statement before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF3009I-E ***** NOT VALID AS IDENTIFIER-2 IN ***** CORRESPONDING STATEMENT.

Programmer Response: Probable user error. Correct identifier-2 in the indicated statement before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF3010I-W SUPERFLUOUS 'TO' IGNORED IN ***** CORRESPONDING STATEMENT.

Programmer Response: Probable user error. Remove superfluous TO from indicated statement and recompile if necessary.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF3011I-W NO CORRESPONDENCE FOUND BETWEEN IDENTIFIER-1 AND *****.

Programmer Response: Probable user error. Set up the correct correspondence between the identifiers before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF3012I-D COMPILER ERROR. COMPILATION WILL NOT BE COMPLETE.

Explanation: No matched DCB in QFILE table.

Programmer Response: The compiler has reached a point in its processing where it is unable to continue. If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF3013I-D COMPILER ERROR. COMPILATION WILL NOT BE COMPLETE.

Explanation: The dictionary pointer is less than the QVAR entry for elementary item.

Programmer Response: The compiler has reached a point in its processing where it is unable to continue. If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF3014I-D COMPILER ERROR. COMPILATION WILL NOT BE COMPLETE.

Explanation: No match has been found in QVAR table for elementary item.

Programmer Response: The compiler has reached a point in its processing where it is unable to continue. If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF3016I-D COMPILER ERROR. COMPILATION WILL NOT BE COMPLETE.

Explanation: Error in processing CORRESPONDING option.

Programmer Response: The compiler has reached a point in its processing where it is unable to continue. If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF3017I-D COMPILER ERROR. COMPILATION WILL NOT BE COMPLETE.

Explanation: Minor code is invalid.

Programmer Response: The compiler has reached a point in its processing where it is unable to continue. If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF3018I-E SPECIAL REGISTERS TIME-OF-DAY OR CURRENT-DATE MAY ONLY BE USED IN THE MOVE STATEMENT.

Programmer Response: Probable user error. Remove references to TIME-OF-DAY and CURRENT-DATE from statements other than MOVE before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF3019I-D COMPILER ERROR. COMPILATION WILL NOT BE COMPLETE.

Explanation: Invalid level found while processing glossary.

Programmer Response: The compiler has reached a point in its processing where it is unable to continue. If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF3020I-E REPORT NAME ILLEGAL AS USED. DISCARDED.

Explanation: Report name is invalid as used.

Programmer Response: Probable user error. Report-name may be specified only in the GENERATE, INITIATE, or TERMINATE statements. Remove all other references before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF3021I-C ***** NOT UNIQUE IN ITS GROUP. DISCARDED.

Programmer Response: Probable user error. Eliminate duplication of indicated item before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF3022I-E ***** NOT VALID AS IDENTIFIER-1 IN SEARCH STATEMENT.

Programmer Response: Probable user error. Correct identifier-1 before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF3023I-W ITEMS CONTAINING THE USAGE IS INDEX, REDEFINES, RENAMES OR OCCURS CLAUSES DO NOT QUALIFY AS CORRESPONDING DATANAMES.

Programmer Response: Probable user error. Make any necessary corrections before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF3024I-E NO KEYS WERE SPECIFIED FOR *****. STATEMENT DISCARDED.

Programmer Response: Probable user error. Define keys specified as identifier-1 in SEARCH statement before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF3025I-E AN ERROR WAS DETECTED PROCESSING THE KEYS FOR *****.

Programmer Response: Probable user error. For a SEARCH ALL statement, check if the KEY option appears in the OCCURS clause of identifier-1; for a SEARCH statement, ensure that INDEXED BY option is specified. 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.

IKF3026I-E DATANAME-2 OMITTED IN ***** CORRESPONDING STATEMENT.

Programmer Response: Probable user error. Supply data-name-2 in indicated statement before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF3027I-W DATANAME UNDER LABEL RECORD IS NON-UNIQUE. LAST DATA DESCRIPTION OF ***** IS ASSUMED.

Programmer Response: Probable user error. Eliminate duplicate use of data-name and recompile if necessary.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF3028I-E SPECIAL REGISTERS MAY NOT BE USED WITH ACCEPT DISPLAY OR EXHIBIT VERBS. *****.

Programmer Response: Probable user error. Check the use of special registers as operands. None may be used with either DISPLAY or EXHIBIT; only TALLY may be used with ACCEPT. Change the source statement(s) as necessary, and 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.

IKF3029I-E OCCURS DEPENDING ON OBJECT NOT DEFINED FOR *****.

Programmer Response: Probable user error. Ensure that the object of the OCCURS DEPENDING ON clause is defined within the COBOL program.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4001I-C OUTCOME OF A PRECEDING CONDITION LEADS TO NON-EXISTENT 'NEXT SENTENCE'. 'GOBACK' INSERTED.

Programmer Response: Probable user error. Add a sentence after IF statement before recompiling, or if a "NOT TRUE" evaluation of condition leads to a logical end of a program, add a GOBACK statement and recompile if necessary.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4002I-E ***** STATEMENT INCOMPLETE. STATEMENT DISCARDED.

Programmer Response: Probable user error. Complete indicated statement before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4003I-E EXPECTING NEW STATEMENT. FOUND *****. DELETING TILL NEXT VERB OR PROCEDURE NAME.

Programmer Response: Probable user error. Replace indicated item with a valid statement before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4004I-E *****/***** IS ILLEGALLY USED IN ***** STATEMENT. DISCARDED.

Programmer Response: Probable user error. Replace invalidly used items before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4005I-E ***** AND ***** VIOLATE RULE ABOUT LENGTH OF TRANSFORM OPERANDS. STATEMENT DISCARDED.

Programmer Response: Probable user error. Correct length of TRANSFORM statement operands before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4006I-C ***** STATEMENT CONTAINS UNPAIRED LEFT PARENTHESES. OUTERMOST IGNORED.

Programmer Response: Probable user error. Ensure that a corresponding right parenthesis appears for each left parenthesis before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4007I-C ***** MISSING OR MISPLACED IN ***** STATEMENT. ASSUMED IN REQUIRED POSITION.

Programmer Response: Probable user error. Supply missing or misplaced item in indicated statement and recompile if necessary.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4008I-W SUPERFLUOUS ***** FOUND IN ***** STATEMENT. IGNORED.

Programmer Response: Probable user error. Remove superfluous item from indicated statement before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4009I-E EXAMINE STATEMENT REQUIRES FIGURATIVE CONSTANT, SINGLE ALPHANUMERIC CHARACTER OR 1-DIGIT UNSIGNED NUMERIC INTEGRAL LITERAL. FOUND *****. STATEMENT DELETED.

Programmer Response: Probable user error. Correct EXAMINE statement operand before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4010I-C ***** STATEMENT CONTAINS UNPAIRED RIGHT PARENTHESES. OUTERMOST IGNORED.

Programmer Response: Probable user error. Ensure that a corresponding left parenthesis appears for each right parenthesis before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4011I-E ***** IS NOT AN ALLOWABLE CHARACTER FOR *****. STATEMENT DISCARDED.

Programmer Response: Probable user error. Correct indicated statement or option before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4012I-E COMPARISON BETWEEN TWO LITERALS IS ILLEGAL. TEST DISCARDED.

Programmer Response: Probable user error. Correct operands of comparison before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4013I-C RELATIONAL MISSING IN IF OR CONDITIONAL STATEMENT. ASSUMED EQUAL.

Programmer Response: Probable user error. Supply desired relational operator in IF or conditional statement before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4014I-E EXAMINE STATEMENT REQUIRES DATA-NAME WHOSE USAGE IS DISPLAY. FOUND *****/*****. STATEMENT DISCARDED.

Programmer Response: Probable user error. Supply valid EXAMINE statement operand before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4015I-E 'GO TO.' IS ILLEGAL UNLESS ALTERED. STATEMENT DISCARDED.

Programmer Response: Probable user error. Insert proper ALTER statement and recompile.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4016I-C OPERAND OF ***** APPEARS IN WRONG SEGMENT OF PROGRAM. STATEMENT ACCEPTED AS WRITTEN.

Programmer Response: Probable user error. Place operand of indicated statement in the proper segment before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4017I-E ELSE UNMATCHED BY CONDITION IS DISCARDED.

Programmer Response: Probable user error. Correct logic of nested IF condition before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4018I-E SET STATEMENT HAS AN ILLEGAL OPERAND BEFORE 'TO' OR INCOMPATIBLE OPERANDS. OPERAND BEFORE 'TO' DISCARDED.

Programmer Response: Probable user error. Correct SET statement operand(s) before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4019I-E *****/***** MAY NOT BE USED AS ARITHMETIC OPERAND IN ***** STATEMENT. ARBITRARILY SUBSTITUTING *****.

Programmer Response: Probable user error. Correct item in indicated statement before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4020I-C SIGN BEFORE ***** IS DISCARDED.

Programmer Response: Probable user error. Remove illegal sign from indicated item before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4021I-W MINUS SIGN FOLLOWED BY SPACE ACCEPTED AS REVERSING SIGN OF FOLLOWING LITERAL.

Programmer Response: Probable user error. Delete space after minus sign and recompile if necessary.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4022I-C EXIT MUST BE SINGLE-WORD PARAGRAPH PRECEDED BY A PROCEDURE-NAME. STATEMENT DISCARDED.

Programmer Response: Probable user error. Correct syntax of EXIT statement before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4023I-E STORE-FIELD WHEN USED IN COMPUTATION MUST BE NUMERIC DATA-NAME, OTHERWISE IT MUST BE REPORT ITEM OR NUMERIC DATA-NAME. FOUND *****. STATEMENT DISCARDED.

Programmer Response: Probable user error. Respecify item as numeric or replace name of store field before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4024I-E TWO OPERANDS ARE REQUIRED BEFORE 'GIVING'. STATEMENT DISCARDED.

Programmer Response: Probable user error. Correct syntax of arithmetic statement before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4026I-E *****/***** IS ILLEGALLY USED IN ***** TEST. TEST DISCARDED.

Programmer Response: Probable user error. Correct operand(s) of indicated test before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4027I-C RIGHT TERM OF A CONDITION MAY NOT BE NEGATED. NEGATION IS APPLIED TO THE RELATIONAL.

Programmer Response: Probable user error. Correct placement of NOT operator in relation condition before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4028I-C TWO 'NOT'S' IN SUCCESSION ILLEGAL. ACCEPTED AS CANCELLING EACH OTHER.

Programmer Response: Probable user error. Remove one of NOT operators in relation condition before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck control cards, and compiler output.

IKF4029I-E *****/***** MAY NOT BE COMPARED WITH *****/*****. TEST DISCARDED.

Programmer Response: Probable user error. Correct operands of relation condition before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4030I-E FOUND '*****' AFTER CONDITION, EXPECT 'OR', 'AND', OR VERB TO IMMEDIATELY FOLLOW CONDITION. DELETING TILL ONE OF THESE IS FOUND.

Programmer Response: Probable user error. Ensure that either OR, AND, or an imperative-statement follows condition before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4031I-E PROCEDURE-NAME NOT THAT OF A SINGLE GO PARAGRAPH MAY NOT BE ALTERED. STATEMENT DISCARDED.

Programmer Response: Probable user error. Correct operand of ALTER statement before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4032I-C NO ACTION INDICATED IF PRECEDING CONDITION IS TRUE. NEXT SENTENCE ASSUMED.

Programmer Response: Probable user error. Correct formation of IF statement before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4033I-E PROCEDURE-NAME WHICH IS THE END-OF-RANGE OF A PERFORM STATEMENT MAY NOT BE ALTERED. STATEMENT DISCARDED.

Programmer Response: Probable user error. Correct operand of ALTER statement before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4034I-C GO DEPENDING ON MUST BE FOLLOWED BY INTEGRAL IDENTIFIER LESS THAN OR EQUAL TO 4 DIGITS IN LENGTH. FOUND *****. STATEMENT DISCARDED.

Programmer Response: Probable user error. Supply valid operand of DEPENDING ON option before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4035I-W NO MORE THAN 3 DATA-NAMES SHOULD BE VARIED IN PERFORM STATEMENT. STATEMENT ACCEPTED AS WRITTEN.

Explanation: This compiler can normally handle a program varying more than three data-names, but the practice is invalid under standard COBOL language rules and is not recommended.

Programmer Response: If desired, limit number of operands of VARYING option and recompile if necessary.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4036I-W PERFORM RANGE IS FROM ***** TO *****, WHICH PRECEDES IT. STATEMENT ACCEPTED AS WRITTEN.

Explanation: This compiler can normally handle the perform range indicated, but the practice is not recommended.

Programmer Response: If desired, change operands of PERFORM statement and recompile if necessary.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4037I-E SYNTAX REQUIRES PROCEDURE-NAME TO FOLLOW 'THRU'. FOUND *****. ***** OPTION DISCARDED.

Programmer Response: Probable user error. Correct syntax of PERFORM statement before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4038I-E VARYING OPTION REQUIRES NUMERIC DATA-NAME. FOUND LITERAL. ARBITRARILY SUBSTITUTING *****

Programmer Response: Probable user error. Correct operand of VARYING option before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4039I-E *****/***** IN VARYING/TIMES OPTION IS NOT NUMERIC. ARBITRARILY SUBSTITUTING *****.

Programmer Response: Probable user error. Correct operand of VARYING or TIMES option before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4040I-E ***** FILE ***** MAY NOT BE OPENED ***** AND IS DISCARDED.

Programmer Response: Probable user error. Correct OPEN option for indicated file before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4041I-E SYNTAX REQUIRES 'INPUT', 'OUTPUT', OR 'I/O' AFTER OPEN. FOUND *****. DELETING TILL ONE OF THESE IS FOUND.

Programmer Response: Probable user error. Correct syntax of OPEN statement before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4042I-E SYNTAX REQUIRES FILE-NAME IN ***** STATEMENT. FOUND *****. DELETING TILL LEGAL ELEMENT FOUND.

Programmer Response: Probable user error. Supply valid file-name in indicated statement before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4043I-W REWRITE STATEMENT SHOULD NOT BE USED WITH A DIRECT BDAM-D FILE. STATEMENT ACCEPTED AS WRITTEN.

Programmer Response: Probable user error. Ensure that if REWRITE is used for a direct file, the system-name in the ASSIGN clause contains a 'W' in the organization field. 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.

IKF4044I-C '*****' SHOULD NOT BE MOVED TO NUMERIC FIELD. SUBSTITUTING *****.

Programmer Response: Probable user error. Either correct usage of items moved to numeric field or correct usage of field before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4045I-C 'ADVANCING' OPTION MUST BE FOLLOWED BY MNEMONIC NAME, NUMERIC INTEGRAL DATANAME OR INTEGER LESS THAN 100. FOUND *****. SUBSTITUTING *****.

Programmer Response: Probable user error. Include as identifier-2 in the WRITE statement a mnemonic name, a numeric integral data-name, or an integer less than 100. 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.

IKF4046I-E ILLEGAL TO *****/***** FILE *****. STATEMENT DISCARDED.

Programmer Response: Probable user error. Correct the invalid specification for the indicated file before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4047I-E READ OR WRITE VERB ILLEGAL FOR LABEL RECORDS. STATEMENT DISCARDED.

Programmer Response: Probable user error. Ensure that if data-names specified in the LABEL RECORDS clause are described in the Linkage Section, all READ and WRITE statements for the file are in the program that contains the Linkage Section.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4048I-E USE VERB MAY NOT APPEAR EXCEPT IN DECLARATIVES SECTION. STATEMENT DISCARDED.

Programmer Response: Probable user error. Remove USE statement from non-declarative portion of Procedure Division before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4049I-W INAPPROPRIATE OPTIONAL COBOL WORDS PRECEDING ***** IGNORED.

Programmer Response: Probable user error. Remove inappropriate words preceding indicated entry before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4050I-E SYNTAX REQUIRES *****. FOUND *****. STATEMENT DISCARDED.

Programmer Response: Probable user error. Correct syntax of statement before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4051I-E 'AFTER POSITIONING' REQUIRES 1-CHARACTER ALPHANUMERIC DATA NAME OR INTEGER LESS THAN 4. FOUND *****. SUBSTITUTING *****.

Programmer Response: Probable user error. Check the WRITE statement with the AFTER POSITIONING option. If necessary, rewrite the statement to include either identifier-2 as PICTURE X or the integer 0, 1, 2, or 3. 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.

IKF4052I-E *****/***** MAY NOT BE TARGET FIELD FOR *****/***** IN ***** STATEMENT, AND IS DISCARDED.

Programmer Response: Probable user error. Correct operands of indicated statement before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4053I-E POSITIONING VALID FOR CLOSE REEL WHILE DISP VALID FOR CLOSE. ***** FILE SKIPPED.

Programmer Response: Probable user error. Use either the CLOSE statement with the DISP option or the CLOSE statement with both the REEL option and the POSITIONING option. Rewrite the source statement as necessary; then recompile.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4054I-E SYNTAX REQUIRES SORT-FILE NAME. FOUND *****. STATEMENT DISCARDED.

Programmer Response: Probable user error. Supply valid sort-file-name before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4055I-C SORT SEQUENCE NOT SPECIFIED. ASCENDING ASSUMED.

Programmer Response: Probable user error. Specify ASCENDING and/or DESCENDING option and recompile if necessary.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4056I-E SYNTAX REQUIRES *****. FOUND *****. DISCARDED.

Programmer Response: Probable user error. Correct syntax of statement before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4057I-E NUMBER OF SORT KEYS EXCEEDS MAXIMUM OR TOTAL KEY LENGTH EXCEEDS 256 BYTES. ***** DISCARDED.

Programmer Response: Probable user error. Ensure that the number of sort keys is no more than 12 and the total length of all keys does not exceed 256 bytes before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4058I-E SYNTAX REQUIRES 'USING' ('GIVING') TO BE FOLLOWED BY STANDARD SEQUENTIAL FILE-NAME DEFINED UNDER AN FD. FOUND *****. STATEMENT DISCARDED.

Programmer Response: Probable user error. Correct syntax to include standard sequential file-name and recompile.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4059I-E SORT-KEY MUST BE NON-SUBSCRIPTED FIXED-LENGTH DATA-NAME DEFINED UNDER AN SD. FOUND *****. DISCARDED.

Programmer Response: Probable user error. Correct type and/or position of sort key before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4060I-C ***** IS NOT A POSITIVE NUMERIC INTEGRAL LITERAL OF REQUIRED LENGTH. ***** OPTION DISCARDED.

Programmer Response: Probable user error. Correct operand of indicated option before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4061I-W NEITHER NAMED NOR CHANGED SPECIFIED. STATEMENT ACCEPTED. WILL BE TREATED AS FORMATTED DISPLAY.

Programmer Response: Probable user error. Specify an EXHIBIT statement option before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4062I-W 'NAMED CHANGED' ACCEPTED AS 'CHANGED NAMED'.

Programmer Response: Probable user error. Correct EXHIBIT statement option and recompile if necessary.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4063I-W PREVIOUS DEBUG PACKET REFERS TO SAME PROCEDURE-NAME. CARD DELETED AND FOLLOWING STATEMENTS ATTACHED TO IMMEDIATELY PRECEDING PACKET.

Programmer Response: Probable user error. Ensure that only one DEBUG packet refers to a given location in program before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4064I-E ***** IS NOT A POSITIVE NUMERIC INTEGRAL LITERAL OF REQUIRED LENGTH. SUBSTITUTING *****.

Programmer Response: Probable user error. Correct indicated entry before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4065I-W NUMERIC LITERAL IN EXAMINE STATEMENT SHOULD BE UNSIGNED. SIGN IGNORED.

Programmer Response: Probable user error. Remove sign from numeric literal and recompile if necessary.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4066I-E SYNTAX REQUIRES 01 LEVEL SD DATA-NAME IN RELEASE STATEMENT. FOUND *****. STATEMENT DISCARDED.

Programmer Response: Probable user error. Correct RELEASE statement operand before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4067I-W ALL CHARACTER SHOULD NOT BE USED AS LITERAL IN EXAMINE STATEMENT. STATEMENT ACCEPTED AS WRITTEN.

Programmer Response: Probable user error. Supply valid literal in EXAMINE statement before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4068I-D COMPILER ERROR. COMPILATION WILL NOT BE COMPLETE.

Explanation: Undefined data attribute.

Programmer Response: The compiler has reached a point in its processing where it is unable to continue. If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4069I-C SYNTAX REQUIRES DEVICE-NAME. FOUND ***** IN ***** STATEMENT. SYSTEM UNIT ASSUMED.

Programmer Response: Probable user error. Specify a valid device-name or mnemonic-name associated with a device-name in the SPECIAL-NAMES paragraph before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4070I-C 'AT-END' CLAUSE REQUIRED HERE. 'AT-END-NEXT-SENTENCE' ASSUMED.

Programmer Response: Probable user error. Insert in the source statement at AT END option followed by an imperative statement. 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.

IKF4071I-E ***** EXCEEDS LEGAL LENGTH. DISCARDED.

Programmer Response: Probable user error. Correct length of indicated item before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4072I-W EXIT FROM ***** PROCEDURE ASSUMED BEFORE *****.

Programmer Response: Probable user error. Ensure that END DECLARATIVES, a section-name within the declaratives section, or the end of the range of the PERFORM exists following routine before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4073I-W ***** SHOULD NOT APPEAR IN DECLARATIVE SECTION. STATEMENT ACCEPTED AS WRITTEN.

Explanation: The statement will be compiled, but its use is illegal under standard COBOL rules and is not recommended.

Programmer Response: Probable user error. Remove indicated statement from declarative section before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4074I-C STATEMENT CONTAINS FLOATING POINT DATA ITEMS. REMAINDER IGNORED.

Programmer Response: Probable user error. If the REMAINDER option of the DIVIDE statement is required, ensure that none of the operands are floating-point items before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4075I-C 'NEXT SENTENCE' ILLEGAL AND DISCARDED. BOTH ***** AND NOT ***** WILL CAUSE EXECUTION OF NEXT VERB.

Programmer Response: Probable user error. Remove illegal NEXT SENTENCE specification from AT END, ON SIZE ERROR, or END-OF-PAGE options before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4076I-E ***** REQUIRES ***** LEVELS OF SUBSCRIPTING OR INDEXING. SUBSTITUTING FIRST OCCURRENCE OF *****.

Programmer Response: Probable user error. Provide required level of subscripting or indexing for indicated item before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4077I-E ***** MAY NOT BE USED AS A SUBSCRIPT SINCE IT REQUIRES SUBSCRIPTING ITSELF. SUBSTITUTING FIRST OCCURRENCE OF *****.

Programmer Response: Probable user error. Provide a subscript that itself requires no subscripting before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4078I-E SUBSCRIPT MUST BE INTEGRAL DATA-NAME OR LITERAL. FOUND NON-INTEGER *****. SUBSTITUTING FIRST OCCURRENCE OF *****.

Programmer Response: Probable user error. Ensure that subscript is either a data-name representing an integral value or a literal before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4079I-E ***** FOUND AMONG SUBSCRIPTS. SUBSTITUTING FIRST OCCURRENCE OF *****.

Programmer Response: Probable user error. Substitute valid subscript for indicated item before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4080I-W DEBUG CARD MAY NOT REFER TO A PROCEDURE NAME WHICH ITSELF IS IN A DEBUG PACKET. CARD DELETED AND FOLLOWING STATEMENTS ATTACHED TO IMMEDIATELY PRECEDING PACKET.

Programmer Response: Probable user error. Ensure that the location specified on one DEBUG card is neither used on any other DEBUG card, nor is a location within any other DEBUG packet before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4081I-C ***** EXCEEDS ***** CHARACTERS. UP TO 114 ACCEPTED.

Programmer Response: Probable user error. Correct length of operand before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4082I-E ***** IS NOT DEFINED AS SUBSCRIPTED OR INDEXED. SUBSCRIPTS DISCARDED.

Programmer Response: Probable user error. Specify required options of OCCURS clause for item before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4083I-E OCCURS-DEPENDING-ON VARIABLE MUST BE INTEGRAL NON-SUBSCRIPTED DATA-NAME. FOUND *****. ARBITRARILY SUBSTITUTING *****.

Programmer Response: Probable user error. Correct operand of DEPENDING ON option before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4084I-C ILLOGICAL USE OF PARENTHESES ACCEPTED WITH DOUBTS AS TO MEANING.

Programmer Response: Probable user error. Check logic behind use of parentheses before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4085I-E RECORD DESCRIPTION FOR FILE ***** MISSING OR ILLEGAL. STATEMENT DISCARDED.

Programmer Response: Probable user error. Ensure that any errors detected by compiler during Data Division scan on indicated file's record description are corrected before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4086I-C ***** CONDITION USED WHERE ONLY IMPERATIVE STATEMENTS ARE LEGAL MAY CAUSE ERRORS IN PROCESSING.

Programmer Response: Probable user error. Ensure that conditional statements do not appear where imperative-statements are required before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4087I-E 'END DECLARATIVES' MISSING OR MISPLACED. PROGRAM CANNOT BE EXECUTED.

Programmer Response: Probable user error. Ensure that the end of the Declaratives Section has been indicated by an END DECLARATIVES statement before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4088I-D COMPILER ERROR. COMPILATION WILL NOT BE COMPLETE.

Explanation: I-C text count field is 0. Skipping to phase 5.

Programmer Response: The compiler has reached a point in its processing where it is unable to continue. If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4089I-W *****/***** SHOULD NOT BE TARGET FIELD FOR *****/***** IN ***** STATEMENT. STATEMENT ACCEPTED AS WRITTEN.

Programmer Response: Probable user error. Correct operand being used as target field in indicated statement before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4090I-E SORT-KEY MUST BE IN FIXED POSITION NOT MORE THAN 4092 BYTES FROM START OF RECORD. ***** DISCARDED.

Programmer Response: Probable user error. Correct position of sort key within record before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4091I-E SYNTAX REQUIRES OPERAND. FOUND *****. TEST DISCARDED.

Programmer Response: Probable user error. Supply a valid operand for statement before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4092I-W EXTERNAL DECIMAL NAME USED IN TRANSFORM STATEMENT. STATEMENT ACCEPTED AS WRITTEN.

Programmer Response: Probable user error. Ensure that TRANSFORM statement operands are either alphabetic, alphanumeric, or numeric edited items (identifier-3 only), before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4093I-C ALL WRITE STATEMENTS FOR ***** SHOULD HAVE ***** OPTION. ***** ***** 1 LINE ASSUMED.

Programmer Response: Probable user error. Ensure that if a WRITE statement with the ADVANCING or POSITIONING option is written for a record in a file, every WRITE statement for the file specifies the same option. 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.

IKF4094I-W ***** IS IN A RECORD OF AN APPLY-WRITE-ONLY FILE, AND REFERRING TO IT MAY CAUSE ERRORS IF FILE IS OPENED AS OUTPUT WHEN ***** STATEMENT IS EXECUTED.

Programmer Response: Probable user error. Referring to subfields of records of a file for which APPLY WRITE-ONLY has been specified is not recommended. Make any necessary changes before recompiling.

If the program does not execute properly, and if after taking out references the problem recurs, do the following before calling IBM for programming support. Have source deck, control cards, and compiler output available.

IKF4095I-E WRITE FROM IDENTIFIER REQUIRED FOR *****, TO WHICH WRITE-ONLY IS APPLIED. STATEMENT DISCARDED.

Programmer Response: Probable user error. Specify WRITE with the FROM option for files for which APPLY WRITE-ONLY has been specified.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4096I-W ***** STATEMENT WILL NEVER BE EXECUTED.

Explanation: The logic of the COBOL source program prevents the computer from executing the statement noted. The compiler, however, accepts the statement as written.

Programmer Response: Probable user error. Check placement of statement and recompile if necessary.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4097I-C UNIT(REEL) OPTION ILLEGAL FOR *****. DISCARDED.

Programmer Response: Probable user error. Remove illegal option from indicated statement before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4098I-E 'ALTER' STATEMENT REFERS TO A GO TO IN A DIFFERENT INDEPENDENT SEGMENT. IGNORED.

Programmer Response: Probable user error. Ensure that the operand of the ALTER statement refers to a paragraph-name within an independent segment of the same priority as the segment containing the ALTER statement before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4099I-E NO EXIT SPECIFIED BEFORE END OF THIS DECLARATIVE SECTION. CONTROL WILL FALL THROUGH TO NEXT SECTION.

Programmer Response: Probable user error. Correct logic of statements within Declaratives Section before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4100I-W IDENTIFIER FOLLOWING INTO (FROM) IN READ (WRITE) STATEMENT SHOULD NOT BE DEFINED UNDER SAME FD AS RECORD NAME. STATEMENT ACCEPTED AS WRITTEN.

Programmer Response: Probable user error. Ensure that the operand of the FROM or INTO option is an identifier that is the name of a Working-Storage or Linkage Section item, or a record of another file before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4101I-E SET STATEMENT REQUIRES OPERAND AFTER 'UP' OR 'DOWN' TO BE NUMERIC INTEGRAL DATA-NAME OR POSITIVE INTEGRAL NUMERIC LITERAL. FOUND *****. STATEMENT DISCARDED.

Programmer Response: Probable user error. Correct operand of UP or DOWN option before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4102I-E SET STATEMENT REQUIRES OPERAND AFTER 'TO' TO BE INDEX NAME, INDEX DATA ITEM, NUMERIC INTEGRAL DATA-NAME OR INTEGRAL NUMERIC LITERAL GREATER THAN ZERO. FOUND *****. STATEMENT DISCARDED.

Programmer Response: Probable user error. Correct operand of TO option before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4103I-C ALL MUST BE FOLLOWED BY ALPHANUMERIC LITERAL. FOUND *****. 'ALL' DISCARDED.

Programmer Response: Probable user error. Correct formation of figurative constant before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4104I-E SEARCH OR SEARCH ALL STATEMENT HAS EITHER SUBSCRIPTED OR INDEXED IDENTIFIER-1 OR ILLEGAL OPERAND. SCANNING TIL 'AT END' OR 'WHEN' DELETING TIL ONE OF THESE IS FOUND.

Programmer Response: Probable user error. Correct operand of SEARCH or SEARCH ALL statement before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4105I-E DATA-NAME CANNOT BE BOTH INDEXED AND SUBSCRIPTED IN ***** STATEMENT. SUBSCRIPTS DISCARDED.

Programmer Response: Probable user error. Correct qualification of data-name in indicated statement before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4106I-E DATA-NAME MUST BE INDEXED BY INDEX NAME OR INDEX NAME PLUS OR MINUS AN INTEGRAL NUMERIC LITERAL. SUBSTITUTING FIRST OCCURRENCE OF *****.

Programmer Response: Probable user error. Correct manner in which data-name is indexed before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4107I-E ***** ILLEGAL FOR SORT FILE. STATEMENT DISCARDED.

Programmer Response: Probable user error. Correct invalid specification indicated before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4108I-E CALLED PROGRAM MAY NOT BE SEGMENTED. ENTRY STATEMENT IGNORED.

Programmer Response: Probable user error. Ensure that all restrictions on subprogram linkage are followed before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4109I-E KEY IN SEARCH-ALL FLOATING POINT OR STERLING STATEMENT CHANGED TO SEARCH STATEMENT.

Programmer Response: Probable user error. Ensure that keys are either DISPLAY, COMPUTATIONAL, or COMPUTATIONAL-3 items before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4110I-E CONDITION IN SEARCH ALL STATEMENT TESTS KEY WITHOUT TESTING ALL PRECEDING KEYS. STATEMENT DISCARDED.

Programmer Response: Probable user error. Ensure that the condition specified in the SEARCH ALL statement tests all keys before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4111I-E INVALID CONDITION OR INVALID FORMULA IN CONDITION IN SEARCH-ALL STATEMENT. STATEMENT DISCARDED.

Programmer Response: Probable user error. Correct condition in SEARCH ALL statement before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4112I-W SET UP OR DOWN SHOULD NOT INCREMENT INDEX-NAME BY INDEX DATA ITEM. ACCEPTED AS WRITTEN.

Programmer Response: Probable user error. Correct SET statement before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4113I-E LABEL DECLARATIVE ILLEGAL FOR ISAM FILE *****. FILE SKIPPED.

Programmer Response: Probable user error. Ensure that the USE statement for labels associated with an indexed file specifies only options compatible with standard label processing. 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.

IKF4114I-E LABEL DECLARATIVE FOR BDAM FILE SHOULD NOT HAVE UNIT OR REEL OPTION. ***** FILE SKIPPED.

Programmer Response: Probable user error. Correct USE statement associated with BDAM file before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF4115I-E ***** STATEMENT REQUIRES IDENTIFIER WHOSE USAGE IS DISPLAY. FOUND SPECIAL REGISTER. STATEMENT DISCARDED.

Programmer Response: Probable user error. For the statement indicated, remove special register as an operand and provide an identifier whose USAGE IS DISPLAY; 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.

IKF4141I-W IN A QSAM OR QISAM REWRITE, THE INVALID KEY CLAUSE HAS NO MEANING AND IS IGNORED.

Programmer Response: Delete the INVALID KEY clause from the 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.

IKF5001I-D COMPILER ERROR. COMPILATION WILL NOT BE COMPLETE.

Explanation: An unrecoverable compiler logic error or machine error has occurred while trying to assign a double register.

Programmer Response: The compiler has reached a point in its processing where it is unable to continue. If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output with PMAP and DMAP.

IKF5002I-D COMPILER ERROR. COMPILATION WILL NOT BE COMPLETE.

Explanation: An unrecoverable compiler logic error or machine error occurred while processing a subscripted or indexed data-name.

Programmer Response: The compiler has reached a point in its processing where it is unable to continue. If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output with PMAP and DMAP.

IKF5003I-C A DIVISOR IS A ZERO CONSTANT. RESULT OF DIVIDE WILL BE SET TO ALL 9'S.

Programmer Response: Probable user error. Correct divisor to prevent division by zero before recompiling.

If the problem recurs, do the following before calling IBM for programming support. Have source deck, control cards, and compiler output with PMAP and DMAP available.

IKF5004I-W ALPHANUMERIC SENDING FIELD EXCEEDS MAXIMUM PERMISSIBLE SIZE. 18 LOW ORDER BYTES USED.

Programmer Response: Probable user error. Correct length of alphanumeric sending field or receiving field before recompiling.

If the problem recurs, do the following before calling IBM for programming support. Have source deck, control cards, and compiler output with PMAP and DMAP available.

IKF5005I-D COMPILER ERROR. COMPILATION WILL NOT BE COMPLETE.

Explanation: An unrecoverable compiler logic error or machine error occurred while processing a move.

Programmer Response: The compiler has reached a point in its processing where it is unable to continue. If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output with PMAP and DAMP.

IKF5006I-D COMPILER ERROR. COMPILATION WILL NOT BE COMPLETE.

Explanation: Unexpected input to the move or store processor.

Programmer Response: The compiler has reached a point in its processing where it is unable to continue. If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output with PMAP and DMAP.

IKF5007I-D COMPILER ERROR. COMPILATION WILL NOT BE COMPLETE.

Explanation: Unexpected input to the arithmetic code generator.

Programmer Response: The compiler has reached a point in its processing where it is unable to continue processing. If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output with PMAP and DMAP.

KF5008I-D COMPILER ERROR. COMPILATION WILL NOT BE COMPLETE.

Explanation: Unexpected input to the floating-point arithmetic routine 'FPCVBH'.

Programmer Response: The compiler has reached a point in its processing where it is unable to continue. If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output with PMAP and DMAP.

IKF5009I-D COMPILER ERROR. COMPILATION WILL NOT BE COMPLETE.

Explanation: Lost subscript or index ID in table 'XSSNT'.

Programmer Response: The compiler has reached a point in its processing where it is unable to continue. If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output with PMAP and DMAP.

IKF5010I-C A CONSTANT INTERMEDIATE RESULT HAD TO HAVE ITS HIGH ORDER DIGIT POSITION TRUNCATED.

Programmer Response: Probable user error. If high order truncation is not desired, make necessary corrections before recompiling.

If the problem recurs, do the following before calling IBM for programming support. Have source deck, control cards, and compiler output with PMAP and DMAP available.

IKF5011I-W AN INTERMEDIATE RESULT OR A SENDING FIELD MIGHT HAVE ITS HIGH ORDER DIGIT POSITION TRUNCATED.

Programmer Response: Probable user error. If high order truncation is not desired, make necessary corrections before recompiling. If after making necessary correction, high order truncation occurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output with PMAP and DMAP.

IKF5012I-D COMPILER ERROR. COMPILATION WILL NOT BE COMPLETE.

Explanation: Lost intermediate result attributes in 'XINTR' table.

Programmer Response: The compiler has reached a point in its processing where it is unable to continue. If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output with PMAP and DMAP.

IKF5013I-C ILLEGAL COMPARISON OF TWO NUMERIC LITERALS. STATEMENT DISCARDED.
Programmer Response: Probable user error. Correct operands of the comparision before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output with PMAP and DMAP.

IKF5014I-E KEY IN SEARCH ALL AT INVALID OFFSET. STATEMENT DISCARDED.

Programmer Response: Probable user error. Correct offset of key in SEARCH ALL statement before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output with PMAP and DMAP.

IKF5015I-E INVALID USE OF SPECIAL REGISTER. STATEMENT DISCARDED.

Programmer Response: Probable user error. Check data-names, procedure-names, etc., to ensure that a special register has not been used in an illegal capacity before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output with PMAP and DMAP.

IKF5016I-D MORE THAN 255 SUBSCRIPT ADDRESS CELLS USED. PROGRAM CANNOT EXECUTE PROPERLY.

Programmer Response: The compiler has reached a point in its processing where it has encountered an unrecoverable error. If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output with PMAP and DMAP.

IKF5017I-W EXHIBIT CHANGED OPERAND GREATER THAN 256 BYTES. LENGTH OF 256 ASSUMED.

Programmer Response: Probable user error. Specify an operand with length less than the 256 bytes for the EXHIBIT CHANGED statement; then recompile.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output with PMAP and DMAP.

IKF5018I-W NAME OF IDENTIFIER IN EXHIBIT STATEMENT EXCEEDS MAXIMUM. TRUNCATED TO 120.

Programmer Response: Probable user error. Correct operand length in EXHIBIT statement before recompiling.

If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output with PMAP and DMAP.

Note: The preceding messages are grouped in the compiler output listing. The following messages may be interspersed in the compiler output listing.

IKF6001I-C ERROR FOUND PROCESSING FILE 4 TEXT. 00 CODE IN LISTING TEXT.

Programmer Response: Probable user error. Correct the other errors indicated in your source statements; 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.

IKF6002I-C ERROR FOUND PROCESSING F4 TEXT. END OF LISTING TEXT REACHED.

Programmer Response: Probable user error. Correct the other errors indicated in your source statements; 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.

IKF6003I-D COMPILER ERROR. COMPILATION WILL NOT BE COMPLETE.

Explanation: Error found processing F4 text. Unknown data A-TEXT code.

Programmer Response: The compiler has reached a point in its processing where it is unable to continue. If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF6004I-C ERROR FOUND PROCESSING FILE 4 TEXT. EOF REACHED WHILE PROCESSING LISTING TEXT.

Programmer Response: Probable user error. Correct the other errors indicated in your source statements; 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.

IKF6005I-D COMPILER ERROR. COMPILATION WILL NOT BE COMPLETE.

Explanation: Error found processing F1 text.

Programmer Response: The compiler has reached a point in its processing where it is unable to continue. If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

IKF6006I-E SUPMAP SPECIFIED AND E-LEVEL DIAGNOSTIC HAS OCCURRED. PMAP LOAD DECK IGNORED.

Programmer Response: Probable user error. If even with a severity level of E you want to take advantage of the options indicated, as well as have the object code printed, either replace SUPMAP with NOSUPMAP or let the compiler default to it. 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.

IKF6007I-D TABLE HAS EXCEEDED MAXIMUM SIZE. PMAP, LOAD MODULE AND DECK WILL BE INCOMPLETE. INCREASE SIZE PARAMETER.

Programmer Response: Probable user error. Increase the size parameter, and then recompile. If the problem recurs, have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

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: Probable user error. 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: Probable user error. 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.

IKF003I- Jobname, stepname, unit address, device type, ddname, operation attempted, error description. Each of the following fields if printed will vary in length:

```
UNIT RECORD--Access method
MAGNETIC TAPE--Relative block number (decimal), access
               method
DIRECT ACCESS--Actual track address and block number
```

Programmer Response: Provide the additional control information required; 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.

IKF111I- EXCEPTIONAL I/O CONDITION SENSED PROCESSING ddname.

Explanation: This message is issued 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.

IKF888I- UNSUCCESSFUL SORT FOR SORT-FILE-NAME.

Explanation: The System/360 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.

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 cancelled. 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.

Operator Response: Probable user error. Issue a REPLY command. The contents of the text field should be supplied by the programmer.

IKF999I- UNSUCCESSFUL OPEN FOR ddname.

Explanation: The operating system could not find a corresponding ddname on a DD statement for a file assigned in a SELECT clause or a file previously closed with LOCK. DDNAME replaced by DDNAME of the file.

Operator Response: Probable user error. User should have indicated either to continue processing or to cancel the job. Note that if the user elects to continue processing, any READ or WRITE encountered for the file will result in a ABEND.

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 (problem program/processor 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 (problem program/processor 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.

This appendix contains a summary of the major COBOL limits. Each limit is categorized as applicable to a COBOL source program in general or to one of several special features of this COBOL compiler. In a number of cases, as indicated in the summary, several factors together determine the effective COBOL limits.

Table 36. General Limits for COBOL Source Programs (Part 1 of 2)

	User Specification	Limit
Input-Output Section	Number of files per program	255 minus the number of sort files referred to in a SAME RECORD AREA clause. (See also Working-Storage Section.)
	Number of records per file	No effective limit.
	Number of characters per record	Record length limits, which are set by data management, are affected by such file characteristics as device type, file type, and so on.
Data Division	Size of numeric literal	18 digits.
	Size of non-numeric literal	120 characters.
	Size of a PICTURE	30 characters.
	Number of data-names per program	No effective limit.
Working-Storage Section	Size of Working-Storage	The product of 4096 and (the number 255 minus the number of file descriptions).
	Number of 01-level entries	No effective limit.
	Size of a PICTURE	30 characters.
Procedure Division	Number of paragraphs per program	65,535 or less, depending on the length of paragraph-names and section-names.
	Number of statements per program	Set by the number of internal counters. For example, there can be a maximum of 65,535 generated names.
	Size of statements	No effective limit.
	Size of operand in ACCEPT or DISPLAY statement	131,071 characters.
	Size of non-numeric operand in a comparison	131,071 characters.
	Size of numeric operand in a comparison	18 digits.

Table 36. General Limits for COBOL Source Programs (Part 2 of 2)

	User Specification	Limit
Procedure Division (continued)	Number of nested conditions per sentence	No effective limit.
	Number of receiving fields in a MOVE sequence	No effective limit.
	Literals	16,255; the total length of all literals may not exceed 32,511 bytes.

Table 37. Limits for Special Features of COBOL

	User Specification	Limit
Source Program Library	Number of COPY statements per program	No effective limit.
Segmentation	Number of sections per program	65,535 including both section-names and paragraph-names.
	Number of paragraphs per section	Subject to the restrictions on the number of sections, paragraphs, and statements in a COBOL program.
	Number of segments per program	100 unique segment numbers; the number of segments with the same segment number is unlimited.
	Number of statements per segment	No effective limit.
Table Handling	Number of tables per program	65,535 one-dimensional tables, each with one index-name. There is no effective limit on the number of subscripted tables.
	Size of a table	131,071 characters

(Where more than one page reference is given, the major reference appears first.)

Special Character Subjects

READER'S COMMENTS

TITLE: IBM OS Full American National Standard COBOL Compiler and Library, Version 2 Programmer's Guide

ORDER NO. GC28-6399-2

Your comments assist us in improving the usefulness of our publications; they are an important part of the input used in preparing updates to the publications. All comments and suggestions become the property of IBM.

Please do not use this form for technical questions about the system or for requests for additional publications; this only delays the response. Instead, direct your inquiries or requests to your IBM representative or to the IBM Branch Office serving your locality.

Corrections or clarifications needed:

Page *Comment*

Please include your name and address in the space below if you wish a reply.

Thank you for your cooperation. No postage necessary if mailed in the U.S.A.

cut along this line

fold fold

FIRST CLASS
PERMIT NO. 33504
NEW YORK, N.Y.

BUSINESS REPLY MAIL

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY . . .

IBM CORPORATION
1271 Avenue of the Americas
New York, New York 10020

Attention: PUBLICATIONS

fold fold

IBM

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)

IBM OS Full ANS COBOL V2 P.G. Printed in U.S.A. GC28-6399-2

GC28-6399-2

IBM OS Full ANS COBOL V2 P.G. Printed in U.S.A. GC28-6399-2

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)