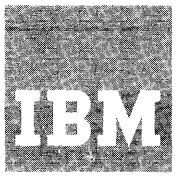
File No. S360-36
GC28-6703-0

OS



Systems Reference Library

IBM System/360 Operating System:

Job Control Language User's Guide

The job control language (JCL) is used with all System/360 Operating System control programs. Every job submitted for execution by the operating system must include JCL statements. These statements contain information required by the operating system to initiate and control the processing of jobs.

This publication contains tutorial information on JCL for programmers. Special emphasis is placed on "how to" perform specific functions using a subset of the JCL statements rather than on describing the full facilities of each statement. This publication has four parts:

- Part I: Introduction to the job control language -- describes how to use each JCL statement.
- Part II: JCL for compilers, linkage editor, and loader -- contains a summary of the JCL statements used by those programs and examples of their use.
- Part III: Cataloged and in-stream procedures -- describes how to use and write cataloged and in-stream procedures.
- Part IV: Examples of cataloged procedures for compilations, link edits, and executions -- contains IBM supplied cataloged procedures for those functions and examples of their use.

After becoming familiar with the information presented in this manual, you may use IBM System/360 Operating System: Job Control Language Reference, GC28-6704 for review and reference.

First Edition (June, 1970)

This edition applies to release 19, of IBM System/360 Operating System, and to all subsequent releases until otherwise indicated in new editions or Technical Newsletters. Changes are continually made to the information herein; before using this publication in connection with the operation of IBM systems, consult the latest IBM System/360 SRL Newsletter, Order No. GN20-0360, for the 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 Systems Publications, Department D58, PO Box 390, Poughkeepsie, N. Y. 12602

Preface

This publication describes the facilities of the job control language (JCL) to new programmers or to programmers who are not familiar with the IBM System/360 Operating System. It is assumed that the reader is familiar with the concepts and terminology introduced in the prerequisite publications listed below and that he can write programs using the assembler or a higher-level language such as ALGOL, COBOL, American National Standard COBOL (formerly USAS COBOL), FORTRAN or PL/I. These languages are used throughout this publication to illustrate the use of JCL in various applications. If you use another language, you can still use this publication to learn JCL and refer to the publications associated with your language for specific examples.

There are only two language-dependent parameters in JCL statements: the PARM parameter of the EXEC statement and the DCB parameter of the DD statement. The appropriate values for those two parameters are summarized in this publication for the users of the assembler, ALGOL, COBOL E and F, American National Standard COBOL, FORTRAN E, G, and H, and PL/I F; however, you should check the related Programmer's Guides for up-to-date information. If you use another language, you must find the values for those parameters in the publications associated with your language.

PREREQUISITE PUBLICATIONS

IBM System/360 Operating System:

Introduction, GC28-6534

Concepts and Facilities, GC28-6535

ASSOCIATED PUBLICATIONS

IBM System/360 Operating System:

ALGOL Programmer's Guide, GC33-4000

American National Standard COBOL Programmer's Guide, GC28-6399

Assembler (E) Programmer's Guide, GC28-6595

Assembler (F) Programmer's Guide, GC26-3756

COBOL (E) Programmer's Guide, CG24-5029

COBOL (F) Programmer's Guide, CG28-6380

FORTRAN IV (E) Programmer's Guide, GC28-6603

FORTRAN IV (G and H) Programmer's Guide, GC28-6817

Linkage Editor and Loader, GC28-6538

PL/I (F) Programmer's Guide, GC28-6594

Contents

Illustrations

Figures

Tables

Summary of Major Changes--Release 19

The Release 19 changes listed below are described in this manual. They are indicated in the text by a vertical line to the left of the change.

Item	Description	Areas Affected
System Management Facilities Subset 1	The TIME parameter on the JOB and EXEC statements now applies to MFT as well as MVT.	19,30,38,48,58, 64
System Management Facilities Subset 2	The addition of the OUTLIM parameter on the DD statement that specifies SYSOUT to limit an output data set.	77,92-93,392
Input/Output Recovery Management Support	A new command, SWAP, that allows Dynamic Device Reconfiguration of two volumes has been added.	247,248
Data Management Support for American National Standard COBOL	Two new values for BFTEK, DCB subparameter, have been added. A specifies record area buffering: R specifies record buffering.	357
Recognition of EOF on Input	B, a new value for DCB subparameter OPTCD, requests that end-of-file recognition be disregarded for tapes.	361
ISAM Improvements	For ISAM, a newly created data set can now overlay an older one -- reusing the space. The independent overflow area of an ISAM data set can now be on a different device type from the prime area.	342,346-349, 351-352
Direct System Output Facility	In MFT and MVT, an output data set can now be written directly to the desired unit record or magnetic tape device.	89
Seven-Track Tape Default of 800 BPI	The default for 7-track tape is now 800 bits-per-inch.	359
DD DUMMY Substitution at Restart	A data set that is not needed after restart can be defined by coding the DUMMY parameter.	367
In-Stream Procedures	A facility has been added that allows procedures to be included in the input stream of a job.	13,15,19-23,54 243,245,293-314, 390,392
Main Storage Hierarchy Support MVT Extension	If you code the REGION parameter and request storage only from hierarchy 1, no hierarchy 0 segment is allocated.	43,68
Blocksize Adjustment for Sysout Data Sets	If the BLKSIZE parameter for a SYSOUT data set is not an integral multiple of and larger than the logical record length, it is adjusted.	358

Introduction

This publication contains tutorial information on the job control language. This information is presented in four parts:

Part I: Introduction to the job control language
Part II: JCL for compilers, linkage editor, and loader
Part III: Cataloged and in-stream procedures
Part IV: Examples of cataloged procedures for compilations, link edits, and executions.

The text relies heavily on examples to illustrate the facilities of JCL. To facilitate reading Part I of this manual, all examples appear on shaded areas. The use of shaded areas allows you to skip the examples when you are reviewing the instructions, or to quickly find examples of how to code a given function.

Parts II, III, and IV also contain many examples, but they are isolated in sections within each part. Shading is not used in this case.

There are four appendixes to the publication:

Appendix A: Indexed Sequential data sets
Appendix B: Glossary of DCB Subparameters
Appendix C: Using the Restart Facilities.
Appendix D: Summary of the DD Statement.

A glossary of terms used in this publication follows the appendixes.

Part I: Introduction to the Job Control Language

The purpose of an operating system is to supervise and optimize the work done in a computer. The IBM System/360 Operating System achieves this purpose by managing the allocation of system resources to the different units of work (jobs) to be performed by the installation. The system resources are the central processing unit, main storage, input/output devices, and any programs that are part of the system.

To be effective, the installation's operating system must be general enough to accommodate the variety of jobs to be performed in that installation. Therefore, you, the user, must communicate with the operating system to describe the requirements of your particular job. You do this through the Job Control Language (JCL).

The JCL statements allow you to tell the system how you have divided your job into job steps (units of work, each of which is associated with one processing program and related data), where your data is, and what resources are needed to execute each step. Other facilities of the language allow you to:

- Optimize use of channels, units, volumes, and direct access space.
- Pass data sets used by more than one step from one step to the next step that uses it, to reduce mounting and retrieval time.
- Copy information on other JCL statements with a backward-reference facility to reduce recoding time.
- Retrieve a data set by name using the system catalog, eliminating the need to know its exact location.

In multiprogramming environments (MFT and MVT) the job control language also allows you to:

- Share data sets between two or more job steps that are operating independently.
- Classify jobs according to their characteristics so that the system may balance the mix of jobs for more efficient operation.

JCL also provides a series of commands that are used by the operator to communicate with the system.

The flexibility of the job control language is characterized by its large number of optional facilities. Most applications require only a limited number of these facilities. Applications that require a heavy use of JCL or those that are performed on a regular basis can be considerably simplified through the use of cataloged or in-stream procedures. A procedure is a set of precoded JCL statements that can be retrieved by coding a simple name on one JCL statement. Any statement in the set can be temporarily modified by other JCL statements submitted at the time the procedure is used. A procedure can contain precoded JCL statements for one or more job steps. A job step in a cataloged procedure is called a procedure step. Cataloged procedures reside on a procedure library. In-stream procedures are coded as part of your job.

What is a Job?

A job is a series of operations that solve a particular problem. These operations are carried out by one or more processing programs. The processing programs you may use in a job are the language processors and service programs provided by IBM and your own programs (see Table 1). You may also use cataloged procedures which in turn may contain one or more procedure steps. (Each procedure step uses a processing program.)

In other words, your job may use one or more processing programs and cataloged procedures to accomplish its purpose. Each unit of work associated with one of those programs or procedure steps is called a job step. Within each job step you must also provide whatever data the processing program may need.

Table 1. Processing Programs

Processing Programs		
Language Processors	Service Programs	Application Programs
ALGOL	Data set utilities	User-written
Assembler	Independent utilities	
COBOL	Linkage Editor	
FORTRAN	Loader	
PL/I	Sort/Merge	
RPG	System utilities	
	TESTRAN	

For example, you may write a program in COBOL to process insurance premium payments. Your program must be compiled and link edited before it can be executed. Therefore, your job will have three job steps: compilation, link editing, and execution. During the compilation step the COBOL compiler processes your source program (input data) and produces an object module (output data). In the next step, the linkage editor uses the object module as its input data and produces a load module (output data). In the last step, the load module (your program in executable form) processes the insurance premium transactions (input data) and records them in a master file (output data).

In addition to writing the source program, you must also write the JCL statements that control the selection of the COBOL compiler and linkage editor and define the data used in your job. First, you must write a JOB statement which marks the beginning of your job. You may also use this statement to record some other information such as your name and account number. After the JOB statement, you write an execute (EXEC) statement to mark the beginning of your first job step. In this EXEC statement you request the COBOL compiler. Following the EXEC statement, you write a data definition (DD) statement for each data set the COBOL compiler requires. One of these DD statements tells the compiler where to find your source program. (The source program can also follow the DD statements required in this step. The DD statement that defines your source program can indicate this fact.) Another DD statement tells the compiler where to place the object module. Other DD statements define data sets that can be used as work areas by the compiler and for printing messages and compilation listings. After you write all the statements required by the COBOL compiler, you must write another EXEC statement to mark the beginning of the second job step and

to request the linkage editor. In turn, the linkage editor requires certain data sets, which you also define through DD statements. One of the DD statements tells the linkage editor where the COBOL compiler placed the object module. Another DD statement tells the linkage editor where it is to place the load module. The remaining DD statements define data sets for work areas and for printing messages and listings. Your next EXEC statement begins the third step and requests that your program (load module produced in the previous step) be executed. You will need DD statements to tell your program where the insurance premium transactions are (they can also follow the DD statements for this step if indicated by a DD statement), and where the master file is. You will also define any work areas and other data sets your program requires.

If you plan to use your program several times, you can have the linkage editor place it in a library (a permanent data set). Then you do not need to compile and link edit it every time you use it, and can then reduce your job to one job step.

Alternatively, you could use a cataloged procedure containing three steps: compilation, link editing, and execution of your program. In this case, your EXEC statement would call the procedure, and all those functions would be performed automatically, but you would have to provide some additional information, such as where your program and your data are.

Processing Your Job

In order to have a job processed, the JCL statements and any related input data must be introduced to the operating system through an I/O device (input unit) chosen by the operator for this purpose. The sequence of JCL statements and input data for all the jobs being submitted through an input unit is called the input stream. The input unit can be a card reader, a magnetic tape, or a telecommunications line. In systems with MFT or MVT, the input unit can also be a direct access device.

For example, assume a PCP system where the input unit is a card reader. You submit the job you prepared to process insurance premium payments in the form of punched cards (Figure 1). This job has three steps and the first and last steps contain input data (your source program in the first step and the insurance premium transactions in the third step). The operator places your deck in the card reader (input unit) together with the decks for other jobs to be processed. (The card decks for all these jobs constitute the input stream.) The operator then "starts the reader", that is, he instructs the job management routines of the operating system to start reading the input stream. The operating system records the control statements of the first job step in a job queue data set (SYS1.SYSJOBQE) until after they are used. Then it examines the first job step and determines its needs. The first step of your job requested the COBOL compiler and defined other data sets. Therefore, the operating system determines whether there is any space available on the devices you indicated for the data sets the COBOL compiler will create during this step (for example, the object module) and whether any data sets required by the compiler are available (for example, your source program). If all data set requirements are met, the COBOL compiler is brought into main storage and given control. After your program is compiled, the operating system reads and determines the requirements of the second step, which requested the linkage editor. The operating system performs the same operations for the data sets required by the linkage editor and then brings it into main storage and gives it control. After the linkage editor produces the load module, the operating system reads and processes the third step. Its requirements are determined and your program is brought into main storage and given control.

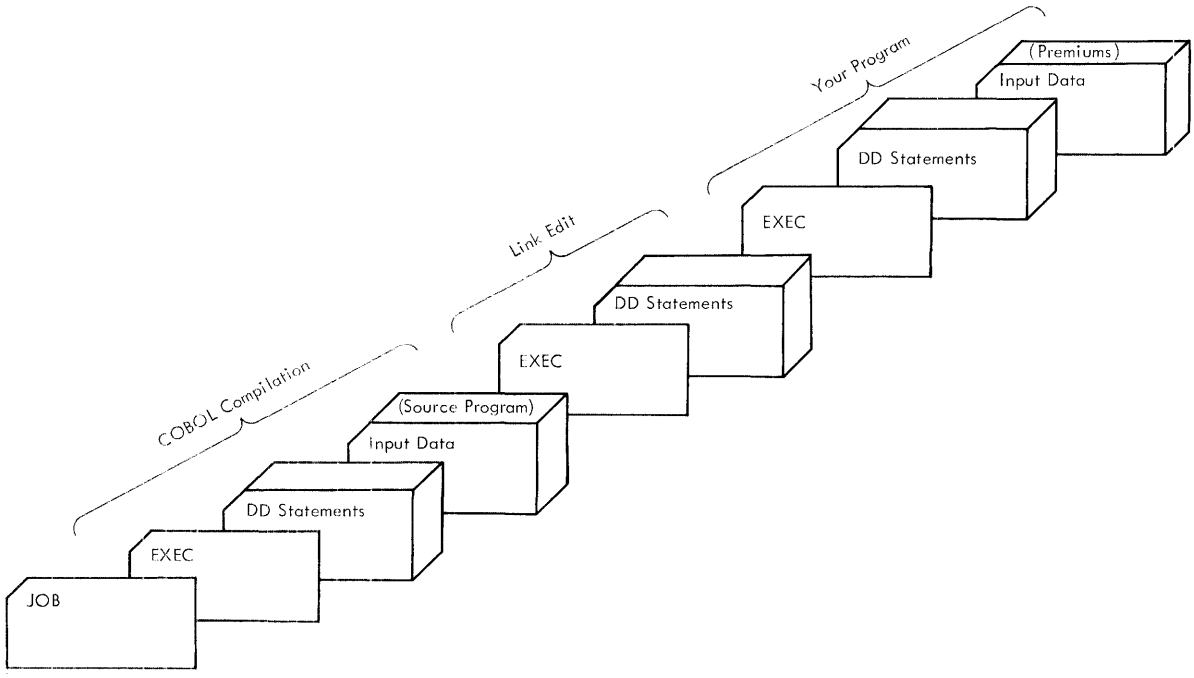


Figure 1. Your Program

After your job is completed, the operating system starts processing the next job in the input stream. If there were errors in your job, or if the resources you requested were not available (for example, your program did not fit in the available main storage), the operating system would have written appropriate messages and terminated your job before starting the next one.

Systems with PCP can have only one input stream, and the jobs are processed one step at a time as described in the previous example. Systems with MFT or MVT can have more than one input stream and can read control statements and data for more than one job. The control statements are recorded in the job queue data set (SYS1.SYSJOBQE) and the data is recorded in temporary data sets on direct access devices.

MFT and MVT allow you to assign classes and priorities to your jobs. These classes and priorities let you classify your jobs according to their characteristics and importance. Therefore, the system can increase performance by balancing the mix of jobs that are executed at a given time according to the resources they require.

There can be up to 15 job classes in your installation. These classes are designated by the letters A through O. The type of job assigned to each class is arbitrary and should be determined by each installation. For example, some installations may assign a class to each of the following types of job:

- Jobs that use a large amount of main storage.
- Jobs that run for a long time.
- Teleprocessing jobs.

Within each class you may assign priorities to determine the order of execution. For example, in the class of "jobs that run for a long time" you may wish to assign a higher priority to the weekly payroll program than to the monthly inventory analysis program.

The MFT or MVT system reads the various input streams and separates the jobs in each stream by class. The jobs are enqueued on one of 15 available job queues, corresponding to their class. (All 15 job class queues are in SYS1.SYSJOBQE.) The execution of each job within the queue is then determined by the priority within the class. Jobs of equal priority are enqueued on a first-in-first-out (FIFO) basis.

Using these classes and priorities, the system can process more than one job at the same time. Each job is executed one step at a time and steps of different jobs can be interleaved. For example, if while the system is executing a job that runs for a long time enough resources are available to process teleprocessing jobs, several teleprocessing jobs can also run. The teleprocessing jobs are selected according to their priority. Once the long-running job is finished, the system might select a job that uses a large amount of main storage. The system determines whether the resources it has left are sufficient for a high priority job in one or more of the other queues. If they are, that job (or jobs) is selected and processed at the same time.

Job Control Language Statements

There are nine JCL statements:

1. Job (JOB) statement
2. Execute (EXEC) statement
3. Data definition (DD) statement
4. Delimiter statement
5. Null statement
6. Procedure (PROC) statement
7. Procedure end (PEND) statement
8. Comment statement
9. Command statement

Parameters coded on these JCL statements help the job scheduler to regulate the execution of jobs and job steps, retrieve and dispose of data, allocate I/O resources, and communicate with the operator.

Summary of the JCL Statements

The job statement (or JOB statement) marks the beginning of a job and, when jobs are stacked in the input stream, marks the end of the JCL statements for the preceding job. It may contain accounting information for use by your installation's accounting routines, give conditions for early termination of the job, and regulate the display of job scheduler messages. In systems with MFT or MVT, you can use parameters to assign job priority, to request a specific class for job scheduler messages, to hold a job for later execution, and to limit the maximum amount of time the job may use the central processing unit (CPU). With MVT, you can also specify the amount of main storage to be allocated to the job.

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 can also provide job step accounting information, give conditions for bypassing or executing the job step, and pass control information to a processing program. In systems with MFT or MVT, a parameter allows you to assign a limit on the CPU time used by a job step. In systems with MVT, you can use an additional parameter to 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 I/O resources. DD statement parameters identify the data set, give volume and unit information, and describe the data set's labels and physical attributes.

The delimiter statement (or /* statement) and the null statement are markers in an input stream. The delimiter statement is used to separate data placed in the input stream from subsequent JCL statements. The null statement can be used to mark the end of the JCL statement and data for a job.

The PROC statement may appear as the first JCL statement in a cataloged or in-stream procedure. For cataloged procedures, the PROC statement is used to assign default values to symbolic parameters defined in the procedure. For in-stream procedures, it is used to mark the beginning of the procedure.

The PEND statement is used to mark the end of an in-stream procedure.

The comment statement can be inserted before or after any JCL statement that follows the JOB statement and can contain any information thought helpful by the person who codes the JCL statements.

The command statement is used 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 Input Stream

The input stream is the sequence of JCL statements and data submitted to the operating system for processing. Each job in the input stream has its own set of JCL statements and data. Jobs are bounded by JOB statements. Each JOB statement marks the beginning of one job and the end of the preceding job. The end of a job can also be marked by a null statement. Figure 2 shows a group of jobs and their boundaries.

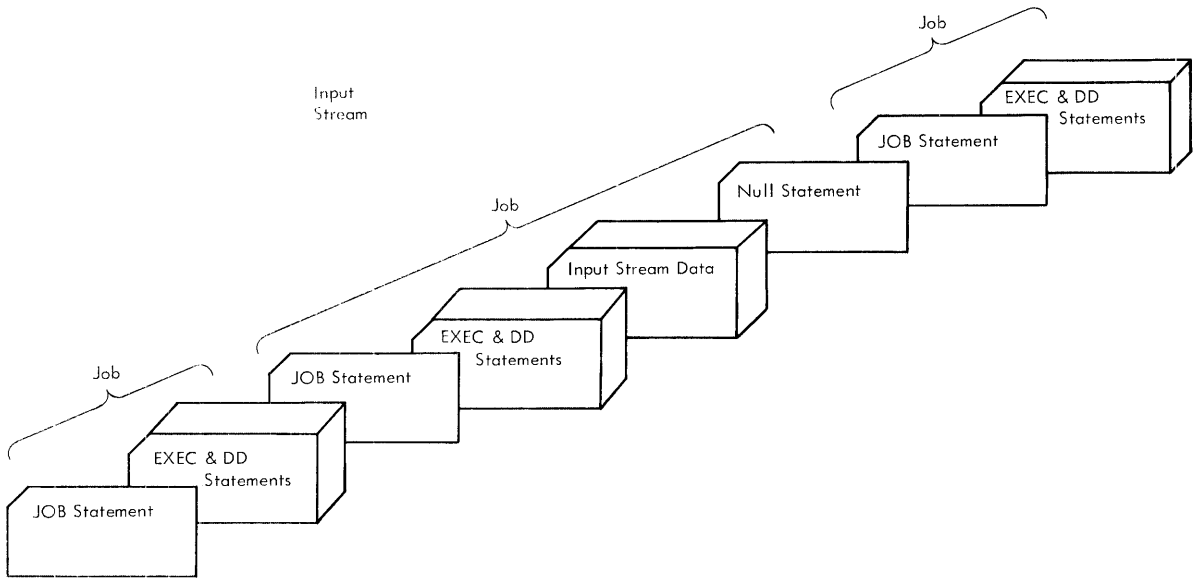


Figure 2. Defining Job Boundaries

Each job consists of one or more job steps. Job steps are bounded by EXEC statements. Each EXEC statement marks the beginning of a job step and the end of the preceding step. The end of the last step in the job is marked by the JOB statement of the following job, or by a null statement.

The EXEC statement is followed by the DD statements that define the data sets required by the job step. If one of the data sets is in the input stream, the data must be preceded by a DD statement and followed by a delimiter (/*) statement. (In systems with MFT or MVT, a DD statement and delimiter statement are not always required to bound data in the input stream. However, it is good practice to do so and they will be shown in the examples in this manual.) If the last DD statement in a job defines data in the input stream, a null statement should be used to mark the end of the job.

Figure 3 shows a group of jobs and their associated steps and data.

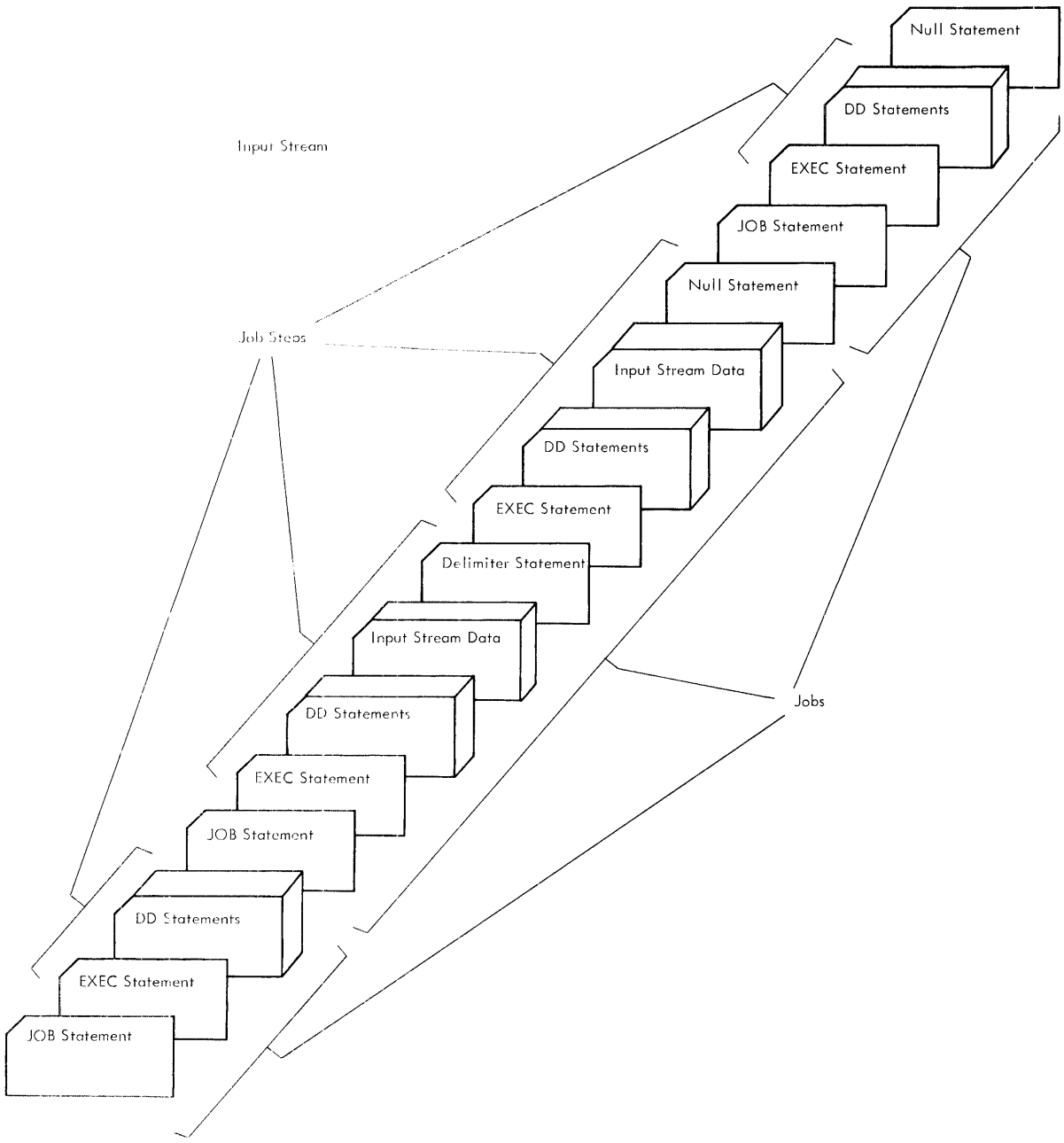


Figure 3. Defining Job Step Boundaries

Coding Conventions

The job control statements must conform to certain conventions. That is, you must follow certain simple rules when coding JCL for your job. These rules concern fields in the JCL statements, spacing of those fields, continuing the statements onto additional cards or card images, coding special characters, using the "backward-reference" facility, the notation used to define JCL statements in this publication, and a coding form for JCL statements.

Fields in the JCL Statements

JCL statements contain five fields:

1. identification
2. name
3. operation
4. operand
5. comments

The appropriate identification is required for each statement. In some statements, one or more of the remaining four fields are omitted.

Figure 4 shows the fields in each statement.

Statement	Fields
Job	//name JOB operand[1] comments[1]
Execute	//name[1] EXEC operand comments[1]
Data definition	//name[1] DD operand comments[1]
PROC(cataloged)	//name[1] PROC operand comments[1]
PROC(in-stream)	//name PROC operand[1] comments[2]
PEND	//name[1] PEND comments[1]
Command	// operation operand comments[1]
Delimiter	/* comments[1]
Null	//
Comments	//* comments

[1]Optional
[2]Optional -- Do not code comments unless you code one or more operands.

Figure 4. JCL Statement Fields

The identification field indicates that the statement is a JCL statement. It contains identifying characters (//,/*, or //*) which must begin in column 1.

The name field identifies the JCL statement so that other statements or system control blocks can refer to it. It can range from one to eight characters in length, and can contain any alphameric or national (@, $, and #) characters. However, the first character of the name must be a letter or national character and must be in column 3.

The operation field specifies the type of JCL statement, or, in the case of the command statement, the command. It can contain only one of

a set of prescribed operations or commands. The operation field has no column requirements, but must be preceded and followed by at least one blank.

The operand field contains parameters separated by commas. Parameters are composites of prescribed words (keywords) and variables for which information must be substituted. The operand field has no fixed length or column requirements, but must be preceded and followed by at least one blank. (Note that you must not code blanks between the parameters in the operand field.)

The comments field can contain any information thought helpful by the person who codes the control statement. It has no fixed length or column requirements, but must be separated from the operand field by at least one blank.

Except for the comment statement, identifying characters and fields are contained in columns 1 through 71 of the control statement. If the total number of characters exceeds 71, code the excess characters on one or more continuation statements. In the comments statement, comments can be coded in columns 4 through 80. The comments cannot be continued onto another statement. (If you need more space for comments, use several consecutive comments statements.)

Parameters in the Operand Field

The operand field is made up of two types of parameters: positional parameters and keyword parameters. A positional parameter is characterized by its position in the operand field in relation to other parameters; a keyword parameter is positionally independent with respect to others of its type, and is characterized by a keyword followed by an equal sign and variable information (a keyword parameter). Both positional parameters and the variable information in keyword parameters can be in the form of a list of several items (subparameters) of information. You must not code blanks between the parameters or subparameters of the operand field.

Positional parameters must be coded in the operand field in a specific order and before any keyword parameters are coded. The absence of a positional parameter is indicated by a comma coded in its place. However, if the absent positional parameter is the last one, or if all later positional parameters are also absent, you need not code replacing commas. If all positional parameters are absent from the operand, you need not code any replacing commas.

Keyword parameters must follow the positional parameters, but can be coded anywhere in the operand field with respect to each other. Because of this positional independence, you need not indicate the absence of a keyword parameter.

A positional parameter or the variable information in a keyword parameter sometimes assumes the form of a list of subparameters. Such a list may be composed of both positional and keyword subparameters that follow the same rules and restrictions as positional and keyword parameters. You must enclose a subparameter list in parentheses, unless the list reduces to a single subparameter.

The EXEC statement and DD statements associated with cataloged procedures can contain one other type of parameter -- a symbolic parameter. A symbolic parameters is characterized by a name preceded by an ampersand (&). A symbolic parameter stands as a symbol for a parameter, a subparameter, or a value. Symbolic parameters allow you to make any information in the operand field of a procedure EXEC

statement or DD statement variable. The value to be assumed by a symbolic parameter is coded on the EXEC statement that invokes the procedure. This value is in effect only during that execution of the procedure.

Spacing JCL Statement Fields

Except for the identifying characters in columns 1 and 2, or columns 1, 2, and 3, and the name field beginning in column 3, control statement fields can be written in free form, that is, they need not begin in a particular column. The only requirement is that you separate the name, operation, operand, and comments fields by at least one blank. Since a blank serves as a field delimiter, the operand field must be coded continuously, that is, you cannot code blanks between parameters.

Continuing JCL Statements

Except for the comment statement, JCL 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 you wish to place parameters on separate cards, you must follow the operating system continuation conventions. To continue an operand field:

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

2. Separate any comments from the interrupted field with at least one blank. Comments must end at or before column 71.

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

4. Continue the interrupted operand beginning in any column from 4 through 16. If columns 4 through 16 are left blank, the remainder of the card or card image is treated as a comment and the continuation of the preceding statement is assumed to be complete.

Note: Optionally, a nonblank character can be coded in column 72 of the card to indicate that the operand field continues on the next card.

Comments 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, at or before column 71.

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.

Note that comments written on a comments (//*) statement cannot be continued. You should use several comments statements if you need more space for comments.

Figure 5 illustrates the continuation conventions.

```
//NAME DD POSITIONALPARAMETER,KEYWORD=VALUE,  COMMENT
//  KEYWORD=(VALUE1,VALUE2,VALUE3,          COMMENT
//           VALUE4,VALUE5)    ANOTHER COMMENT                    X
//  MORE COMMENTS WITHOUT OPERANDS REQUIRE                        X
//  NONBLANK CHARACTER IN COL.72 OF PRECEDING CARD
//* THIS IS A COMMENTS STATEMENT, IN ORDER
//* TO CONTINUE IT USE ANOTHER COMMENTS STATEMENT
```

Figure 5. Continuing JCL Statements

Coding Special Characters

Alphameric, national, and special characters are used in writing JCL statements. Table 2 defines the alphameric, national, and special character sets. Special characters are used in JCL to delimit parameters and fields, and to perform other syntactical functions. With the exception of the cases listed below, parameters must be coded in alphameric and national characters.

Table 2. Character Sets

Character Set	Contents	
Alphameric	Letters	A through Z
	Numbers	0 through 9
National	"At" sign	@
	Dollar sign	$
	Pound sign	#
Special	Comma	,
	Period	.
	Slash	/
	Apostrophe	'
	Left parenthesis	(
	Right parenthesis	)
	Asterisk	*
	Ampersand	&
	Plus sign	+
	Hyphen	-
	Equal sign	=
	Blank	

Seven parameters are exempt from the special character rule, that is, they can contain special characters as well as alphameric and national characters:

1. The accounting information on the JOB statement.
2. The programmer's name on the JOB statement.
3. The checkid field of the RESTART parameter on the JOB statement.
4. The ACCT parameter on the EXEC statement.
5. The PARM parameter on the EXEC statement.

6. The DSNAME parameter on the DD statement.

7. The volume serial number in the VOLUME parameter on the DD statement. However, when using various typewriter heads or printer chains, difficulties in volume serial recognition may arise if you use other than alphameric characters in the VOLUME parameter.

If any of these parameters contain special characters, you must notify the system of this by enclosing the item that contains the special characters in apostrophes (5-8 punch), e.g., ACCT='123+456'. If one of the special characters is in apostrophe, you must identify it by coding two consective apostrophes in its place, for example 'O''NEILL'.

In five cases, parameters can contain special characters and not be enclosed in apostrophes:

1. The programmer's name on the JOB statement can contain periods without being enclosed in apostrophes, e.g., T.JONES.

2. In the UNIT parameter of the DD statement, the unit address can contain a slash and the unit type can contain a hyphen without being enclosed in apostrophes, e.g., UNIT=293/5, UNIT=2400-2.

3. The account number and items of accounting information on the JOB statement, the ACCT parameter on the EXEC statement, and the volume serial number in the VOLUME parameter on the DD statement can contain hyphens without being enclosed in apostrophes.

4. The DSNAME parameter on the DD statement can contain hyphens without being enclosed in apostrophes. If the DSNAME parameter is a qualified name, it can contain periods without being enclosed in apostrophes. If the statement identifies a generation of a generation data group, the generation number in the DSNAME parameter can contain a plus or minus (hyphen) sign without being enclosed in apostrophes. If the DD statement defines a temporary data set, the DSNAME parameter can contain, as the first two characters, ampersands without being enclosed in apostrophes.

5. Any parameter that makes a backward reference can contain an asterisk and periods without being enclosed in apostrophes. (See "Backward References" below.)

Backward References

Many parameters in the JCL statements allow you to use a backward-reference facility to fill in information. This backward-reference facility permits you to copy previously coded information or refer to DD statements that appear earlier in the job. Most backward-references are of the form *.stepname.ddname, where the term "stepname" identifies an earlier job step and "ddname" identifies the DD statement to which you are referring. The named DD statement must be contained in the named job step. If the DD statement appears in the same job step as the reference, you can eliminate the term stepname, i.e., *.ddname.

When the DD statement is contained in a cataloged procedure step, you may want to refer to it in a later job step outside the procedure. In order to refer to it, you must give both the name of the job step that invokes the procedure and the name of the procedure step that contains the DD statement, i.e., *.stepname.procstepname.ddname.

Notation for Defining JCL Statement Parameters

The formats of the parameters described in this publication for the JOB, EXEC, and DD statements appear in the chapters on those statements. Notations used in the format descriptions are described below.

1. Uppercase letters and words are coded on the JCL statement exactly as they appear in the format description, as are the following characters.

ampersand	&
asterisk	*
comma	,
equal sign	=
parentheses	()
period	.

2. Lowercase letters, words, and symbols appearing in the format description represent variables for which specific information is substituted when the parameter is coded.

   For example, PRTY=priority is the format description for the PRTY parameter. When you code the PRTY parameter on a JOB statement, you substitute a number for the word "priority."

3. Braces {} are a special notation and are never coded on a JCL statement. Braces are used to group related items; they indicate that you must code one of the items.

   For example, {TRK / CYL / block size} is part of the format description for the SPACE parameter. When you code the SPACE parameter, you must code either TRK, CYL, or a substitute for "block size," which would be a number.

4. Brackets [] are a special notation and are never coded on a JCL statement. Brackets indicate that the enclosed item or items are optional and you can code one or none of the items.

   For example, [,DEFER] is part of the format description for the UNIT parameter. When you code the UNIT parameter, you can include ,DEFER in the UNIT parameter or omit it.

   An example of more than one item enclosed in brackets is

   [EXPDT=yyddd / RETPD=nnnn]

   which is part of the format description for the LABEL parameter. When you code the LABEL parameter, you can include either EXPDT=yyddd or RETPD=nnnn in the LABEL parameter or omit both.

   Sometimes, one of a group of items enclosed in brackets is a comma. You code the comma when none of the other items in the group is used and a following part of the parameter is still to be coded. The comma indicates to the system that you have not selected to code any of the items enclosed in brackets.

For example, [,progname][,form number]) is part of the format description for the SYSOUT parameter in systems with MFT and MVT. When you code the SYSOUT parameter, you have the option of coding both ",progname" and ",form number", omitting both, or coding only one. The comma enclosed in brackets with ",progname" must be coded when ",progname" is not coded but ",form number" is coded; that is, you would code: ,,form number).

Sometimes, one of a group of items enclosed in brackets is underscored. This means that the underscored item is assumed if you omit coding any of the items in the group.
For example, [,DELETE / ,KEEP / ,CATLG] is part of the format description of the DISP parameter. If you omit coding any of the items in the group, the term ",DELETE" is assumed by the system.

5. An ellipsis ... (three consecutive periods) is a special notation and is never coded on a JCL statement. An ellipsis is used to indicate that the preceding item can be coded more than once in succession.

   For example, COND=((code,operator),...) is the format description for the COND parameter on the JOB statement. The ellipsis indicates that (code,operator) can be repeated.

Coding Form

For your convenience in coding JCL statements, you can use Form N74167, a punch card containing formatted lines, each representing a different type of statement. (See Figure 6.) Some of the lines can be used for concatenations, overrides, and continuation statements.

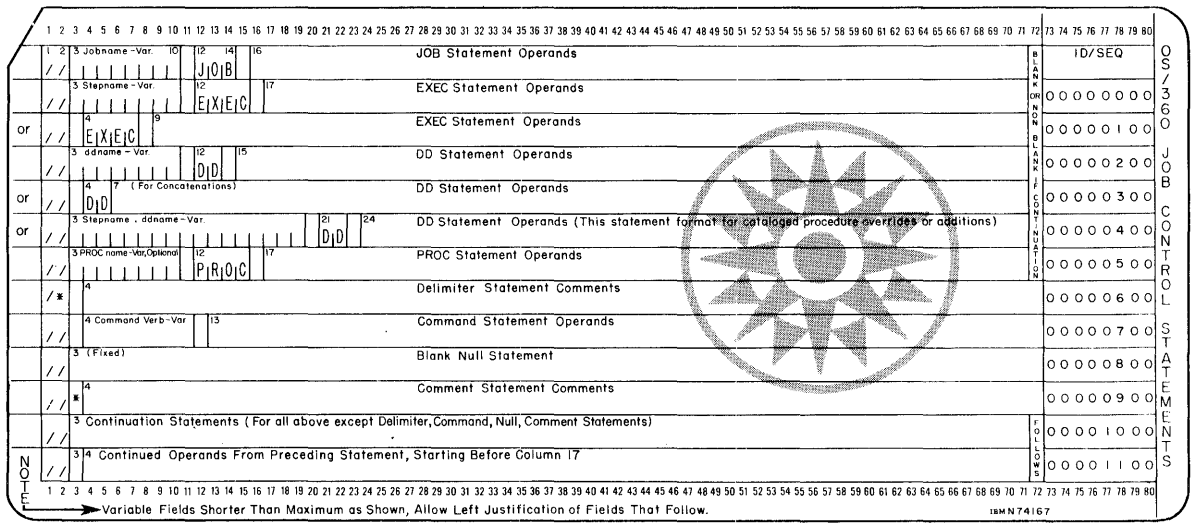


Figure 6. JCL Coding Form

Table 3. JOB Statement Parameters

Type	Parameter	Function	P/K	Comments
Installation management information	accounting information	Supplies an account number and additional accounting information to your installation's accounting routines.	P	Can be made mandatory
	programmer name	Your name.	P	Can be made mandatory
Operating system messages	MSGLEVEL information	Tells the operating system what (messages and display of JCL statements) you want about your job; e.g., device allocation messages statements.	K	
	MSGCLASS	Tells the operating system in which output class you want the MSGLEVEL messages and other job scheduler messages to appear.	K	Systems with MFT or MVT only.
Processing options	COND	Specifies the conditions under which your job is to be terminated.	K	
	TYPRUN	Prevents your job from being executed until the operator releases it.	K	Systems with MFT or MVT only.
	TIME	Specifies the maximum amount of time your job may use the CPU.	K	Systems with MFT or MVT only.
Queuing options	CLASS	Assigns your job to a job class.	K	Systems with MFT or MVT only.
	PRTY	Assigns your job to a priority within its job class.	K	Systems with MFT or MVT only.
Main Storage options	REGION	Specifies the maximum region size required by your job.	K	Systems with MVT only
	ROLL	Specifies whether your job can use the rollout/rollin function to obtain more main storage for a job step and whether your job can be rolled out.	K	Systems with MVT only
Checkpoint/ restart	RD	Specifies whether your program can use the automatic restart facility of checkpoint/restart.	K	
	RESTART	Specifies that you are resubmitting your program after it failed and that you want it restarted at some specific job step or checkpoint.	K	

Legend: P = Positional Parameter
K = Keyword Parameter

The JOB Statement

The JOB statement must be the first JCL statement of your job. It must contain a valid jobname in its name field and the word JOB in its operation field. All parameters in its operand field are optional, although your installation can establish that the account number and the programmer's name be mandatory. If no parameters are coded in the operand field of the JOB statement, no comments can be coded on the statement.

The parameters in the operand field allow you to specify six types of optional information:

1. Installation management information.
2. Operating system messages.
3. Processing options.
4. Queuing options.
5. Main storage option.
6. Checkpoint/restart.

Table 3 shows the parameters used for each type of information and their function. Please fold out Table 3 while reading this chapter.

The following paragraphs discuss the job name and the optional parameters of the JOB statement. For further details refer to the publications IBM System/360 Operating System: Job Control Language Reference.

Naming the Job

Your job must have a name. It is required by the job scheduler portion of the operating system.

The jobname is coded in the name field of the JOB statement. It can range from one to eight characters in length and can contain any alphameric or national (@, $, #) characters. However, the first character of the name must be a letter or national character and must begin in column 3. Jobnames need not be unique in a PCP environment because jobs are executed one at a time. However, no two jobs should have the same name in an MFT or MVT environment because more than one job can be executed at a time.

The following are examples of jobnames in several JOB statements:

```
//MYJOB         JOB
//MCS167        JOB
//R#123         JOB
//ANALYSIS      JOB
//@5AB          JOB
```

Command statements use certain keywords that you should not use as jobnames:

```
A
CONSOLES
DSNAME
JOBNAMES
N
```

Q
R
SPACE
STATUS
T

If you must use one of these terms as a jobname, you should instruct the operator to enclose it in parentheses if he uses it in a command statement. For example, if you have assigned the jobname T to a job and the operator wishes to display the status of the job, he must issue a command stating DISPLAY (T) . If the parentheses were omitted, he would get the usual time-of-day and date display resulting from a DISPLAY T command.

Installation Management Information

You can specify two types of installation management information in the JOB statement: accounting information and programmer's name. These two items are specified through positional parameters. The first positional parameter is for the accounting information; the second positional parameter is for the programmer's name:

```
//jobname  JOB  account,programmer
```

If you omit the accounting information, you must indicate its absence with a comma:

```
//jobname  JOB  ,programmer
```

If you omit both the accounting information and the programmer's name and you are going to specify other parameters (such as MSGLEVEL) you do not need to indicate their absence with commas:

```
//jobname  JOB  MSGLEVEL=...
```

The accounting information or programmer's name or both can be made mandatory by the installation when the operating system is generated (PCP) or in the input reader procedure (MFT or MVT). That is, your job will fail if you do not write them in your JOB statement. Your manager or supervisor should inform you of this requirement.

Accounting Information Parameter

The first positional parameter of the JOB statement allows you to supply accounting information. It has the following format:

```
┌──────────────────────────────────────────────────────────────────┐
│  ([acct#][,additional accounting information])                   │
└──────────────────────────────────────────────────────────────────┘
```

Replace the term "acct#" with the account number to which you want the job charged; replace the term "additional accounting information" with other items required by your installation's accounting routines. Your manager or supervisor should tell you exactly how to code the accounting information parameter. The following are general rules for coding the accounting information.

- The account number and each item of accounting information are considered subparameters and each must be separated by a comma. The job accounting information may be enclosed in either parentheses or

apostrophes (5-8 punch) as long as none of the subparameters contains special characters.
For example:

```
(12A75,DEPTD58,705) or '12A75,DEPTD58,705'
```

- If a subparameter contains special characters (except hyphens), you can either, 1) enclose that subparameter in apostrophes and all the job accounting information in parentheses, or 2) enclose all the job accounting information in apostrophes.
For example:

```
(12A75,'DEPT/D58',705) or '12A75,DEPT/D58,705'

(12A75,DEPT-D58,705) or '12A75,DEPT-D58,705'
```

- If the special character is an apostrophe, it must be shown as two consecutive apostrophes.
For example, show DEPT'D58 as:

```
(12A75,'DEPT"D58',705) or '12A75,DEPT"D58,705'
```

- If job accounting information contains only an account number and the number does not contain special characters, you need not enclose the number in parentheses or apostrophes. If the account number does contain special characters, enclose the number in apostrophes.
For example:

```
12A75
'12A.75'
'12A"75'
```

- If job accounting information does not contain an account number, you must indicate its absence by coding a comma preceding the additional accounting information.
For example:

```
(,DEPTD58,705)
(,'DEPT/D58',705) or ',DEPT/D58,705'
```

Job accounting information can consist of up to 142 characters indicating the commas that separate the subparameters. This means you may have to continue the information onto another card. If the information is to be continued, enclose it in parentheses. Any subparameter enclosed in apostrophes must be coded on one card. To continue the information, follow the continuation conventions as outlined in "Continuing Control Statements" in the section "Coding Conventions". The following is an example of continuing job accounting information onto another card.

```
//YOURJOB    JOB    (12A75,'DEPT/D58',
//                  705)
```

Programmer's Name Parameter

The second positional parameter indicates your name or identification to your installation's accounting routines. This parameter must follow the accounting information. The following are rules for coding the programmer's name:

- The number of characters in the name, including blanks, cannot exceed 20.

- If the name contains special characters, other than periods, it must be enclosed in apostrophes.

For example:

```
T.JONES
'T JONES' (blanks are special characters)
'SP/4 T.JONES'
```

- If the special character is an apostrophe, it must be shown as two consecutive apostrophes.
  For example the names O'Neill and J.O'Brien must be shown as:

```
'O"NEILL'
'J.O"BRIEN'
```

- If the job accounting information is not coded, you must indicate its absence by a comma preceding the programmer's name.
  For example:

```
//MCS  JOB  ,MARIA
```

Operating System Messages

The job scheduler portion of the operating system produces messages about your job on the system output unit. These messages are a listing of the JCL statements in your job and allocation/termination messages indicating the resources used by your job. The MSGLEVEL parameter lets you choose the type of messages to be produced about your job.

If your job is to run under MFT or MVT, you can also specify the class of output unit on which those messages and other job scheduler messages are to appear. The MSGCLASS parameter allows you to choose from the output classes established in your installation's output writer procedures. Your manager or supervisor should tell you which output classes are available. (For information on output writer procedures, see the section "System Reader, Initiator and Writer Cataloged Procedures" in the publication IBM System/360 Operating System: System Programmer's Guide.)

Do not use the MSGCLASS parameter if your job is to run under PCP. (If you use MSGCLASS, it is ignored by the system.) Job scheduler messages are always produced on the standard system output unit.

MSGLEVEL Parameter

The MSGLEVEL parameter tells the job scheduler what information you want as output from your job. You can request the following output:

- The JOB statement.
- All input JCL statements.
- All cataloged procedure statements for procedures invoked by the job's steps, and the internal representation of procedure statements after symbolic parameter substitution.
- Allocation, disposition, and allocation recovery messages (allocation/termination messages). Allocation/termination messages have the prefix IEF and are listed in the publication IBM System/360 Operating System: Messages and Codes.

The format of the MSGLEVEL parameter is:

```
MSGLEVEL=([statements][,messages])
```

Replace "statements" with one of the following:

0- only the JOB statement is to be displayed.

1- all JCL statements in your job, cataloged procedure statements, and the internal representation of statements after symbolic substitution are to be displayed.

2- only the JCL statements in your job are to be displayed (cataloged procedure statements will not appear).

Replace "messages" with one of the following:

0- no allocation/termination messages are to appear, unless the job abnormally terminates. If this occurs, these messages will appear as output.

1- all allocation/termination messages are to appear.

If you omit the MSGLEVEL parameter or one of the subparameters, a default value is assumed. In systems with PCP, the default value established during system generation is used. In systems with MFT or MVT, the default value in the input reader procedure is assumed. Your manager or supervisor should tell you the defaults chosen in your installation.

For example, if you want only the JOB statement and no allocation/termination messages displayed, code:

```
MSGLEVEL=(0,0)
```

If you want only the JCL statements in your job and all allocation/termination messages displayed, code:

```
MSGLEVEL=(2,1)
```

If you want to use your installation's default for JCL messages and no allocation/termination messages displayed, code:

```
MSGLEVEL=(,0)
```

If you want all your JCL statements, cataloged procedures, and internal representation of statments, and your installation's default for allocation/termination messages, code:

```
MSGLEVEL=1
```

If you want your installation's defaults for both JCL statements, and allocation/termination messages, omit the MSGLEVEL parameter.

MSGCLASS Parameter (Systems With MFT or MVT Only)

Systems with MFT or MVT can process several jobs at the same time. If only one output class were used, the job scheduler messages and other output from all the jobs would be interspersed on the same output device. The MSGCLASS parameter allows you to route the job scheduler messages to an output class other than the normal message class, A.

The format of the MSGCLASS parameter is:

```
MSGCLASS=x
```

Replace "x" with a letter (A-Z) or a number (0-9). Your job scheduler messages will be produced on a device assigned to the class you selected. If you omit the MSGCLASS parameter, job scheduler messages are routed to the default output class specified in the output reader procedure. The default for the MSGCLASS parameter is A unless changed by your installation.

Your manager or supervisor should tell you which output classes are available in your installation. Some of these output classes are standard and some are reserved for special uses. You may have to notify the operator if you are using a special output class in your job because he has to "start an output writer" for that output class in order to obtain the output.

Processing Options

There are three processing options available to your job through JOB statement parameters.

The COND parameter specifies conditions for terminating your job if one or more of its job steps are unsuccessful. For example, if your job is a compile-link edit-go job, you can have your job terminated after an unsuccessful compilation or link edit. That is, the system will not try to process the remaining steps in your job thus saving computing time.

The TYPRUN parameter is only available in systems with MFT or MVT. This parameter prevents your job from being executed until released by the operator. For instance, you may tell the operator not to release your job until a certain other job has been processed.

The TIME parameter is only available in systems with MFT or MVT. It lets you set the maximum amount of time your job may use the CPU. This parameter lets you find out how long your job uses the CPU and limits the CPU time wasted if your program goes into a "loop".

COND Parameter

The operating system determines whether a job is to be discontinued after a given job step by comparing the return code produced by that job step to the conditions specified with the COND parameters. A return code is a number determined by the operating system or by the processing program which indicates the relative "success" of the job step. The return codes of the operating system and IBM-supplied processing programs are fixed numbers with specific meanings. They are listed in the publication IBM System/360 Operating System: Messages and Codes and in the publications associated with each processing program.

Only those user processing programs written in the assembler language or PL/I can set return codes for testing. The user return codes are usually standardized in each installation. For example, each step in your installation's payroll program may have its own set of return codes. One return code for a given job step may indicate that all payroll records were successfully processed while another may indicate that there were faulty input records. You can set up the COND parameter

so that the job is discontinued if the return code that indicates faulty records is produced by that job step.

Not all return codes indicate either success or failure. For example, in the case of a compiler one return code can indicate no errors during compilation, a second code can indicate that the minor errors encountered are not likely to prevent link editing and execution of the compiled program, a third code can indicate that the major errors encountered will probably cause further processing of the compiled program to fail, and a fourth code can indicate that the compilation process has terminated abnormally. The COND parameter allows you to discontinue the job if any of these return codes are produced. You may choose to continue processing only if no errors are found or, for debugging purposes, you may choose to continue processing even if major errors are found.

Note: If any job step is abnormally terminated (ABEND), all subsequent steps are bypassed unless the COND parameter of the EXEC statement is used to prevent it. (See the section on "The EXEC Statement.") If you want to restart the same step that terminated abnormally you can use the checkpoint/restart facility of the operating system. (See "Checkpoint/Restart" in this chapter.)

The format of the COND parameter is:

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

Replace "code" with any decimal number from 0 to 4095. 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)

If you coded COND=((50,GE),(60,LT)), it would read "if 50 is greater than or equal to a return code, or 60 is less than a return code, I want the remaining job steps bypassed." In other words, the job continues as long as return codes range from 51 through 60. If you want to make only one return code test, you need not code the outer parentheses. For example, COND=(8,NE). A maximum of eight conditions can be established. For example, if you code: COND=((5,GT),(8,EQ),(12,EQ),(17,EQ),(19,EQ), (21,EQ), (23,LE)) your job will continue only if the return codes are: 5,6,7,9,10,11,13,14,16,18,20, or 22.

The tests you specify with the COND parameter are made to the return code, if any, produced by each step in your job. You can best take advantage of this parameter when the return codes of each job step have compatible meanings. For example, a return code of 4 from the ALGOL compiler indicates that the source program was compiled and some minor errors were found; the same return code of 4 from the linkage editor indicates that a load module was produced, but an error which may cause failure at execution time has been found. If you want to take a chance and continue processing even if small errors are found, you should code COND=(4,LE), that is, the job will terminate if the return code of any step is greater than 4. If you only want to continue processing if no errors are found, you should code COND=(4,LT), that is, the job will terminate if the return code of any step is greater than or equal to 4. (All codes greater than 4 indicate major errors for both the ALGOL compiler and the linkage editor.)

If the same return code has different meanings in different job steps, or if you want to take different actions according to which job step produced the return code, you should use the COND parameter of the EXEC statement to set up conditions for individual job steps.

If you omit the COND parameter from the JOB statement, no return code tests are performed throughout the job. If you want return codes tested for a given job step, use the COND parameter of the EXEC statement for that job step. If the COND parameter is not used in either the JOB or the EXEC statements, no return code tests are performed and the system will try to execute each step in the job.

Note: The COND parameter of the EXEC statement is slightly different from the COND parameter of the JOB statement. See the section on "The EXEC statement". Examples of using the COND parameter in both the JOB and EXEC statements are also shown in that section.

TYPRUN Parameter (Systems With MFT or MVT Only)

In systems with MFT or MVT you cannot predict whether a job in a job class queue will be selected for execution before another job in a different job class queue. (Job classes are discussed in "Processing Your Job" in the section "Part I: Introduction to the Job Control Language".) If you have a job that must be executed after another one, you can use the TYPRUN parameter to prevent execution of your job until the first one completes processing. You must tell the operator that you have held your job and the name of the job that must be processed before yours. Then, when the console indicates that the first job has ended, the operator should issue a command to RELEASE your job for processing.

The format of the TYPRUN parameter is:

```
TYPRUN=HOLD
```

If both jobs are in the same job class queue, it is not enough to assign a higher priority to the job that must be executed first. You must also use the TYPRUN=HOLD parameter, because the higher priority controls only the selection sequence and does not guarantee that the first job will complete execution before the second is selected.

In systems with PCP, all you have to do is place the jobs in the input queue in the order in which they are to be executed. Do not use the TYPRUN=HOLD parameter in systems with PCP.

TIME Parameter (Systems With MVT Only)

The TIME parameter specifies the maximum amount of time a job may use the CPU. Two benefits of the TIME parameter are that it allows you to find out through messages how long the job uses the CPU (CPU time used appears on the output listing), and it helps limit the CPU time wasted by a step that goes into a loop. Normally, a job that exceeds its time limit is terminated. However, if the System Management Facilities option is included in the system and a user exit routine is provided, this routine can extend the time limit so that processing can continue. (The System Management Facilities option is described in the publication IBM System/360 Operating System: Concepts and Facilities. User exit routines to be used with the System Management Facilities option are discussed in the publication IBM System/360 Operating System: System Programmer's Guide.)

The format of the TIME parameter is:

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

Replace the terms "minutes" and "seconds" with the maximum number of minutes and seconds that the job can use the CPU. The number of minutes must be less than 1440 (24 hours); the number of seconds must be less than 60. That is, the maximum time you can specify is TIME=(1439,59).

If your job may require use of the CPU for more than 1439 minutes, code TIME=1440 to eliminate the time limit.

If the CPU time limit is given in minutes only, you need not code the parentheses. For example, "twelve minutes" is coded TIME=12. If the CPU time limit is given in seconds only, you must code both the parentheses and a comma to indicate the absence of minutes. For example, "thirty seconds" is coded TIME=(,30).

If you omit the TIME parameter on the JOB statement, there is no CPU time limit assigned to the job; however, each job step is still timed. (See the description of the TIME parameter in the section "The EXEC Statement".)

Coding TIME=1440 also lifts the restrictions on the amount of time a job step may remain in a wait state. If the System Management Facilities option is included in the system, the installation determines this time limit. In this case, a job step remaining in a wait state for more than the established time limit causes termination of the job unless a user-provided exit routine extends the wait-state time limit for that step. If the System Management Facilities option is not included, the system automatically provides a 30-minute time limit for wait states; that is, a job step remaining in a wait state for more than 30 consecutive minutes causes termination of the job. If any of the job steps should be allowed to remain in a wait state for more than the established time limit, code TIME=1440 to eliminate the time limit.

Note: You can specify different CPU time limits for each step in the job by coding the TIME parameter on the EXEC statement associated with each step, as described in the section "The EXEC Statement".

Queuing Options (Systems With MFT or MVT Only)

One of the most important features of MFT and MVT is the ability to balance the job mix by recognizing the classes and priorities assigned to jobs. Job classes and priorities are discussed in "Processing Your Job" in the section "Part I: Introduction to the Job Control Language".

There can be up to 15 job classes in your installation. The type of job assigned to each class is arbitrary and should be determined by each installation. For example, some installations may assign a class to each of the following types of jobs:

- I/O-bound jobs.
- CPU-bound jobs.
- Jobs that are being debugged.
- Jobs that use a particular device. For example, if there is only one 7-track tape drive in your installation, two programs that use that drive cannot be multiprogrammed. Therefore, those programs should be assigned to the same job class to avoid their simultaneous selection.

In general, all jobs of the same characteristics should be in the same class.

The order of execution within each class is determined by the priority assigned to each job. There can be up to 14 priorities in each class. The higher the priority the sooner your job will be executed.

Your manager or supervisor should tell you which class and priority to assign to your job.

CLASS Parameter (Systems With MFT or MVT Only)

The CLASS parameter is used to assign a job class to your jcb. The format of the CLASS parameter is:

```
CLASS=jobclass
```

Replace the term "jobclass" with a letter from A through O. For example, if your job belongs to class C, code

```
CLASS=C
```

If you omit the CLASS parameter, or code CLASS=A, the default job class of A is assigned to the job.

Note: If your installation provides time-slicing facilities in a system with MFT, the CLASS parameter can be used to make your job part of a group of jobs to be time-sliced. 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 you want your job to be time-sliced, specify a class that was assigned only to the partitions selected for time-slicing.

PRTY Parameter (Systems With MFT or MVT Only)

Default job priorities within each class are established in the input reader procedure. If you wish to assign a different priority to your job, use the PRTY parameter.

The format of the PRTY parameter is:

```
PRTY=number
```

Replace the term "number" with a decimal number from 0 through 13. The highest priority number is 13.

Whenever possible, you should 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 MVT.

Notes:

- If your installation provides time-slicing facilities in a system with MVT, the PRTY parameter can be used to make your job part of a group of jobs to be time-sliced. At system generation, the priorities of the time-sliced groups are selected. If the job priority number you specify corresponds with a priority number selected for time-slicing, then your job will be time-sliced.
- If you want to assign a different priority to a job step, you can code the DPRTY parameter on the EXEC statement associated with the step, as described in the next chapter. The priority assigned to the job applies to any step that does not use the DPRTY parameter.

Main Storage Options (Systems With MVT Only)

MVT processes several jobs at the same time in different regions of main storage. The size of each region depends on the requirements of each job. Once the processing of that job is completed, the system uses that space for other jobs. (The free space can be subdivided into smaller regions, used for a region the same size, or combined with adjacent free space for a larger region.)

Regions are contiguous areas of main storage. However, you can define a two-part region if your operating system was generated with "Main Storage Hierarchy Support". Main storage hierarchy support provides for storage hierarchies 0 and 1. If 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 during 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.

The REGION parameter allows you to specify the region size to be used by your job.

Notes:

- If you want to specify different region sizes for each step in the job, you can, instead, code the REGION parameter on the EXEC statement associated with each step, as described in the next chapter.

- In systems without main storage hierarchy support, processor storage is referred to as main storage. In systems with main storage hierarchy support, main storage comprises both processor storage (hierarchy 0) and 2361 core storage (hierarchy 1).

Sometimes a job step will run out of space in its region. Normally, this means that the step is terminated. However if the rollout/rollin option was generated for your MVT system, the system can obtain more space for your job step. This is done by "rolling out" (writing onto a direct access device) another job step that is currently being executed and allocating its region to your job step. After your step is processed, the rolled-out step is "rolled in" (read back into its region) and its processing continues.

The ROLL parameter lets you specify whether the job steps in your job can be rolled out and whether they can cause rollout of other job steps.

Note: ROLL parameters can also be coded on EXEC statements, but are superseded by a ROLL parameter coded on the JOB statement.

REGION Parameter (Systems With MVT Only)

The REGION parameter allows you:

1. To request the maximum amount of main storage to be allocated to the job. This figure must include the size of those components that are required by your program and are not resident in storage.

2. To request the amount of main storage to be allocated to the job and in which storage hierarchy or hierarchies the space is to be allocated. This request should be made only if main storage hierarchy support is specified during system generation.

The storage requirements you must consider when specifying a region size are outlined in the publication IBM System/360 Operating System: Storage Estimates.

1. The format of the REGION parameter for systems without storage hierarchies is:

```
REGION=valueK
```

   Replace the term "value" with the number of contiguous 1024-byte areas you want allocated to the job, for example, REGION=52K. This number can range from one to five digits but cannot exceed 16383. It should be specified as an even number because if you specify an odd number, the system treats it as the next highest even number. If you specify the maximum of 16383, you get 16384 bytes, however you must not specify 16384.

   If you omit the REGION parameter, the default value (as established in the input reader procedure) is assumed.

2. The format of the REGION parameter for systems with storage hierarchies is:

```
REGION=([value0K][,value1K])
```

   Replace the term "$value_0$" with the number of contiguous 1024-byte areas you want allocated to the job in hierarchy 0 (processor storage); replace the term "$value_1$" with the number of contiguous 1024-byte areas to be allocated in hierarchy 1, (2361 Core Storage), e.g., REGION=(60K,200K). Each value specified should be an even number. (If you specify an odd number, the system treats it as the next highest even number).

   The following rules apply to the hierarchy sizes:

   - If 2361 Core Storage is not present but main storage hierarchy support was specified during 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. In this case, the sum of $value_0$ and $value_1$ cannot exceed 16383. For example, REGION=(10K,4K).

   - If 2361 Core Storage is present, $value_0$, cannot exceed 16383, and $value_1$ cannot exceed 1024 if using a single Model I, or 2048, if using a single Model 2.

- If main storage hierarchy support was not generated and 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, no hierarchy 0 segment is allocated.

In systems with main storage hierarchy support, you can omit either subparameter, thereby requesting storage in only one hierarchy. If you are requesting storage only in hierarchy 1, you must code a comma to indicate the absence of the first subparameter, for example, REGION=(,52K). If you are requesting storage only in hierarchy 0, you need not code the parentheses, for example, REGION=70K.

If you omit the REGION parameter, the default value (as established in the input reader procedure) is assumed. When the default region size is assumed, storage is always allocated in hierarchy 0.

ROLL Parameter (Systems With MVT Only)

The function of rollout/rollin is to allocate additional main storage to a job step whose own region contains no more available space. 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. The ROLL parameter allows you to indicate whether or not each of your job steps can be rolled out and whether or not they can cause rollout of other job steps.

The format of the ROLL parameter is:

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

The "x" declares each job step's ability to be rolled out. If "x" is YES, each job step can be rolled out; if "x" is NO, the job steps of this job cannot be rolled out. The "y" declares each job step's ability to cause rollout of another job step. If "y" is YES, each job step can cause rollout; if "y" is NO, the job steps cannot cause rollout. (YES must be specified if you want additional main storage allocated to the job's steps if it is required.) If you include the ROLL parameter, both the "x" and "y" subparameters must be coded.

For example, ROLL=(NO,YES) indicates that your job steps cannot be rolled out but that any one of them can cause roll out.

Roll parameters can also be coded on EXEC statements, but are superseded by a ROLL parameter coded on the JOB statement. If this parameter is omitted, the default specified in the reader procedure is used. In the IBM-supplied reader procedures, the default specified is ROLL=(YES,NO).

Note: Teleprocessing jobs that use the autopoll option should not have the ability to be rolled out. A rolled out job using this option cannot be restarted properly. Therefore, you should always code ROLL=(NO,YES) or ROLL=(NO,NO) for this kind of job.

Checkpoint/Restart

When a job step abnormally terminates (ABEND), you may have to resubmit the job for execution. This means lost computer time and a delay in obtaining the desired results. To reduce these effects, you can use the restart facilities of the operating system provided your program is written in the assembler language, COBOL, or PL/I.

If a job step abnormally terminates, or if a system failure occurs in a system with MFT or MVT, the restart facilities allow you to request that the job step be restarted either at the beginning of the step (step restart) or at some point within the step (checkpoint restart). Furthermore, restart can occur automatically after abnormal termination, or it can be deferred until job is resubmitted. Automatic restarts are specified with the RD parameter, and deferred restarts with the RESTART parameter. For detailed information on the checkpoint/restart facilities, using the assembler language refer to the Publication IBM System/360 Operating System: Supervisor and Data Management Services; using COBOL, to IBM System/360 Operating System: COBOL (E) Programmer's Guide, or to IBM System/360 Operating System: COBOL (F) Programmer's Guide, or to IBM System/360 Operating System: American National Standard COBOL Programmer's Guide; using PL/I, to IBM System/360 Operating System: PL/I (F) Programmer's Guide; and also to "Appendix C: Using the Restart Facilities" of this publication, and to the publication IBM System/360 Operating System: Advanced Checkpoint/Restart Planning Guide, GC28-6708.

RD Parameter

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 either case, automatic restart can occur only if:

1. you use the RD parameter to request restart,
2. the return code returned during abnormal termination indicates that the step is eligible for restart, and
3. the operator authorizes restart.

In order for checkpoint restart to occur, the CHKPT macro instruction (assembler), or the RERUN clause (COBOL), or the CALL IHECKPT statement (PL/I) must have been executed in your processing program prior to abnormal termination. Through the RD (restart definition) parameter, you can specify that step restart can occur or that the action of the CHKPT macro instruction (assembler), or the RERUN clause (COBOL), or the CALL IHECKPT statement (PL/I) is to be suppressed.

The format of the RD parameter is:

```
RD=request
```

Replace the word "request" with:

R — To permit automatic step restart.

NC — to suppress the action of the CHKPT macro instruction, or RERUN clause, or CALL IHECKPT statement, and not to permit automatic restart

NR to request that the CHKPT macro instruction, or RERUN clause, or CALL IHECKPT statement 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, or RERUN clause, or CALL IHECKPT statement.

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

RD=R: If the processing programs used by the job do not include any CHKPT macro instructions, (assembler), or RERUN clauses (COBOL), or CALL IHECKPT statements (PL/I), RD=R allows execution to be resumed at the beginning of the step that abnormally terminates. If any of the programs do include one or more CHKPT macro instructions, or RERUN clauses, or CALL IHECKPT statements, step restart can occur if a step abnormally terminates before execution of a CHKPT macro instruction or RERUN clause, or CALL IHECKPT statement; thereafter, checkpoint restart can occur. If you cancel the effects of the CHKPT macro instructions, or RERUN clauses, or CALL IHECKPT statements before a checkpoint restart is performed, the request for automatic step restart is again in effect.

RD=NC or RD=RNC: RD=NC or RD=RNC should be specified when you want to suppress the action of all CHKPT macro instructions (assembler), or RERUN clauses (COBOL), or CALL IHECKPT statements (PL/I) included in the programs used by this job. When RD=NC is specified, neither step restart nor checkpoint restart can occur. When RD=RNC is specified, step restart can occur. RD=NC has no effect on processing if CHKPT macro instructions, or RERUN clauses, or CALL IHECKPT statements are not included in the programs.

RD=NR: RD=NR permits a CHKPT macro instruction (assembler), or RERUN clause (COBOL), or CALL IHECKPT statement (PL/I) to establish a checkpoint, but does not permit automatic step or checkpoint restarts. Instead, at a later time, you can resubmit the job and begin execution at a specific checkpoint. (The RESTART parameter is used for resubmitting a job for restart. See the next topic.) This parameter has no effect on processing if CHKPT macro instructions, RERUN clauses, or CALL IHECKPT statements are not included in the program.

If automatic restart is requested for an abnormally terminated step and restart is authorized by the operator, special disposition processing is performed. If automatic step restart is to occur, 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 with a status of NEW in the restart step are deleted. If automatic checkpoint restart is to occur, all data sets currently in use by the job are kept. (For further information on data set disposition refer to the section "The DD Statement" and to "Appendix C: Using the Restart Facilities."

If you omit the RD parameter and no CHKPT macro instructions, or RERUN clauses, or CALL IHECKPT statements are executed, automatic restart cannot occur. If you omit the RD parameter but one or more CHKPT macro instructions or RERUN clauses, or CALL IHECKPT statements are executed, automatic checkpoint restart can occur.

Notes:

- If you want to make different requests for each step in the job, you can, instead, code the RD parameter on the EXEC statement associated with each step, as described in the next chapter. If you specify the RD parameter on the JOB statement, RD parameters on the job's EXEC statements are ignored.

- When using a system with MFT or MVT, MSGLEVEL=(1,0) or MSGLEVEL=(1,1) must be coded on the JOB statement if automatic restart is to occur. If you omit this parameter, restart is not performed.

- If restart is requested, assign each step a unique step name. (Upon restart, the system searches the name field for the name of the abnormally terminated step. If two steps have the same name, restart is attempted at the first step encountered with the name regardless of whether or not this is the step that abnormally terminated.) Step names are assigned in the EXEC statement for each step.

RESTART Parameter

If your job abnormally terminated and you are resubmitting it for execution, you can use the restart facilities. 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, the CHKPT macro instruction (in assembler language), or RERUN clause (in COBOL), or CALL IHECKPT statement (in PL/I) must have been executed in your processing program during the original execution of the job. Through the RESTART parameter you can specify where execution is to be restarted.

Note: Do not use the RESTART parameter the first time you run your job.

If you want execution to be restarted at a particular job step, code the keyword parameter.

```
RESTART=stepname
```

in the operand field of the JOB statement before resubmitting the job. Replace the word "stepname" with the name of the step at which execution is to be restarted. You can replace "stepname" with an asterisk (*) if execution is to be restarted at the first job step; that is, RESTART =*.

If you want execution to be restarted at a particular checkpoint within a particular job step, code the keyword parameter.

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

in the operand field of the JOB statement before resubmitting the job. Replace the word "stepname" with the name of the step in which execution is to be restarted. You can replace "stepname" with an asterisk (*) if execution is to be restarted within the first job step. Replace the term "checkid" with the name that identifies the checkpoint within the step. (If the name contains special characters, it must be enclosed in

apostrophes. If one of the special characters is an apostrophe, identify it by coding two consecutive apostrophes in its place.)

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.) The SYSCHK DD statement is described in "Special DD Statements" in the section "The DD Statement."

If execution is to be restarted at or within a cataloged procedure step, you must give both the name of the step that invokes the procedure and the procedure step name, i.e., RESTART-stepname.procstepname or RESTART=(stepname.procstepname,checkid). If the first job step invokes a cataloged procedure and you want execution to be restarted at the first procedure step, you can replace "stepname.procstepname" with an asterisk(*).

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

Table 4. EXEC Statement Parameters

Type	Parameter	Function	P/K	Comments
Processing Program Information	PGM	NaYou me the program to be executed in this step.	P	must specify either a program name or a procedure name.
	PROC	Names the procedure to be used in this step.	P	
	Procedure Name	Alternative way of naming the procedure to be used in this step without using the PROC keyword.	P	
	PARM	Passes special control information to those programs that require it.	K	
Installation Management Information	ACCT	Supplies accounts information to your installation's accounting routines.	K	
Processing Options	COND	Specifies conditions for executing or bypassing the job step.	K	
	TIME	Specifies the maximum amount of time this job step may use the CPU.	K	Systems with MFT or MVT only.
Queueing Option	DPRTY	Assigns a dispatching priority to the job step.		Systems with MVT only.
Main Storage Options	REGION	Specifies the maximum region size required by this job step.	K	Systems with MVT only.
	ROLL	Specifies whether this job step can use the rollout/rollin function.	K	Systems with MVT only.
Checkpoint/ Restart	RD	Specifies whether this job step can use the automatic restart facility of checkpoint/restart.	K	

Legend: P = Positional parameter
K = Keyword parameter

The EXEC Statement

The EXEC statement must be the first JCL statement of each job step in your job. It must also be the first statement of each procedure step in a cataloged procedure. The EXEC statement is followed in the input stream by DD statements and data that pertain to the step.

The main function of the EXEC statement is to identify the program to be executed or the cataloged procedure to be used. For example, Figure 7 shows the input deck for a three-step job. The EXEC statement of the first step requests a program named MYPROG. The DD statements and data required by MYPROG follow the EXEC statement. The EXEC statement of the second step requests a cataloged procedure named PROCONE. When the second step is to be executed the system will use PROCONE. PROCONE has two procedure steps. The EXEC statement of the first procedure step requests a program named XXX; the EXEC statement of the second procedure step requests a program named YYY. After the cataloged procedure is used the third step of your job is executed. The EXEC statement of the third step requests a program named PRINT. The DD statements required by PRINT follow the third EXEC statement. Note that you only supply the JCL statements for your job. The cataloged procedure already exists in the procedure library (SYS1.PROCLIB). Therefore, you need not be concerned with writing the JCL statements for the cataloged procedure unless you are actually writing the cataloged procedure to place it in the procedure library. You can, however, modify cataloged procedure statements by placing the corrections in the input deck for your job. For example, in Figure 7 the DD statements that follow the EXEC statement of the second step are used to make changes in corresponding DD statements in the cataloged procedure. The methods used for modifying cataloged procedures are described in the section "Part III: Cataloged Procedures".

The EXEC statement must contain the word EXEC in its operation field. The stepname (name field) and most parameters in the operand field are optional. The only required information in the operand field is either the name of the program to be executed or the name of the procedure to be used. The parameters in the operand field allows you to specify six types of information:

1. Processing program information
2. Installation management information
3. Processing options
4. Queueing option
5. Main storage option
6. Checkpoint restart information

Table 4 shows the parameters used for each type of information and their function. Please fold out Table 4 while reading this chapter.

The following paragraphs discuss the step name and the optional parameters of the EXEC statement. For further details refer to the publication IBM System/360 Operating System: Job Control Language Reference Guide.

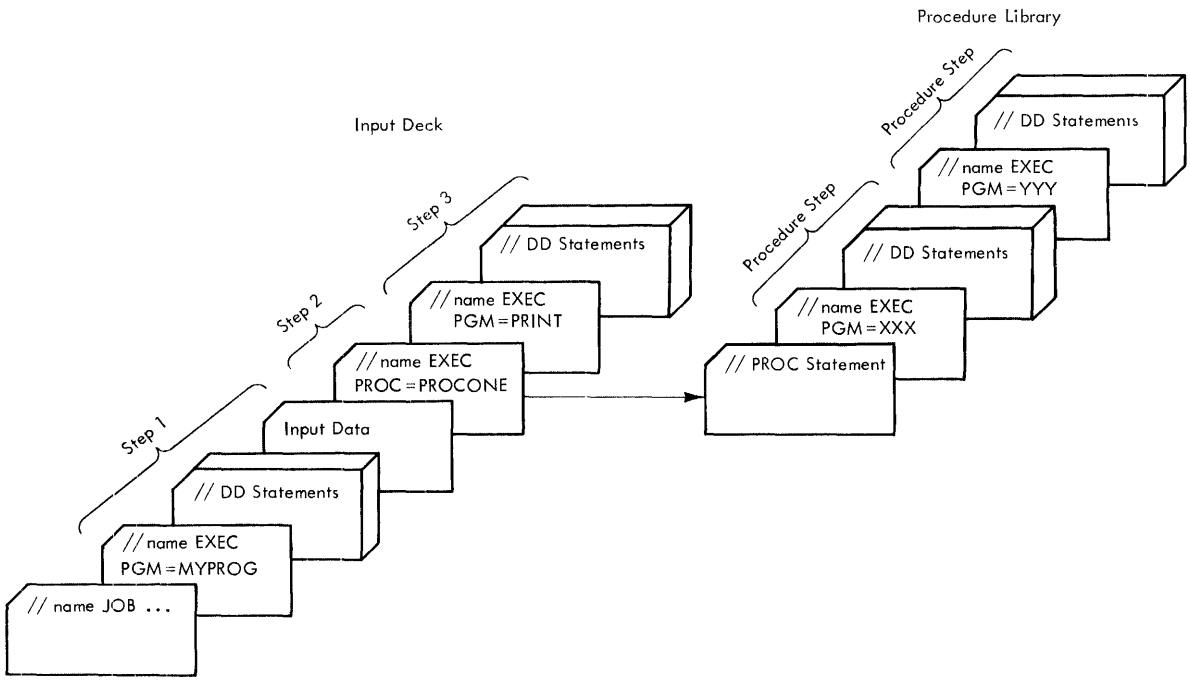


Figure 7. Using the EXEC Statement

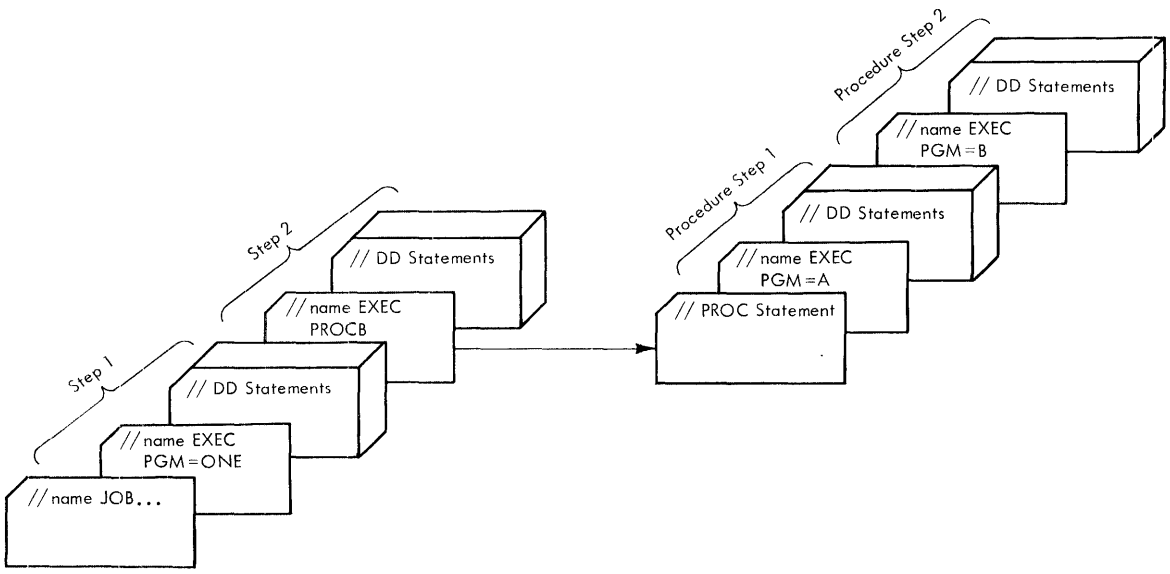


Figure 8. Modifying a Cataloged Procedure

Naming the Step

The stepname identifies a job step within a job. You must specify a stepname if later JCL statements refer to the step (see "Backward References" in the chapter "Coding Conventions"), or if the step is going to be part of a cataloged procedure or if you are going to perform step or checkpoint restart at or within the step. Otherwise, the stepname is optional.

Note: It is recommended that you give a stepname to each step in your job because stepnames are used in many operating system messages and it would be easier for you to find out what part of your job causes the messages if you supply a stepname. (If the step is unnamed, the part of the message where the stepname would appear is left blank.) Also, it will save recording time if you decide at a later date to use backward references or to turn your JCL statements into a cataloged procedure.

The stepname is coded in the name field of the EXEC statement. It can range from one to eight characters in length and can contain any alphanumeric or national (@,$,#) characters. However, the first character of the name must be a letter or national character and must begin in column 3. Each stepname in a job or procedure must be unique.

The following are examples of step names in several EXEC statements:

```
//STEP1      EXEC...
//CHECK      EXEC...
//A$9        EXEC...
//LINKEDIT   EXEC...
```

Processing Program Information

The main purpose of the EXEC statement is to identify either the program to be executed or the cataloged procedure to be used in the job step. The PGM parameter is used to identify the program to be executed. The PROC parameter is used to identify the cataloged procedure. You can also identify the cataloged procedure by simply coding its name as the first parameter in the operand field of the EXEC statement.

The PGM parameter and the PROC parameter (or procedure name) are mutually exclusive. One of them must be coded in the EXEC statement.

Your manager or supervisor should give you the names of the programs and cataloged procedures available in your installation. (Also, see Part II of this publication).

Some programs require special information for their processing. For example, you may have to let it know whether or not you want an output listing. You indicate your choices through the PARM parameter.

Each program that requires them has a fixed set of options you can specify through the PARM parameter. The options required by IBM programs are listed in the publication associated with the program. (Also, see Part II of this publication.)

PGM Parameter

You must use the PGM parameter to indicate which processing program is to to be used in this job step and where this program resides. Processing programs can reside in three types of libraries (partitioned data sets):

1. The System library (SYS1.LINKLIB)
2. Private libraries
3. Temporary libraries

The way you code the PGM parameter depends upon which type of library the program resides in. The PGM parameter must be the first parameter in the operand field.

1. The system library is a partitioned data set named SYS1.LINKLIB. All IBM-supplied processing programs and, probably, the most frequently used programs written by your installation reside in SYS1.LINKLIB. The format of the PGM parameter for specifying programs that reside in SYS1.LINKLIB is:

```
┌──────────────────────────────────────────────────────────────┐
|  PGM=progname                                                |
└──────────────────────────────────────────────────────────────┘
```

   Replace the term "progname" with the name or alias associated with the program. For example, the name of the FORTRAN H compiler is IEKAA00, therefore, to request it you should code:

```
//name EXEC PGM=IEKAA00
```

   You can also refer to the program through its alias. For example, the name of the loader is IEWLDRGO and its alias is LOADER. You can request the loader with either one of these two statements:

```
//name EXEC PGM=IEWLDRGO
//name EXEC PGM=LOADER
```

   Not all programs have an alias. For example, your installation may have several levels of the linkage editor, but only one of them can have the alias IEWL. Your manager or supervisor should give you a list of the names and aliases of the processing programs in your installation. (Also, see Part II of this publication.)

2. Private libraries are partitioned data sets that store programs not used sufficiently to warrant their inclusion in the system library (SYS1.LINKLIB). For example, a set of programs that prepare quarterly sales tax reports could be placed in a private library. The format of the PGM parameter is the same as for programs in SYS1.LINKLIB:

```
┌──────────────────────────────────────────────────────────────┐
|  PGM=progname                                                |
└──────────────────────────────────────────────────────────────┘
```

   Replace the term "progname" with the name or alias of the program. You indicate the fact that the program resides in a private library by inserting a special DD statement in your input deck. This DD statement defines the private library. You can name the DD statement either JOBLIB or STEPLIB. The JOBLIB DD statement specifies that the library is available to all steps in the job. The STEPLIB DD statement specifies that the library is available only to this job step. The use of these DD statements is explained in "Special DD Statements" in the section "The DD Statement."

3. 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 particulary useful for storing the program output (load module) of a linkage editor run until it is executed by a later job step. The program stored in a temporary library is assigned a name by the system, which is not predictable by the programmer. Therefore, you use the PGM parameter to identify it by location rather than by name. You do this using the backward-reference feature of JCL (see "Backward References" in the section "Coding Conventions".) The format of the PGM parameters for specifying programs that reside in temporary libraries is:

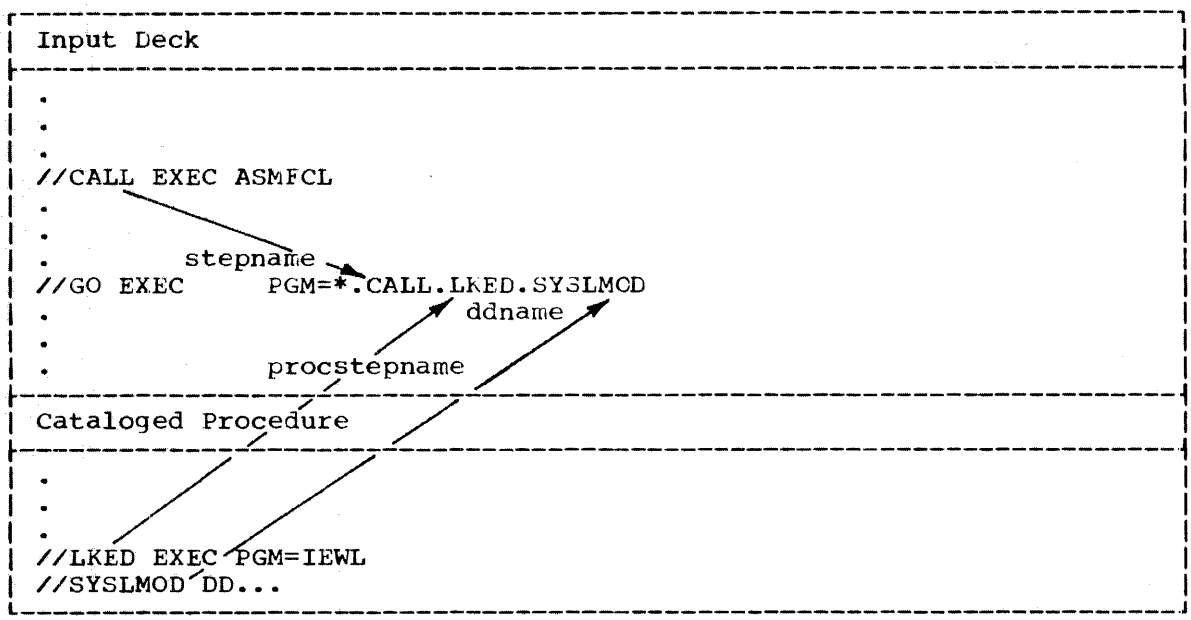

```
PGM=*.stepname.ddname
```

Replace the term "stepname" with the name of the EXEC statement of the job step (of the same job) where the temporary library is created. Replace the term "ddname" with the name of the DD statement that defined the library. The following example shows the EXEC statement for a link edit step, the DD statement in that step that defines the temporary library, and the EXEC statement that requests the execution of the program stored in the temporary library.

```
.
.
.
//LINK      EXEC  PGM=IEWL      This EXEC statement requests the
                                linkage editor
//SYSLMOD   DD    ...           This DD statement defines the
.                               temporary library
.
.
//GO        EXEC  PGM=*.LINK.SYSLMOD
```

When the temporary library is created in a cataloged procedure step, (of the same job) you may want to call it in a later job step outside the procedure. In order to call it, you must give both the name of the job step that invokes the procedure and the procedure step name, i.e., PGM=*.stepname.procstepname.ddname.

The first part of the following example shows an EXEC statement that calls a cataloged procedure named ASMFCL (ASMFCL is a cataloged procedure which assembles and link edits a source program), and the EXEC statement that requests the execution of the program link edited with the cataloged procedure. The second part of the example shows the EXEC statement of the procedure step in ASMFCL where the temporary library is created, and the DD statement that defines the library.



Programs residing in the system library or private libraries can also be executed by coding PGM=*.stepname.ddname, provided the named DD statement defines the program as a member of such a library.

Note: If your DD statements request space allocation and disposition processing you can satisfy these requirements prior to executing your program (see the next chapter). To do this, write PGM=IEFBR14 instead of specifying your program's name. This also allows you to check the accuracy of your JCL statements. (If you create a data set when using this program, the data sets status will be OLD when you execute your own program. Make sure you change the DD statement to indicate this fact.)

PROC or Procedure Name Parameter

Instead of executing a particular program a job step may use a cataloged or in-stream procedure. A cataloged or in-stream procedure can contain JCL statements for several steps, each of which executes a particular program. Cataloged procedures are members of the procedure library. (The IBM-supplied procedure library is named SYS1.PROCLIB; at your installation, there may be additional procedure libraries which would have different names.)

The format of the PROC parameter is:

```
PROC=procedure name
```

Replace the term "procedure name" with the member name of the cataloged procedure or the name on the PROC statement of the in-stream procedure. If you prefer, omit "PROC" and simply code the procedure name. For example, you can request a cataloged procedure named COBFLG with either one of the following EXEC statements:

```
//name EXEC PROC=COBFLG
//name EXEC COBFLG
```

The PROC parameter or the procedure name must be the first parameter in the operand field.

Note:

- Subsequent parameters in the operand field can be used to override EXEC statement parameters in the cataloged procedure.
- Such parameters reflect a reference to cataloged procedure steps in their keywords.
- For details on using and modifying cataloged or in-stream procedures, see "Part III: Cataloged and In-stream Procedures."

PARM Parameter

Some IBM-supplied processing programs allow you to select alternatives from a set of options. For example, two of the options the FORTRAN G compiler gives you are: (1) whether or not a listing of the object module is to be printed, and (2) the number of lines in each page of the listing. Your choices are indicated by certain values given to the PARM parameter. For example, to indicate that you want a listing and that each page is to have 65 lines write PARM=(LIST,'LINECNT=65').

Each IBM program that requires PARM information has a specific value for each of its options. For example, in the case of the FORTRAN G compiler, LIST specifies that a listing of the object module is to be printed. The PARM values are listed in the publication associated with the program and in Part II of this publication.

In many cases, default values can be selected for PARM values during system generation. That is, the system programmer will select one alternative (e.g., LIST) or assign a fixed value to another (e.g., LINECNT=40). The system will assume the default option unless you specify the other alternative or change the fixed value. For example, assume the defaults are LIST and LINECNT=40. If you want a listing of the object module that has 30 lines per page, write:

```
PARM='LINECNT=30' (LIST is the default)
```

If you do not want a listing, write:

```
PARM=NOLIST (LINECNT is ignored because the listing will not be
             printed.
```

If you want a listing that has 40 lines per page, omit PARM and both defaults are assumed.

Your manager or supervisor will tell you which default values were generated for the installation's operating system.

The installation's processing programs or your own program can accept PARM values if the program is written in assembler language or PL/I. The system will place the PARM values in an area of main storage available to the program. You can then obtain these values by following the instructions in "Acquiring the Information in the PARM field of the EXEC Statement" in the publication IBM System/360 Operating System: Supervisor and Data Management Services, GC28-6646, or in the publication IBM System/360 Operating System: PL/I (F) Programmer's Guide.

The format of the PARM parameter is:

```
PARM=value
```

Replace the term "value" with up to 100 characters of data. The following are general rules for coding the PARM parameter:

- If the value contains more than one expression separated by commas, the value must be enclosed in either apostrophes or parentheses. For example:

```
PARM=(DECK,LIST,NOMAP)
PARM='DECK,LIST,NOMAP'
```

- If any of the expressions contain special characters, you can either 1) enclose that expression in apostrophes (5-8 punch) and the value in parentheses, or 2) enclose the entire value in apostrophes. (The enclosing apostrophes and parentheses are not considered part of the information and do not count towards the maximum of 100 characters of data; commas within apostrophes are passed as part of the information.) For example:

```
PARM=(DECK,'NAME=FIRST',LIST) or PARM='DECK,NAME=FIRST,LIST'
PARM=(P1,167,'P*AA') or PARM='P1,167,P*AA'
```

- If the special character is an apostrophe, it must be shown as two consecutive apostrophes. For example show NAT'L as:

```
PARM='NAT"L'
```

  When two apostrophes are coded only one is passed to the processing program.

- If the special character is an ampersand and you are not defining a symbolic parameter, show the ampersand as two consecutive ampersands. For example, show ABC&D as:

```
PARM='ABC&&D'
```

  When two ampersands are coded, only one is passed to the processing program.

- If there is only one value and the value does not contain special characters, you need not enclose the value in parentheses or apostrophes. If the value contains special characters enclose the value in apostrophes. For example:

```
PARM=LIST
PARM='L.24'
PARM='NAT"L'
```

Since the value can consist of up to 100 characters, you may have to continue the value onto another card. If the value is to be continued, enclose the value in parentheses. Any value enclosed in apostrophes must be coded on one card. To continue the value, follow the continuation conventions as outlined in the topic "Continuing Control Statements" in the section "Coding Conventions". The continuation comma is considered part of the value field and counts towards the maximum of 100 characters of data. The following is an example of continuing the value onto another card.

```
//name EXEC...,PARM=(DECK,LIST,'LINECNT=80',
//         NOMAP)
```

When the job step uses a cataloged procedure, you can pass information to a step in the procedure by including as part of the keyword PARM, the procedure step name, i.e., PARM.procstepname. This specification overrides the PARM parameter in the named procedure step, if one is present. You can code as many parameters of this form as there are

steps in the cataloged procedure. For example, the following EXEC statement passes PARM information to two procedure steps of a cataloged procedure named PROCTWO. The names of the two procedure steps are STEP1 and STEP2.

```
//A1    EXEC PROC=PROCTWO,PARM.STEP1=(ONE,'TWO=B',
//           THREE),PARM.STEP2=(FOUR,FIVE,SIX,
//           'SEVEN=G')
```

To pass information to the first step in a cataloged procedure and nullify all other PARM parameters in the procedure, code the PARM keyword without a procedure step name. For example:

```
//A1    EXEC PROCTWO,PARM=(ONE,'TWO=B',EIGHT)
```


Installation Management Information

Some installations have job step accounting routines in addition to the regular job accounting routines. Job step accounting is particularly useful in cases where a different programmer is assigned to write each step of a job, or where the installation's management wants to know how much time is spent on different functions such as compilation or link editing.

You can specify job step accounting information instead of, or in addition to job accounting information. Job step accounting information is specified with the ACCT parameter in the EXEC statement. Job accounting information is specified with a positional parameter in the JOB statement (see the chapter "The JOB Statement").

ACCT Parameter

The ACCT parameter allows you to supply job step accounting information. It has the following format:

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

Replace the term "accounting information" with one or more subparameters separated by commas. Your manager or supervisor should tell you exactly how to code this parameter. The following are general rules for coding the accounting information:

- The total number of characters of accounting information, plus the commas that separate the subparameters, cannot exceed 142.
- If the list contains only one subparameter, you need not enclose it in parentheses. For example:

  ```
  ACCT=12345
  ```

- If any subparameter contains special characters (except hyphens), you can either 1) enclose the subparameter in apostrophes (5-8 punches) and the value in parentheses, or 2) enclose the entire value in apostrophes. The apostrophes are not considered part of the information. For example:

  ```
  ACCT=(12345,'T/24') or ACCT='12345,T/24'
  ACCT=(12345,T-24)
  ```

When the job step uses a cataloged procedure, you can supply accounting information pertaining to a single procedure step by including, as part of the keyword ACCT, the procedure step name, i.e., ACCT.procstepname. This specification overrides the ACCT parameter in the named procedure step, if one is present. You can code as many parameters of this form as there are steps in the cataloged procedure. For example, the following EXEC statement passes job step accounting information to two procedure steps of a cataloged procedure named PROC3. The name of the two procedure steps are ONE and TWO.

```
//BBB  EXEC PROC3,ACCT.ONE=(COMPILE,'T.JONES',                    X
//          '5/20/69'),ACCT.TWO=(LKED,'T.JONES','5/20/69')
```

To supply accounting information pertaining to all steps in a procedure, code the ACCT parameter without a procedure step name. This specification overrides all ACCT parameters in the procedure, if any are present. For example:

```
//BBB  EXEC PROC=PROC3,ACCT=('T.JONES','5/20/69')
```

Processing Options

There are two processing options available to the job step through EXEC statement parameters.

The COND parameter specifies conditions for executing or bypassing the job step according to the success or failure of previous steps in the job. For example, if one of the steps in your job is an error analysis program, you would only want to execute it if there were errors in the preceding steps.

The TIME parameter is only available in systems with MFT or MVT. It lets you set the maximum amount of time the step may use the CPU. This parameter lets you find out how long the job step uses the CPU and limits the CPU time wasted if the job step goes into a loop.

COND Parameter

Occasionally, some of the steps in a job may be unnecessary. For example, the last step of an inventory job might be to prepare reorder forms for depleted items. You would not want to execute this last step if one of the previous steps discovered that there were no missing items. The COND parameter of the EXEC statement lets you:

- Make as many as eight tests on return codes issued by preceding job steps or cataloged procedure steps, which 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.

The tests specified with the COND parameter of the EXEC statement are performed in addition to the tests specified with the COND parameter of the JOB statement. That is, the tests in the JOB statement are performed first, and if any are met the job is discontinued regardless of what you specify in the EXEC statements. Using the COND parameter in both the JOB and EXEC statement allows you to set some conditions that apply to all steps in the job and other conditions that apply only to particular job steps. Abnormal termination (ABEND) of a job step

normally causes subsequent steps to be bypassed and the job to be terminated. By means of the COND parameter, however, you can specify execution of a job step after one or more proceding job steps have abnormally terminated. For the COND parameter to be acted on, a job step must ABEND while the program has control. If a job step is abnormally terminated during scheduling, due to failures such as JCL errors or inability to allocate space, the remaining job steps are bypassed, no matter what you specified in any COND parameter.

The format of the COND parameter of the EXEC statement is:


```
                 ┌(code,operator)                          ┐
                 │(code,operator,stepname )                │
COND=(           │(code,operator,stepname.procstepname)    │ ,...)
                 │ EVEN                                    │
                 └ ONLY                                    ┘
```

You can write the term (code,operator[,stepname[.procstepname]]) up to eight times, or you can can write either EVEN or ONLY and the term (code,operator[,stepname[.procstepname]]) up to seven times.

Replace the term "code" with any decimal number from 0 through 4095.

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)

Replace the term "stepname" with the name of the preceding job step that issues the return code to be tested. If you do not code a "stepname" the test indicated is performed on all preceding steps. When the return code is issued by a cataloged procedure step, you may want to test it in a later job step outside of the procedure. In order to test it, you must give both the name of the job step that invokes the procedure and the procedure step name, i.e., COND=((code,operator, stepname.procstepname),...).

For example, if you write:

```
COND=((20,GT,STEP1),(60,EQ,STEP2))
```

it would read "Bypass this step if 20 is greater than the return code issued by STEP1, or if STEP2 issues a return code of 60."

If you write:

```
COND=((20,GT,STEP1),(60,EQ))
```

it would read "Bypass this step if 20 is greater than the return code issued by STEP1, or if any of the preceding steps issues a return code of 60".

If you want only one test made, omit the outer parentheses, for example:

```
COND=(10,LT) or COND=(15,NE,STEP5)
```

If you write:

```
COND=(7,LT,STEP4.LINK)
```

it would read "Bypass this step if 7 is less than the return code issued by a procedure step named LINK in the cataloged procedure called by an EXEC statement named STEP4".

The EVEN and ONLY subparameters are mutually exclusive. Whichever subparameter you select can be coded in combination with up to 7 return code tests, and can appear before, between, or after return code tests, for example:

```
COND=(EVEN,(4,GT,STEP3))
COND=((8,GE,STEP1),(16,GE),ONLY)
COND=((15,GT,STEP4),EVEN,(30,EQ,STEP7))
```

The EVEN subparameter causes the step to be executed even if 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 if 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.

For example, if you write:

```
COND=EVEN
```

it would read "Execute this step even if one or more of the preceding steps abnormally terminated during execution".

If you write:

```
COND=((10,LT,STEPA),(20,EQ),ONLY)
```

it would read, "Execute this step only if one of the preceding steps terminated abnormally but bypass it if 10 is less than the return code issued by STEPA or if any of the steps that terminated normally issued a return code of 20".

If you write:

```
COND=((10,LT,STEPA),(20,EQ),EVEN)
```

it would read, "Bypass this step if 10 is less than the return code issued by STEPA, or if any of the preceding steps issues a return code of 20, otherwise execute this step even if one of the preceding steps terminated abnormally".

If you omit the COND parameter, no return code tests are made and the step will be bypassed if any of the preceding job steps abnormally terminated.

Any tests specified with the COND parameter of the JOB statement take precedence over those specified with EXEC statements. For example, Figure 9 shows an input deck with nine steps and the return codes produced by those steps that were executed. The following tests are performed:

- Before STEP2 (STEP1 produced a return code of 6):
  1. Is 10 less than 6? No.
  2. Is the return code 2 or 4? No. Execute STEP2

- Before STEP3 (STEP2 produced a return code of 2):
  1. Is 10 less than 2 or 6? No.
  2. Did one or more of the preceding steps terminate abnormally? No. Bypass STEP3.


```
//MYJOB JOB   A.SMITH,COND=(10,LT)
//STEP1 EXEC  PGM=AAA
.
.
.
//STEP2 EXEC  PGM=BBB,COND=((2,EQ),(4,EQ))
.
.
.
//STEP3 EXEC  PGM=CCC,COND=ONLY
.
.
.
//STEP4 EXEC  PGM=DDD,COND=((5,GT,STEP1),(2,EQ))
.
.
.
//STEP5 EXEC  PGM=EEE
.
.
.
//STEP6 EXEC  PGM=FFF,COND=((8,GT,STEP5),EVEN)
.
.
.
//STEP7 EXEC  PGM=GGG,COND=(4,GT,STEP4)
.
.
.
//STEP8 EXEC  PGM=HHH
.
.
.
//STEP9 EXEC  PGM=III,COND=ONLY
```

Return Code
6
2
-
-
9
10
12
-
-

Figure 9. Using the COND Parameter

- Before STEP4:
  1. Is 10 less than 2 or 6? No.
  2. Is 5 greater than 6? No.
  3. Is one of the preceding return codes equal to 2? Yes. Bypass STEP4.

- Before STEP5:
  1. Is 10 less than 2 or 6? No. Execute STEP5.

- Before STEP6 (STEP5 produced a return code of 9):
  1. Is 10 less than 9, 2, or 6? No.
  2. Is 8 greater than 9? No.
  3. Did one of the preceding steps terminate abnormally? No. Execute STEP6.

- Before STEP7 (STEP6 produced a return code of 10):
  1. Is 10 less than 10, 9, 2, or 6? No.
  2. Is 4 greater than return code of STEP4? STEP4 was bypassed and did not produce a return code so this test is ignored. Execute STEP7.

- Before STEP8 (STEP7 produced a return code of 12):
  1. Is 10 less than 12, 10, 9, 2, or 6? Yes. Bypass STEP8 and STEP9.

Figure 10 is another example of the use of the COND parameter. This figure shows an input deck with nine steps and the return codes produced by those steps that were executed. The following tests are performed:

- Before TWO (ONE produced a return code of 4):
  1. Is 5 equal to 4? No.
  2. Is 7 less than 4? No. Execute TWO.

- Before THREE (TWO terminated abnormally):
  1. Is 5 equal to 4? No.
  2. Is EVEN or ONLY specified in THREE? Yes.
  3. Is 20 greater than 4? Yes. Bypass THREE.

- Before FOUR:
  1. Is 5 equal to 4? No.
  2. Is EVEN or ONLY specified in FOUR? Yes.
  3. Is any of the preceding return codes equal to 3? No. Execute FOUR.

- Before FIVE (FOUR produced a return code of 6):
  1. Is 5 equal to 6 or 4? No.
  2. Is 2 less than the return code of THREE? THREE was bypassed and did not produce a return code, so this test is ignored.
  3. Is EVEN or ONLY specified in FIVE? No. Bypass FIVE.

- Before SIX:
  1. Is 5 equal to 6 or 4? No.
  2. Is EVEN or ONLY specified in SIX? No. Bypass SIX.

- Before SEVEN:
  1. Is 5 equal to 6 or 4? No.
  2. Is EVEN or ONLY specified in SEVEN? Yes.
  3. Is 6 equal to the return code of FIVE? FIVE was bypassed and did not produce a return code, so this test is ignored. Execute SEVEN.

- Before EIGHT (SEVEN produced a return code of 5):
  1. Is 5 equal to 5, 6, or 4? Yes. Bypass EIGHT and NINE.

```
//ABC   JOB   12345,COND=(5,EQ)                    | Return Code
//ONE   EXEC  PGM=A                                |      4
.                                                  |
.                                                  |
.                                                  |
//TWO   EXEC  PGM=B,COND=(7,LT)                    |    ABEND
.                                                  |
.                                                  |
.                                                  |
//THREE EXEC  PGM=C,COND=((20,GT,ONE),EVEN)        |      -
.                                                  |
.                                                  |
.                                                  |
//FOUR  EXEC  PGM=D,COND=((3,EQ),ONLY)             |      6
.                                                  |
.                                                  |
.                                                  |
//FIVE  EXEC  PGM=E,COND=(2,LT,THREE)              |      -
.                                                  |
.                                                  |
.                                                  |
//SIX   EXEC  PGM=F                                |      -
.                                                  |
.                                                  |
.                                                  |
//SEVEN EXEC  PGM=G,COND=((6,EQ,FIVE),ONLY)        |      5
.                                                  |
.                                                  |
.                                                  |
//EIGHT EXEC  PGM=H,COND=EVEN                      |      -
.                                                  |
.                                                  |
.                                                  |
//NINE  EXEC  PGM=I                                |      -
```

Figure 10. Using the COND Parameter with ABEND

When the job step uses a cataloged procedure, you 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 step name, i.e., COND.procstepname. This specification overrides the COND parameter in the named procedure step, if one is present. You can code as many parameters of this form as there are steps in the cataloged procedure. For example, the following EXEC statement passed COND parameters to two procedure steps of a cataloged procedure named PROC4. The name of the two procedure steps are STEP4 and STEP6.

```
//TEST  EXEC  PROC=PROC4,COND.STEP4=((7,LT,STEP1),
//            (5,EQ),EVEN),COND.STEP6=((2,EQ),
//            (10,GT,STEP4))
```

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 step name. This specification replaces all COND parameters in the procedure, if any are present. For example,

```
//TEST  EXEC  PROC4,COND=((7,LT,STEP1),(5,EQ))
```

The stepname you specify in the condition, for example, STEP2 in (5,EQ,STEP2), can be the name of either a preceding procedure step in the cataloged procedure or of a preceding step in the job. Make sure you do not use the same stepnames for EXEC statements in your job as those used for procedure steps in any cataloged procedure used in that job. You can also test the return code produced by a procedure step of

another cataloged procedure used in the job. For example, if you want procedure step ST3 of the PROCB cataloged procedure bypassed when 10 is less than the return code produced by procedure step EDIT of a cataloged procedure called by the EXEC statement named TWO, code:

```
//THREE EXEC  PROCB,COND.ST3=(10,LT,TWO.EDIT)
```

You can establish similar tests for all steps in the procedure by coding the COND parameter without a procedure step name. For example, if you want the entire PROCB cataloged procedure bypassed when 10 is less than the return code produced by step EDIT of a cataloged procedure called by the EXEC statement named TWO, code:

```
//THREE EXEC  PROCB,COND=(10,LT,TWO.EDIT)
```

Notes:

- When a job step that contains the EVEN or ONLY subparameter references a data set that was to be created or cataloged in a preceding step, the data set (1) will not exist if the step creating it was bypassed, or (2) may be incomplete if the step creating it abnormally terminated.
- It is meaningless to specify the COND parameter for the first step of a job.

TIME Parameter (Systems With MVT Only)

The TIME parameter specifies the maximum amount of time a job step or cataloged procedure step may use the CPU. Two benefits of the TIME parameter are that it allows you to find out how long the step uses the CPU (CPU time appears on the output listing) and it helps limit the CPU time wasted by a step that goes into a loop. Normally, a job step that exceeds its time limit is terminated. However, if the System Management Facilities option is included in the CPU system and a user exit routine is provided, this routine can extend the time limit so that processing can continue. (The System Management Facilities option and user exit routines to be used with it are discussed in the publication IBM System/360 Operating System: System Programmer's Guide.)

The format of the TIME parameter is:

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

Replace the term "minutes" and "seconds" with the maximum number of minutes and seconds that the step can use the CPU. The number of minutes must be less than 1440 (24 hours); the number of seconds must be less than 60. That is, the maximum time you can specify is TIME=(1439,59).

If the job step may require use of the CPU for more than 1439 minutes, code TIME=1440 to eliminate the time limit.

If the CPU time limit is given in minutes only, you need not code the parentheses. For example, TIME=12. If the CPU time limit given is seconds only, you must code both the parentheses and a comma to indicate the absence of minutes. For example TIME=(,30).

If you omit the TIME parameter, the default CPU time limit for a job (as established in the cataloged procedure for the input reader) is assumed.

Coding TIME=1440 also lifts the restrictions on the amount of time a job step may remain in a wait state. If the System Management Facilities option is included in the system, the installation determines this time limit. In this case, a job step remaining in a wait state for more than the established time limit causes termination of the job unless a user-provided exit routine extends the wait-state time limit for that step. If the System Management Facilities option is not included, the system automatically provides a 30-minute time limit for wait states; that is, a job step remaining in a wait state for more than 30 consecutive minutes causes termination of the job. If the job step should be allowed to remain in a wait state for more than the established time limit, code TIME=1440 to eliminate the time limit.

When the job step uses a cataloged procedure, you can set a CPU time limit for a single procedure step by including, as part of the keyword TIME, the procedure step name, i.e., TIME.procstepname. This specification overrides the TIME parameter in the named procedure step, if one is present. You can code as many parameters of this form as there are steps in the cataloged procedure. For example, the following EXEC statement sets a time limit for two procedure steps of a cataloged procedure named PROC5. The name of the procedure steps are ABC and DEF.

```
//AAA EXEC  PROC5,TIME.ABC=20,TIME.DEF=(3,40)
```

To set a CPU time limit for an entire procedure, code the TIME parameter without a procedure step name. This specification overrides all TIME parameters in the procedure if any are present. For example:

```
//AAA EXEC  PROC5,TIME=20
```

Queuing Option (Systems With MVT Only)

The DPRTY parameter allows you to specify a dispatching priority for the job step. For further information on dispatching priority, refer to "Task Priority" in the publication IBM System/360 Operating System: Supervisor and Data Management Services, GC28-6646.

DPRTY Parameter (Systems With MVT Only)

The format of the DPRTY parameter is:

```
DPRTY=([value1][,value2])
```

Replace both "$value_1$" and "$value_2$" with a number from 0 through 15. The system uses the following formula to form the dispatching priority:

$(value_1 \times 12) + value_2$=dispatching priority

If you do not assign a number to "$value_1$", a default value of 0 is assumed. If you omit "$value_1$" you must code both the parentheses and a comma preceding "$value_2$" to indicate the absence of "$value_1$". For example, if you code:

```
DPRTY=(,5)
```

a value of DPRTY=(0,5) is assumed.

If you do not assign a number to "value$_2$", a default value of 11 is assumed. If you omit "value$_2$" you need not code the parentheses. For example, if you code:

```
DPRTY=7
```

a value of DPRTY=(7,11) is assumed.

If you omit the DPRTY parameter, the job step is assigned the priority you specified for the entire job either with the PRTY parameter of the JOB statement, or by default.

Whenever possible, you should avoid assigning a number of 15 to "value$_1$". This number is used for certain system functions.

When this step uses a cataloged procedure, you can assign a dispatching priority to a single procedure step by including, as part of the DPRTY parameter, the procedure step name, i.e., DPRTY.procstepname. This specification overrides the DPRTY parameter in the named procedure step, if one is present. You can code as many parameters of this form as there are steps in the cataloged procedure. For example, the following EXEC statement is used to establish a dispatching priority for two procedure steps of a cataloged procedure named PROC6. The names of the procedure steps are UP and DOWN.

```
//STEP9 EXEC PROC6,DPRTY.UP=(,8),DPRTY.DOWN=(4,6)
```

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 any are present. For example:

```
//STEP9 EXEC PROC=PROC6,DPRTY=(5,9)
```

Note: If your installation provides time-slicing facilities in a system with MVT, the DPRTY parameter can be used to make a job step part of a group of job steps to be time-sliced. At system generation, the priorities of the time-sliced groups are selected. If the number assigned to value 1 corresponds to a priority number selected for time-slicing and value 2 is either omitted or assigned a value of 11, then the job step will be time-sliced.

Main Storage Options (Systems With MVT Only)

MVT processes several jobs at the same time in different regions of main storage. The size of each region depends on the requirements of each job. Once the processing of that job is completed, the system uses that space for other jobs. (The free space can be subdivided into smaller regions, used as is for a region the same size, or combined with adjacent free space for a larger region.)

Usually you assign a region size through the REGION parameter of the JOB statement as described in the preceding chapter. In this case each step of the job will be executed in that region. You can, however, specify a different region size for each step in the job using the REGION parameter of the EXEC statement. This is desirable in cases where different steps need a greatly different region size. For example one step of your job may need 16K while another may need 128K. If you do not specify a region size for each step, there would be 112K unused while the first step is executed. Had you specified a region size for each step, MVT could use those 112K for executing other jobs while your first step is executed.

Regions are contiguous areas of main storage. However, you can define a two-part region if your operating system was generated with "main storage hierarchy support". Main storage hierarchy support provides for storage hierarchies 0 and 1. If 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 during 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.

The REGION parameter allows you to specify the region size to be used by your job step.

Notes:

- If you have specified a REGION parameter in the JOB statement, REGION parameters on the JOB'S EXEC statements are ignored.
- In systems without main storage hierarchy support, processor storage is referred to as main storage. In systems with main storage hierarchy support, main storage comprises both processor storage (hierarchy 0) and 2361 core storage (hierarchy 1).

Sometimes a job step will run out of space in its region. Normally, this means that the step is terminated. However, if the rollout/rollin option was generated for your MVT system, the system can obtain more space for your job step. This is accomplished by "rolling out" another job step that is currently being executed and allocating its region to your job step. After processing of your step is ended, the rolled-out step is "rolled in" and its processing continues.

The ROLL parameter lets you specify whether the job steps in your job can be rolled out and whether they can cause roll out of other job steps.

Note: ROLL parameters coded on EXEC statements are superseded by a ROLL parameter coded on the JOB statement.

REGION Parameter (Systems With MVT Only)

The REGION parameter allows you:

1. To request the maximum amount of main storage to be allocated to the job step. This figure must include the size of those components that are required by your program and are not resident in storage.
2. To request the amount of main storage to be allocated to the job step and in which storage hierarchy or hierarchies the space is to be allocated. This request should be made only if main storage hierarchy support is specified during system generation.

The storage requirements you must consider when specifying a region size are outlined in the publication IBM System/360 Operating System: Storage Estimates.

1. The format of the REGION parameter for systems without storage hierarchies is:

```
REGION=valueK
```

Replace the term "value" with the number of contiguous 1024-bytes areas you want allocated to the jobstep, for example, REGION=52K. This number can range from one to five digits but cannot exceed 16383. It should be specified as an even number. (If you specify an odd number, the system treats it as the next highest even number.)

If you omit the REGION parameter, the default value (as established in the input reader procedure) is assumed.

When the job step uses a cataloged procedure, you can request a region size for a single procedure step by including, as part of the REGION parameter, the procedure step name, i.e., REGION.procstepname. This specification overrides the REGION parameter in the named procedure step, if one is present. You can code as many parameters of this form as there are steps in the cataloged procedure. For example, the following EXEC statement is used to assign region sizes to two procedure steps of a cataloged procedure named PROC7. The names of the procedure steps are LITTLE and BIG.

```
//GO EXEC PROC7,REGION.LITTLE=20K,REGION.BIG=200K
```

To request a single region size for an entire cataloged procedure, code the REGION parameter without a procedure step name. This specification overrides all REGION parameters in the procedure, if any are present. For example,

```
//GO EXEC PROC=PROC7,REGION=200K
```

2. The format of the REGION parameter for systems <u>with</u> storage hierarchies is:

```
REGION=([value0K][,value1K])
```

Replace the term "$value_0$" with the number of contiguous 1024-byte areas you want allocated to the job step in hierarchy 0 (processor storage); replace the term "$value_1$" with the number of contiguous 1024-bytes areas to be allocated in hierarchy 1 (2361 storage), e.g., REGION=(60K,200K). Each value specified should be an even number. (If you specify an odd number, the system treats it as the next highest even number.)

The following rules apply to the hierarchy sizes:

- If 2361 Core Storage is <u>not</u> present but main storage hierarchy support was specified during 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. In this case, the sum of $value_0$ and $value_1$ cannot exceed 16383. For example, REGION=(10K,4K).

- If 2361 Core Storage is present, $value_0$, cannot exceed 16383, and $value_1$ cannot exceed 1024 if using a single Model 1, or 2048, if using a single Model 2.

- If main storage hierarchy support was not generated and 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, no hierarchy 0 segment is allocated.

In systems with main storage hierarchy support, you can omit either subparameter, thereby requesting storage in only one hierarchy. If you are requesting storage only in hierarchy 1, you must code a comma to indicate the absence of the first subparameter, for example, REGION=(,52K). If you are requesting storage only in hierarchy 0, you need not code the parentheses, for example, REGION=70K.

If you omit the REGION parameter, the default value (as established in the input reader procedure) is assumed. When the default region size is assumed, storage is always allocated in hierarchy 0.

When the job step uses a cataloged procedure, you can request a region size for a single procedure step by including, as part of the region parameter, the procedure step name i.e., REGION.procstepname. This specification overrides the REGION parameter in the named procedure step, if one is present. You can code as many parameters of this form as there are steps in the cataloged procedure. For example, the following EXEC statement is used to assign region sizes to two procedure steps of a cataloged procedure named PROC8. The names of the procedure steps are FIRST and SECOND.


```
//HIER  EXEC  PROC8,REGION.FIRST=(40K,20K),
//            REGION.SECOND=(,32K)
```

To request a single region size for an entire cataloged procedure, code the REGION parameter without a procedure step name. This specification overrides all REGION parameters in the procedure, if any are present. For example:

```
//HIER  EXEC  PROC=PROC8,REGION=(40K,32K)
```

Note: If you have specifed a REGION parameter on the JOB statement, REGION parameters on the job's EXEC statements are ignored.

ROLL Parameter (Systems With MVT Only)

The function of rollout/rollin is to allocate additional main storage to a job step whose own region contains no more available space. In order to allocate this additional main storage to a job step, another job step may have to be rolled out, i.e., temporarily transferred to secondary storage. The ROLL parameter allows you to indicate whether or not your job step can be rolled out and whether or not it can cause rollout of other job steps.

The format of the ROLL parameter is:

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

The "x" declares the job step's ability to be rolled out. If "x" is YES, the job step can be rolled out; if "x" is NO, the job step cannot be rolled out. The "y" declares the job step's ability to cause rollout of another job step. If "y" is YES, the job step can cause rollout; if "y" is NO, the job step cannot cause rollout. (YES must be specified if you want additional main storage allocated to the job step if it is required.) If you include the ROLL parameter, both the "x" and "y" subparameters must be coded.

For example, ROLL=(NO,YES) indicates that the job step cannot be rolled out but that it can cause roll out.

In systems with main storage hierarchy support, you can omit either subparameter, thereby requesting storage in only one hierarchy. If you are requesting storage only in hierarchy 1, you must code a comma to indicate the absence of the first subparameter, for example, REGION=(,52K). If you are requesting storage only in hierarchy 0, you need not code the parentheses, for example, REGION=70K.

If you omit the REGION parameter, the default value (as established in the input reader procedure) is assumed. When the default region size is assumed, storage is always allocated in hierarchy 0.

When the job step uses a cataloged procedure, you can request a region size for a single procedure step by including, as part of the region parameter, the procedure step name i.e., REGION.procstepname. This specification overrides the REGION parameter in the named procedure step, if one is present. You can code as many parameters of this form as there are steps in the cataloged procedure. For example, the following EXEC statement is used to assign region sizes to two procedure steps of a cataloged procedure named PROC8. The names of the procedure steps are FIRST and SECOND.


```
//HIER  EXEC  PROC8,REGION.FIRST=(40K,20K),
//            REGION.SECOND=(,32K)
```

To request a single region size for an entire cataloged procedure, code the REGION parameter without a procedure step name. This specification overrides all REGION parameters in the procedure, if any are present. For example:

```
//HIER  EXEC  PROC=PROC8,REGION=(40K,32K)
```

Note: If you have specifed a REGION parameter on the JOB statement, REGION parameters on the job's EXEC statements are ignored.

ROLL Parameter (Systems With MVT Only)

The function of rollout/rollin is to allocate additional main storage to a job step whose own region contains no more available space. In order to allocate this additional main storage to a job step, another job step may have to be rolled out, i.e., temporarily transferred to secondary storage. The ROLL parameter allows you to indicate whether or not your job step can be rolled out and whether or not it can cause rollout of other job steps.

The format of the ROLL parameter is:

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

The "x" declares the job step's ability to be rolled out. If "x" is YES, the job step can be rolled out; if "x" is NO, the job step cannot be rolled out. The "y" declares the job step's ability to cause rollout of another job step. If "y" is YES, the job step can cause rollout; if "y" is NO, the job step cannot cause rollout. (YES must be specified if you want additional main storage allocated to the job step if it is required.) If you include the ROLL parameter, both the "x" and "y" subparameters must be coded.

For example, ROLL=(NO,YES) indicates that the job step cannot be rolled out but that it can cause roll out.

When the job step uses a cataloged procedure, you can indicate 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, include, as part of the ROLL parameter, the procedure step name, i.e., ROLL.procstepname. This specification overrides the ROLL parameter in the named procedure step, if one is present. You can code as many parameters of this form as there are steps in the cataloged procedure. For example, the following EXEC statement is used to assign roll conditions to two steps of a cataloged procedure named PROC9. The names of the procedure steps are HOT and COLD.

```
//ROLLER   EXEC   PROC9,ROLL.HOT=(NO,YES),
//                ROLL.COLD=(YES,NO)
```

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, code the ROLL parameter without a procedure step name. This specification overrides all ROLL parameters in the procedure, if any are present. For example:

```
//ROLLER   EXEC   PROC9,ROLL=(YES,NO)
```

If this parameter is omitted, the default specified in the reader procedure is used. In the IBM-supplied reader procedures, the default specified is ROLL=(YES,NO).

Notes:

- Roll parameters coded on EXEC statements are superseded by a ROLL parameter coded on the JOB Statement.
- Teleprocessing jobs that use the autopoll option should not have the ability to be rolled out. A rolled out job using this option cannot be restarted properly. Therefore, you should always code ROLL=(NO,YES) or ROLL=(NO,NO) for this kind of job.

Checkpoint/Restart

When a job step abnormally terminates (ABEND), you may have to resubmit the job for execution. This means lost computer time and a delay in obtaining the desired results. To reduce these effects, you can use the restart facilities of the operating system provided your program is written in the assembler language, COBOL, or PL/1.

If a job step abnormally terminates or if a system failure occurs in a system with MFT or MVT, the restart facilities allow you to request that the job step be restarted automatically either at the beginning of the step (step restart) or within the step (checkpoint restart). Automatic restarts are specified with the RD parameter of the EXEC statement.

For detailed information on the checkpoint/restart facilities, using the assembler language refer to the publication IBM System/360 Operating System: Supervisor and Data Management Services; using COBOL, to IBM System/360 Operating System: COBOL (E) Programmer's Guide, or to IBM System/360 Operating System: COBOL (F) Programmer's Guide, or to IBM System/360 Operating System: American National Standard COBOL Programmer's Guide; using PL/1, to IBM System/360 Operating System: PL/1 (F) Programmer's Guide; and also to "Appendix C: Using the Restart Facilities" of this publication and to the publication IBM System/360 Operating System: Advanced Checkpoint/Restart Planning Guide, GC28-6708.

RD Parameter

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 either case, automatic restart can occur only if:

1. you use the RD parameter to request restart,
2. the return code returned during abnormal termination indicates that the step is eligible for restart, and
3. the operator authorizes restart.

In order for checkpoint restart to occur, the CHKPT macro instruction (assembler), or the RERUN clause (COBOL), or the CALL IHECKPT statement (PL/1) must have been executed in your processing program prior to abnormal termination. Through the RD (restart definition) parameter, you can specify that step restart can occur or that the action of the CHKPT macro instruction (assembler), or the RERUN clause (COBOL), or the CALL IHECKPT statement (PL/1) is to be suppressed.

The format of the RD parameter is:

```
RD=request
```

Replace the work "request" with:

R	to permit automatic step restart
NC	to suppress the action of the CHKPT macro instruction, or RERUN clause, or CALL IHECKPT statement, and not permit automatic restart
NR	to request that the CHKPT macro instruction, or RERUN clause, or CALL IHECKPT statement 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, or RERUN clause, or CALL IHECKPT statement.

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

RD=R: If the processing programs used by this job step do not include any CHKPT macro instructions, (assembler), or RERUN clause (COBOL), or CALL IHECKPT statements (PL/1), 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, or RERUN clauses, or CALL IHECKPT statements step restart can occur if a step abnormally terminates before execution of a CHKPT macro instruction or RERUN clause, or CALL IHECKPT statement; thereafter, checkpoint restart can occur. If you cancel the effects of the CHKPT macro instructions or RERUN clauses, or CALL IHECKPT statements before a checkpoint restart is performed, the request for automatic step restart is again in effect.

RD=NC or RD=RNC: RD=NC or RD=RNC should be specified when you want to suppress the action of all CHKPT macro instructions (assembler), or RERUN clauses (COBOL), or CALL IHECKPT statements (PL/1) included in the programs used by this step. When RD=NC is specified, neither step restart not checkpoint restart can occur. When RD=RNC is specified, step restart can occur. RD=NC has no effect on processing if CHKPT macro instructions, or RERUN clauses, or CALL IHECKPT statements are not included in the program.

RD=NR: RD=NR permits a CHKPT macro instruction (assembler), or RERUN clause (COBOL), or CALL IHECKPT statement (PL/1) to establish a checkpoint, but does not permit automatic step or checkpoint restarts. Instead, at a later time, you can resubmit the job and begin execution at a specific checkpoint. (The RESTART parameter of the JOB statement is used for resubmitting a job for restart. See "Checkpoint/restart" in the chapter "The JOB Statement".) This parameter has no effect on processing if CHKPT macro instructions, RERUN clauses, or CALL IHECKPT statements are not included in the program.

If automatic restart is requested for an abnormally terminated step and restart is authorized by the operator, special disposition processing is performed. If automatic step restart is to occur, 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 with a status of NEW in the restart step are deleted. If automatic checkpoint restart is to occur, all data sets currently in use by the job are kept. (For further information on data set dispositions refer to the section "The DD Statement" and to "Appendix C: Using the Restart Facilities."

If you omit the RD parameter and no CHKPT macro instructions, or RERUN clauses, or CALL IHECKPT statements are executed, automatic restart cannot occur. If you omit the RD parameter but one or more CHKPT macro instructions or RERUN clauses, or CALL IHECKPT statements are executed, automatic checkpoint restart can occur.

When this job step uses a cataloged procedure, you can make a restart request for a single procedure step by including, as part of the RD parameter, the procedure step name, i.e., RD.procstepname. This specification overrides the RD parameter in the named procedure step, if one is present. You can code as many parameters of this form as there are steps in the cataloged procedure.

For example, the following EXEC statement requests restart for two steps of a cataloged procedure named PROC10. The names of the procedure step are FIVE and NINE.

```
//ASTEP   EXEC  PROC10,RD.FIVE=R,RD.NINE=NC
```

To specify a restart request for an entire cataloged procedure, code the RD parameter without a procedure step name. This specification overrides all RD parameters in the procedure, if any are present.

For example:

```
//ASTEP   EXEC  PROC=PROC10,RD=R
```

Notes:

- If you specify the RD parameter on the JOB statement, RD parameters on the job's EXEC statements are ignored.
- When using a system with MVT, MSGLEVEL=(1,0) or MSGLEVEL=(1,1) must be coded on the JOB statement. If you omit this parameter, restart is not performed.
- If restart is requested for this step, assign each step a unique step name. (Upon restart, the system searches the name field for the corresponding name of the abnormally terminated step. If two steps have the same name, restart is attempted at the first step encountered with the corresponding name regardless of whether or not this is the step that abnormally terminated.) Step names are assigned in the name field of the EXEC statement for each step.
- If restart occurs, the CPU time limit assigned to the step with the TIME parameter is restored to its original value.

The DD Statement

There must be a DD statement for each data set used in your job step. DD statements follow the EXEC statement that marks the beginning of the job step. You can include a maximum of 255 DD statements in each job step.

The main functions of the DD statements are to describe the characteristics of data sets and to indicate their location. These functions allow you a great deal of freedom in writing your program. For example, if you are writing a program to process paid bills, you do not have to indicate in your program the size of the input records, or the type of device in which they are located. You can postpone these definitions until you run the program when you must write the DD statement for the input data sets. You can debug your program, and then run it several times with different DD statements for the input record data set. In that way you can determine which record size is most efficiently processed, whether the input should come from a card reader or a magnetic tape unit. All your program needs to know to refer to the data set is the name of the DD statement (ddname) that describes the data set. Each time you execute the program you can use the DD statement to describe a different data set as long as the ddname remains constant. (Refer to "Section 3: Data Management" in the publication IBM System/360 Operating System: Concepts and Facilities for a summary of the data management concepts you must know before reading this chapter on the DD statement.)

You can, however, define data set characteristics within your program so that you will not have to specify those characteristics that remain constant each time you use a data set. The number and type of data set characteristics you can specify in your program rather than in the DD statement depends on the language you are using for writing your program. For example, the assembler language allows you to specify more data set characteristics in your program than the FORTRAN language. However, regardless of the facilities of the language you are using, you should only specify those requirements essential to processing in your program and leave the rest for the DD statement. This gives you more flexibility in writing the program and places fewer restrictions on any future changes you may have to make to the program.

All job steps in your job (except those steps that use a cataloged procedure) require DD statements because every program must have either an input data set, or an output data set, and, in many cases, work data sets in order to operate. The names of the DD statements required for IBM-supplied programs, such as compilers and utilities, are predefined and you must code their parameters according to the rules stated in the publications associated with the programs. (A summary of JCL statements for compilers, linkage editors, and loader appears in Part II of this publication.)

Only you can determine the DD statements required for your own program. In general, you will need one DD statement for each data set defined with:

- A data set number if you are using ALGOL.
- A DCB macro instruction if you are using the assembler.
- An FD entry if you are using COBOL.
- A data set reference number if you are using FORTRAN.
- The TITLE option or file name if you are using PL/1.

You will need more than one DD statement for each data set in the two following cases:

1. Definition and retrieval of index sequential (ISAM) data sets may require up to three DD statements.

2. Several input data sets, each defined by a different DD statement, may be read as if they were a single data set through the technique of concatenation.

In both cases, only the first DD statement is given a ddname.

If the job step uses a cataloged procedure, you can use a DD statement either to override parameters in a DD statement in the procedure, or simply, to add a new DD statement to the procedure. In both cases, the modification remains in effect only for the duration of the job step, it does not change the procedure permanently.

A DD statement must contain the term DD in its operation field. Although all parameters in the DD statement's operand field are optional, a blank operand field is invalid, except when you are overriding DD statements defining concatenated data sets in a cataloged procedure. (See "Overriding, Adding and Nullifying Parameters on a DD Statement" in Part III of this publication.

The parameters in the operand field allow you to specify five types of optional information:

1. Data set information.
2. Location of the data set.
3. Size of the data set.
4. Data attributes.
5. Special processing options.

Table 5 shows the parameters used for each type of information and their functions.

As you can see, not all parameters are needed to define a data set. In fact, some combinations of parameters cannot be used in the same DD statement.

The valid combination of DD statement parameters allow you to perform the following functions:

1. Create a data set:
   a. on unit record devices (card punch or printer)
   b. on system output devices
   c. on magnetic tape
   d. on direct access devices

2. Retrieve an existing data set:
   a. from unit record devices (card reader or paper tape reader)
   b. from input stream
   c. passed data set (from magnetic tape on direct access)
   d. cataloged data set (from magnetic tape or direct access)
   e. kept data set (from magnetic tape on direct access)

3. Extend an existing data set:
   a. passed data set (from magnetic tape or direct access)
   b. cataloged data set (from magnetic tape or direct access)
   c. kept data set (from magnetic tape or direct access)

4. Define special data sets:
   a. private libraries
   b. dump data sets
   c. checkpoint data set

5. Postpone definition of a data set

This chapter describes the ddname and the parameters needed for each of the above functions. The sections that describe each of the functions are self-contained. For example, the section on how to create a data set on magnetic tape contains full descriptions of all the parameters (and their appropriate subparameters) needed for this function. Self-contained sections eliminate ambiguity in the use of parameters, but lead to some repetition in the text. For this reason, a summary of the DD statement is given in Appendix D. This appendix shows the correct formats of the parameters for each of the functions without textual descriptions.

Note: Some devices, such as graphics devices, are not discussed in this manual. For information on how to specify IBM devices not described in this manual, refer to IBM System/360 Operating System: Job Control Language Reference.

Table 5. DD Statement Parameters (Part 1 of 2)

Type	Parameter	Function	P/K	Comments
Data Set Information	DSNAME (or DSN)	Names the data set. If omitted, the data set is considered to be temporary.	K	
	DISP	Specifies the current status of the data set and whether or not it will be retained after the job step.	K	
Location of the Data Set	UNIT	Specifies the type of device to be used for the data set.	K	
	VOLUME (or VOL)	Describes the volume on which the data set resides.	K	For magnetic tape or direct access devices only.
	LABEL	Describes the volume label.	K	For magnetic tape or direct access devices only.
	SYSOUT	Routes a data set through the output stream.	K	
	*	Indicates that the data set is in the input stream.	P	Only one of these parameters can be used in a DD statement.
	DATA	Indicates that the data set is in the input stream and that it contains JCL statements to be treated as data.	P	
Size of the Data Set	SPACE	Requests space on direct access devices.	K	Only one of these parameters can be used in a DD statement.
	SPLIT	Used to split cylinders between data sets	K	
	SUBALLOC	Specifies that the data set shares direct access space with other data sets.	K	
Data Attributes	DCB	Specifies the data attributes not specified in your processing program.	K	

Table 5. DD Statement Parameters (Part 2 of 2)

Type	Parameter	Function	P/K	Comments
Special Processing Options	SEP	Requests that the data set be assigned a separate channel from the ones assigned to earlier data sets.	K	Only one of these parameters can be used in a DD statement.
	AFF	Requests the same separation requirements as a previous DD statement that used the SEP parameter.	K	
	DUMMY	Indicates that I/O operations are to be bypassed for this data set. Commonly used while debugging a program.	P	
	DDNAME	Postpones the definition of a data set. Useful in cataloged procedures.	K	In systems with PCP, all other parameters must be omitted. In systems with MFT or MVT, the DCB subparameters BLKSIZE and BUFNO may be coded.
	UCS	Requests a particular chain in a 1403 printer with the universal character set feature.	K	
	OUTLIM	Specifies a limit for the number of logical records included in the output data set being routed through the output stream.	K	The OUTLIM parameter is ignored unless SYSOUT is coded in the operand field of the same DD statement. For systems with MFT or MVT only.

Legend: P = Positional Parameter
K = Keyword Parameter

Naming the DD Statement

The ddname identifies the DD statement so that subsequent JCL statements and the processing program can refer to it. The following rules apply to the ddname:

1. Only the first DD statement of a group of DD statements that define an indexed sequential (ISAM) data set must have a ddname. The name field of the other DD statements in the group must be left blank.
2. Only the first DD statement of a group of DD statements that define concatenated data sets must have a ddname. The name field of the other DD statements in the group must be left blank.
3. All other DD statements must have ddnames.

Each ddname within a jobstep should be unique. If duplicate ddnames exist, all I/O references are directed to the first such DD statement in the job step.

The ddname is coded in the name field of the DD statement. It can range from one to eight characters in length and can contain any alphanumeric or national (@,$,#) characters. However, the first character of the name must be a letter or national character and must be in column 3. The following are examples of ddnames in several DD statements.

```
//DD1         DD...
//SYSIN       DD...
//FILE        DD...
//AREA25      DD...
//#103        DD...
//FT31F001    DD...
```

The ddnames used by IBM processing programs are predetermined and you must code them as shown in the publications associated with the programs, and in Part II of this publication.

If you code your program in ALGOL or FORTRAN you must use certain ddnames for your data sets. The format of the ddnames is:

```
ALGLDDnn   (ALGOL)
FTnnF001   (FORTRAN)
```

where nn is the data set reference number. For further information refer to the programmer's guide for the language you are using.

The ddnames JOBLIB,STEPLIB,SYSABEND,SYSUDUMP, and SYSCHK are reserved for special system facilities. These ddnames are described in "Special DD statements" later on in this section.

If your job step uses a cataloged procedure, the ddname must be qualified by the procedure step name, i.e., procstepname.ddname. The ddname can identify either a DD statement in the procedure whose parameters you want to override, or a new DD statement you want to add to the procedure. (In both cases, the modification remains in effect only for the duration of the job step; it does not change the procedure permanently.) For example, suppose you are using a cataloged procedure named PROCTWO. The PROCTWO procedure has two procedure steps named FIRST and SECOND. You want to change a DD statement named OUTPUT in FIRST and add a DD statement named DATA to SECOND. Therefore you would code:

```
//name EXEC PROC=PROCTWO
//FIRST.OUTPUT DD...
//SECOND.DATA DD...
```

Table 6. Parameters for Creating a Data Set

Device	Parameter Type	Parameter	Comments
Unit Record Devices	Location of the Data Set	UNIT	Required.
	Data Attributes	DCB	Optional.
	Special Processing Options	UCS	Optional (for 1403 printer with the universal character set feature).
		DUMMY	Optional.
System Output Devices	Location of the Data Set	SYSOUT	Required. Specifies the output class.
		UNIT	Systems with MFT or MVT only. Optional.
	Size of the Data Set	SPACE	Systems with MFT or MVT only. Optional.
	Data Attributes	DCB	Optional.
	Special Processing Option	OUTLIM	Optional. Meaningful only for Systems with MFT or MVT that have the Systems Management Facilities Option.
Magnetic Tape	Data Information	DSNAME (or DSN)	Required if the data set is to be cataloged or used by a later job.
		DISP	Required if the data set is to be cataloged, used by a later step in this job, or used by another job.
	Location of the Data Set	UNIT	Required unless you request (with the VOLUME parameter) the same volume used for an earlier data set in your job. earlier data set in your job.
		VOLUME (or VOL)	Required if you want a specific volume. If you do not use this parameter you get a scratch tape.
		LABEL	Required if you do not want to use standard labels for the data set.
	Data Attributes	DCB	Optional.
	Special Processing Options	SEP	Either parameter can be used.
		AFF	
		DUMMY	Optional.
Direct Access Devices	Data Set Information	DSNAME (or DSN)	Required if the data set is to be cataloged or used by a later job.
		DISP	Required if the data set is to be cataloged, used by a later step in this job, or used by another job.
	Location of the Data Set	UNIT	Required unless you request (with the VOLUME parameter) the same volume used for an earlier data set in your job, or unless you use the SPLIT or SUBALLOC parameters to allocate space to this data set.
		VOLUME (or VOL)	Required if you want a specific volume or multiple volumes. If you do not use this parameter your data set will be allocated on any suitable volume.
		LABEL	Required if you want the data set to have both standard and user labels.
	Size of the Data Set	SPACE	One of these parameters is required. SPLIT can only be used for BSAM or QSAM data sets SPACE must be used for ISAM data sets.
		SPLIT	
		SUBALLOC	
	Data Attributes	DCB	Optional. Required for BDAM and ISAM data sets.
	Special Processing Options	SEP	Either parameter can be used.
		AFF	
		DUMMY	Optional.

Creating a new Data Set

A new data set is an output data set. Before you define it with a DD statement you must decide on what type of device you want it placed. You can choose from:

- Unit record devices (card punch or printer).
- Magnetic tape.
- Direct access devices.
- System output device.

You will need a different set of parameters depending on the type of device you choose (see Table 6). The following topics describe the use of those parameters for each type of device. Please fold out Table 6 while reading this chapter.

Unit Record Devices

You can create an output data set on either a card punch or a printer. As shown in Table 6, only the UNIT parameter is required to indicate the device you want for the data set. The DSNAME and DISP parameters are not used because data sets on unit record devices are always temporary and cannot be retrieved by another DD statement in your job.

This section is summarized in Table 63 of Appendix D.

Location of the Data Set

The location of the data set is given by the UNIT parameter.

UNIT: The format of the UNIT parameter is:

```
UNIT={unit address}
     {device type }
     {group name  }
```

where:

unit address
: is the actual machine address of the device. For example, UNIT=00F. You should not specify the address unless you are sure you want that specific device.

device type
: corresponds to the model number of the I/O device. Coding a device type provides you with a certain degree of device independence in that your data set may be placed in any of a number of devices of the same type. For example, if you code

  UNIT=1442

  your data set will be punched in any 1442 Card Read Punch in the system. The following device types can be specified:

Device Type	Description
1052	1052 Printer-keyboard
1403	1403 Printer or 1404 Printer (continuous form only)
1442	1442 Card Read Punch
1443	Any 1443 Printer
2520	2520 Card Read Punch
2540-2	2540 Card Read Punch (punch feed)

group name
is the name of a collection of devices selected by your installation during system generation. For example, your installation might select the name PUNCH for all card punch units in the configuration. If you do not care which card punch is used for your data set, you would then code

```
UNIT=PUNCH
```

Your manager or supervisor should tell you which group names were generated for your installation.

Data Attributes

The DCB parameter allows you to specify attributes for your data set when your program is to be executed rather than when it is compiled. Any applicable attributes not specified in your program must be specified with the DCB parameter. However, in most cases, your compiler will provide a default value for an attribute if you do not specify it in the program or in the DD statement. Other attributes are always given a fixed value by the compiler and you do not have to specify them at all. For example, you can select a buffering technique with the assembler, but all other IBM compilers select one for you when it is needed.

DCB: You can use the DCB parameter to directly specify the attributes of your data set or to copy those attributes specified in a DD statement for another data set.

The format of the DCB parameter for specifying the attribute is:

```
DCB=(list of attributes)
```

The attributes in the list are coded in the form of keyword subparameters separated by commas; for example,

```
DCB=(BLKSIZE=300,LRECL=100)
```

The valid subparameters that can be used with each compiler for the card punch and printer are shown in Table 7 and 8, respectively. Underscored items are those default values selected if you omit the subparameter. Default values are not shown where the attribute can either be specified in your program, or in the DD statement. If values for a given subparameter are not shown, it is either specified in your source program or given a fixed value by the compiler. A glossary of DCB subparameters is given in Appendix B. Code only those parameters that apply to your compiler as shown in Tables 7 and 8.

Table 7. DCB Subparameters for Card Punch

DCB Subparameter	ALGOL	Assembler	COBOL E	COBOL F	ANS COBOL [3]	FORTRAN E	FORTRAN G & H	PL/I F
BFALN=		F or D [1,2]						
BFTEK=		S or E (QSAM only) [1,2]						
BLKSIZE=	number of bytes	number of bytes [1]		number of bytes [1]	number of bytes [1]	number of bytes	number of bytes	number of bytes [1]
BUFL=		number of bytes [1]						
BUFNO=		number of buffers [1]	number of buffers [1]	number of buffers [1]	number of buffers [1]	1 or 2	1 or 2	number of buffers [1]
EROPT=		ABE [1]	ABE	ABE	ABE			
HIARCHY=		0 or 1 [1]						
LRECL=	number of bytes	number of bytes [1]				number of bytes	number of bytes	number of bytes [1]
MODE=		C or E [1]	C or E	C or E	C or E	C or E	C or E	C or E
NCP=		number of channel programs (BSAM only) [1]						
OPTCD=		[C] [1]	[C]	[C]	[C]	[C]	[C]	[C]
RECFM=	F[B][A]	U[A/M] or V[B/S/BS][A/M] or F[B/S/BS][A/M] [1]				U[A] or F[B][A/M]	U[A/M] or F[B][A/M]	U[A/M] or V[B/S/BS][A/M] or F[B/S/BS][A/M]
STACK=		1 or 2 [1]	1 or 2	1 or 2	1 or 2	1 or 2	1 or 2	1 or 2

[1] This function can be specified in your program rather than in the DD statement.

[2] For QSAM, you must specify both BFALN and BFTEK on the DD statement, or omit both.

[3] American National Standard COBOL.

Table 8. DCB Subparameters for Printer

DCB Subparameter	ALGOL	Assembler	COBOL E	COBOL F	ANS COBOL [4]	FORTRAN E	FORTRAN G & H	PL/I F
BFALN=		F or D [1,2]						
BFTEK=		S or E (QSAM only) [1,2]						
BLKSIZE=	number of bytes	number of bytes [1]		number of bytes [1]	number of bytes [1]	number of bytes	number of bytes	number of bytes [1]
BUFL=		number of bytes [1]						
BUFNO=		number of buffers [1]	number of buffers [1]	number of buffers [1]	number of buffers [1]	1 or 2	1 or 2	number of buffers [1]
EROPT=		ACC or ABE [1]	ACC or ABE	ACC or ABE	ACC or ABE			
HIARCHY=		0 or 1 [1]						
LRECL=	number of bytes	number of bytes [1]				number of bytes	number of bytes	number of bytes [1]
NCP=		number of channel programs (BSAM only) [1]						number of channel programs (BSAM only) [1]
OPTCD=		[C] [U] [1]	[C]	[C]	[C]	[C]	[C]	[C] [U]
PRTSP=		0,1,2, or 3 [1,3]	0,1,2, or 3 [3]	0,1,2, or 3 [3]	0,1,2, or 3 [3]	0,1,2, or 3 [3]	0,1,2, or 3 [3]	0,1,2, or 3 [3]
RECFM=	F[B/BS][A]	U[A/M] or V[B/S/BS][A/M] or F[B/S/BS][A/M] [1]				Formatted: U[A] or F[B][A/M] Unformatted: VS[B][A/M]	Formatted: U[A/M] or F[B][A/M] Unformatted: VS[B][A/M]	U[A/M] or V[B/S/BS][A/M] or F[B/S/BS][A/M] [1]

[1] This function can be specified in your program rather than in the DD statement.

[2] For QSAM, you must specify both BFALN and BFTEK on the DD statement, or omit both.

[3] Do not use if A or M is specified in the RECFM.

[4] American National Standard COBOL.

The format of the DCB parameter for copying the DCB parameter of a previous DD statement in your job is:

```
DCB=({*.ddname                          }[,list of attributes])
     {*.stepname.ddname                 }
     {*.stepname.procstepname.ddname    }
```

Replace "ddname" with the name of the DD statement whose DCB parameter you want to copy. Replace "stepname" with the name of the EXEC statement of the step that contains that DD statement. If the DD statement you want to copy is contained in the same job step, omit the stepname, i.e., DCB=*.ddname. The following example shows how to code the DCB parameter to copy the DCB parameter of a DD statement in a previous step:

```
//STEP2 EXEC...
//DD1 DD...,DCB=(BLKSIZE=1600,LRECL=80)
.
.
.
//STEP4 EXEC...
//COPY DD...,DCB=*.STEP2.DD1
.
.
.
```

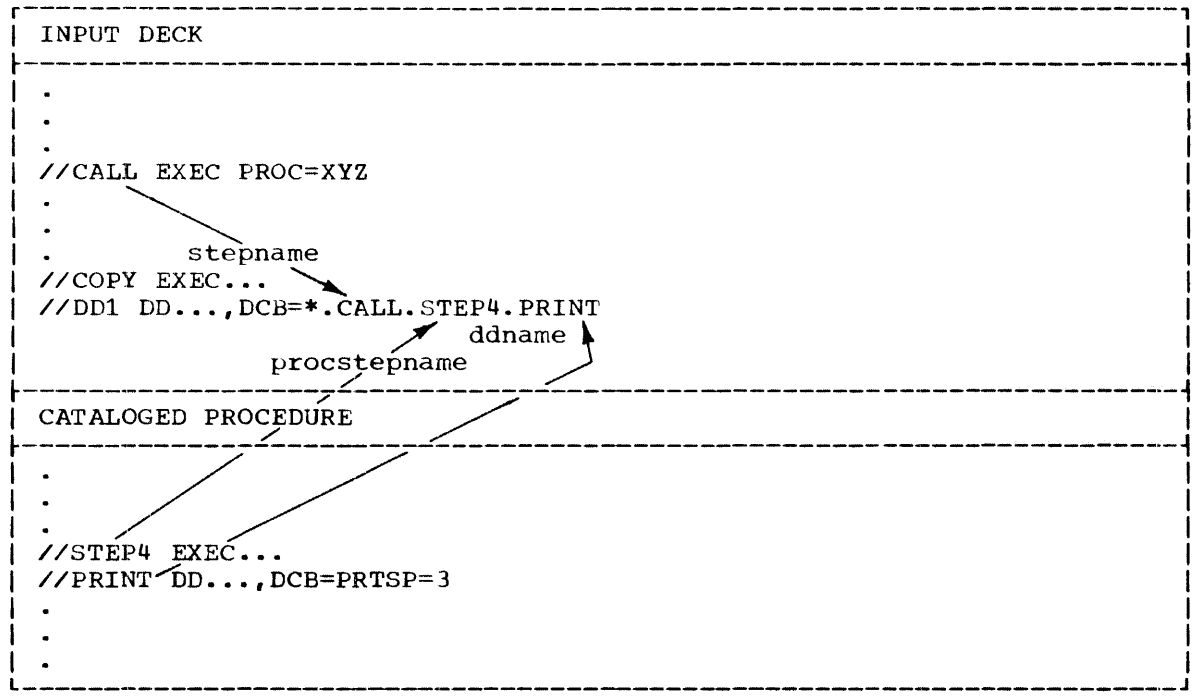
The following example shows how to code the DCB parameter to copy the DCB parameter of a previous DD statement in the same step:

```
.
.
.
//STEPD EXEC...
//DDA DD...,DCB=RECFM=F
.
.
.
//DDC DD...,DCB=*.DDA
.
.
.
```

If you want to copy the DCB parameter of a DD statement contained in a cataloged procedure step, you must give both the name of the job step that invokes the procedure and the procedure step name, i.e.,

```
DCB=*.stepname.procstepname.ddname
```

The first part of the following example shows your input deck; the second part shows a cataloged procedure called by the CALL EXEC statement in your deck.



If you want to modify the DCB subparameters you are copying add the new subparameters to the reference. The subparameters you specify will override the corresponding copied subparameters. For example,

```
.
.
.
//OUT EXEC...
//PUNCH1 DD...,DCB=(BLKSIZE=160,LRECL=80,OPTCD=C,MODE=F,RECFM=F)
//PUNCH2 DD...,DCB=(*.PUNCH1,MODE=C,STACK=2)
.
.
.
```

Special Processing Options

There are two special processing options:

1. If your data set is going to be printed on a 1403 printer with the universal character set feature, you can use the UCS parameter to specify a particular character set image. If you omit the UCS parameter when a 1403 printer with the universal character set special feature is to be used, a default character set is used if the corresponding print chain or cartridge is mounted. Otherwise, the operator is requested to specify the UCS parameter for a default character set and mount the appropriate print chain or cartridge.

2. You can suppress I/O operations on your data set using the DUMMY parameter.

UCS: The format of the UCS parameter is:

```
UCS=(code[,FOLD][,VERIFY])
     [,   ]
```

where:

code
is the character set code which corresponds to the IBM standard character set image you want to use. For example, if you want the "preferred character set, arrangement A", code UCS=PCAN. One of the following codes can be specified.

Code	IBM Standard Character
AN	Arrangement A, standard ECBDIC character set. 48 characters.
HN	Arrangement H, EBCDIC character set for FORTRAN and COBOL, 48 characters.
PCAN	Preferred character set, arrangement A.
PCHN	Preferred character set, arrangement H.
PN	PL/1 character set.
QNC	PL/1 preferred character set for commercial application.
QN	PL/1 preferred character set for scientific applications.
RN	Character set for commercial applications of FORTRAN and COBOL.
TN	Character set for text printing, 120 characters.
SN	Preferred character set for text printing.
XN	High-speed alphanumeric character set for 1403, Model 2.
YN	High-speed alphanumeric character set for 1403, Model 3 or N1.

Not all of the above character set images might be present in your operating system. We suggest you place a check mark next to those available to you. In addition to IBM character set images, your installation may have its own character set images that have been assigned unique character set codes. You should add those codes and a description of the corresponding character set images to the above list. Your manager or supervisor should give you a complete list of available codes. He will also tell you which character set images were selected as defaults during system generation.

FOLD
specifies that you want certain EBCDIC characters to be printed with the graphics corresponding to other EBCDIC characters. For example:

```
UCS=(TN,FOLD)
```

The FOLD option can only be coded when a character set code is specified with the code subparameter. That is, you cannot specify FOLD if you did not specify the code subparameter and are getting a

default character set. For details on the FOLD mode, refer to the publication IBM 2251 Control Unit, GA24-3312. The FOLD mode is most often requested when uppercase and lowercase data is to be printed only in uppercase.

VERIFY
specifies that you want a printer display of the character set images. This specification requests that the operator is to visually verify that the character set image corresponds to the graphics of the chain or train that was mounted. For example,

```
UCS=(QNC,,VERIFY)
```

The VERIFY option can only be coded when a character set code is specified with UCS parameter. That is, you cannot specify VERIFY if you did not specify the code subparameter and are getting a default character set.

Notes:

- You can specify both the FOLD and VERIFY options for the same character set. For example,

  ```
  UCS=(XN,FOLD,VERIFY)
  ```

- If you code VERIFY but omit FOLD, you must code a comma to indicate its absence. For example,

  ```
  UCS=(HN,,VERIFY)
  ```

- If you do not want the FOLD or VERIFY options, you can omit the parentheses. For example,

  ```
  UCS=RN
  ```

DUMMY: The DUMMY parameter allows you to bypass I/O operations and data set allocation and disposition. When your processing program asks to write the dummy data set, the writer request is recognized, but no data is transmitted. This facility is particularly useful when you are debugging your program because it saves you processing time. For example, you can suppress the writing of a data set until you are sure your program is going to produce meaningful output.

The DUMMY parameter is a positional parameter. That means that it must be coded first in the operand field. For example:

```
//OUTPUT DD DUMMY,UNIT=2520,DCB=BLKSIZE=50
//PRINT DD DUMMY,UNIT=1403
```

If you wish, you can omit all other parameters required to define your data set except DCB. (If you were not going to write the DCB parameter you do not have to write it.) For example, the two DD statements shown above could be coded as follows:

```
//OUTPUT DD DUMMY,DCB=BLKSIZE=50
//PRINT DD DUMMY
```

When you are ready to perform I/O operations on your data set, simply omit the DUMMY parameter and code all parameters required to write the data set. For example,

```
//OUTPUT DD UNIT=2520,DCB=BLKSIZE=50
//PRINT DD UNIT=1403
```

System Output Devices

Your installation can designate printers and card punch units as system output devices. These system output devices can be grouped into output classes. (These are the same output classes described for the MSGCLASS parameter of the JOB statement.) For example, one printer can be designated as a class for compilation listings; another printer with a special print chain can be a class for those output data sets that need the special chain. There can be up to 36 output classes, although most installations need fewer. The output classes are designated by a letter (A-Z) or a number (0-9). Your manager or supervisor should give you a list of the output classes in your installation and of the devices in each class.

Whenever you have an output data set to be printed or punched, you can either request a unit as described in "Unit Record Devices," or you can request the output class that has the type of device you need. (Please note that not every printer or card punch unit is necessarily assigned to an output class.) You indicate which output class you want with the SYSOUT parameter. For example, SYSOUT=M means that you want your data set produced on one of the devices of output class M.

Although your data set is finally produced on the output class you requested, the intervening process depends on whether your system has PCP, MFT, or MVT.

Output Class Processing in PCP: Before the jobstep is executed, the system notifies the operator of the output class you selected. The operator then makes available to the system one of the units in that output class. If none of the units in the output class is available at the time, the operator selects a tape unit and the system writes your output on tape. Later the operator transcribes the tape on a unit of the desired output class.

Output Class Processing in MFT and MVT: The system allocates a data set for you on a direct access volume and writes your output on this data set. Later, a system routine called the system output writer (or a special installation-written program) transfers your data set to a unit of the output class you selected. This allows greater flexibility in scheduling print and punch operations and improves operating system efficiency. (You do not have to allocate space for your data set on a direct access device. The system takes care of that automatically with a standard allocation. However, if you have an unusually large data set, you can override the standard allocation using the UNIT and SPACE parameters as described later on in this topic.)

A system with MFT or MVT can also write an output data set directly to the desired unit record or magnetic tape device. When using the direct system output processing, the operator selects a unit record or magnetic tape device for a class by issuing a START DSO (direct system output) command. In addition to the SYSOUT parameter, the DCB and UCS parameters can be coded. If the SYSOUT subparameters other than classname are coded, the specified information is ignored. The UNIT and SPACE parameters are also ignored if the direct system output writer is used. Since the type of processing to be used may not always be known, it is advisable to code these parameters in case an intermediate direct access device is used.

This section is summarized in Table 64 of Appendix D.

Location of the Data Set

You indicate your selected output class you selected with the SYSOUT parameter. If your system has MFT or MVT, your data set can first be written on a direct access device and then transferred to the output class selected. The system assigns you a direct access device unless you request a specific one with the UNIT parameter.

SYSOUT: In systems with PCP, you use the SYSOUT parameter to specify the output class. In systems with MFT or MVT, you can also specify that a special installation-written program is transferred to your data set to the unit record device, and that the data set is to be printed or punched on a special output form.

The format of the SYSOUT parameter for PCP is:

```
SYSOUT=classname
```

Replace the term "classname" with the letter (A-Z) or number (0-9), that indicates the output class desired. If you specify SYSOUT=A, the output data set and system messages resulting from your job will be printed in chronological order on the same output listing. A is the standard output class in PCP and corresponds to a printer.

For example, if you code:

```
//PRINT DD SYSOUT=D
```

your data set will be produced on a device belonging to output class D. If none of those devices is available, the operator makes the system write your data set on tape. At a later time the operator transfers your data set from tape to a device in output class D.

The format of the SYSOUT parameter for MFT or MVT is:

```
SYSOUT=(classname[,program][,form number])
                 [,      ]
```

Replace the term "classname" with the letter (A-Z) or number (0-9) that indicates the output class desired. When using direct system output processing, all SYSOUT subparameters other than classname are ignored if coded.

If you want the output data set and the system messages resulting from your job to be printed in chronological order on the same output listing, specify the same output class you requested with the MSGCLASS parameter of the JOB statement. If you omitted the MSGCLASS parameter, code SYSOUT=A unless the default for the MSGCLASS parameter is not A for your installation.

Replace the term "program" with the name of the special installation program that is to handle the transfer of your data set from direct access to the output class requested. If you do not specify a special program, the standard system output writers will handle the transfer. Your manager or supervisor will tell you whether you should request a special program rather than the standard system output writer.

Replace the term "form number" with the 4-digit form number of a special printer paper or punched card stock you want to use for your data set. If you do not specify a form number your installation provides some standard type of paper or card. Your manager or supervisor will tell you whether you should use a special form number for the output data set.

In the following example you request output class 8 for your data set.

```
//DD8 DD SYSOUT=8
```

Using the same example, if you want to use special form 2107 for writing your data set, code:

```
//DD8 DD SYSOUT=(8,,2107)
```

In the following example, you request output class R for your data set. You also want a program named WTRPRT to transfer the data set from the direct access device to a device of output class R.

```
//PRINT DD SYSOUT=(R,WTRPRT)
```

Using the same example, if you also want your data set written on special form 7329, code:

```
//PRINT DD SYSOUT=(R,WTRPRT,7329)
```

UNIT: The UNIT parameter can be used in system with MFT or MVT to indicate the type of direct access device on which your data set will be recorded before being transferred to the output class by the system output writer. In the UNIT parameter, you can request the type of direct access device you want, how many devices you want (up to a maximum of five), and unit separation from other data sets. (Do not request more than one unit unless you also use the SPACE parameter to make a secondary allocation.) The UNIT parameter should be coded as described in "Direct Access Devices."

If you omit the UNIT parameter the system assigns an available unit.

In the following example you request that a 2314 drive be used to store your data set before it is transferred to output class B.

```
//PUNCH DD SYSOUT=B,UNIT=2314
```

Using the same example, if you want a special program named WTRPCH to effect the transfer from the 2314 unit to output class B, code:

```
//PUNCH DD SYSOUT=(B,WTRPCH),UNIT=2314
```

Size of the Data Set

In systems with MFT or MVT your data set can be written on a direct access device. The system allocates a standard amount of space on the device. Your manager or supervisor should tell you the size of the standard allocation. If your data set is unusually large you can override the standard allocation with the SPACE parameter.

SPACE: The SPACE parameter is described in "Direct Access Devices." You can request space in units of tracks, cylinders, or blocks. If you request it in units of tracks or cylinders you should also specify the UNIT parameter for more efficient allocation.

If you make a secondary allocation, you can specify more than one device in the UNIT parameter. You can also specify the RLSE subparameter to release any unused space.

In the following example, a data set is to be written on a device of output class P. Instead of letting the system use the stand default values you want to assign 20 cylinders on a 2314. In case 20 cylinders are not enough, you want to make a secondary allocation of 5 cylinders.

```
//OUT DD SYSOUT=P,UNIT=2314,SPACE=(CYL,(20,5))
```

Using the preceding example, if you think two 2314 units may be needed, and if you also want to release unused space code:

```
//OUT DD SYSOUT=P,UNIT=(2314,2),SPACE=(CYL,(20,5),RLSE)
```

Data Attributes

The DCB parameter allows you to specify data set attributes for your data set. In PCP, the data attributes you specify are those of a printer or card punch. In MFT and MVT, the data attributes you specify are those of your SYSOUT data set on a direct access device.

DCB: The DCB parameter for PCP is described in "Unit Record Devices", (Tables 7 and 8) and, for MFT and MVT, in "Direct Access Devices" (Table 12).

In the following example, you specify printer spacing (PRTSP) for a data set that is to be printed in output class A.

```
//SYSPRINT DD SYSOUT=A,DCB=PRTSP=2
```

Special Processing Option

The OUTLIM parameter allows you to specify a limit for the number of logical records you want included in the output data set being routed through the output stream. The OUTLIM parameter has meaning only in systems with MFT or MVT that have the System Management Facilities option with system, job, and step data collection. Your manager or supervisor should tell you whether your system has the System Management Facilities option.

OUTLIM: The format of the OUTLIM parameter is:

```
OUTLIM=number
```

Replace the term "number" with a numeric value of 1 through 16777215. The OUTLIM parameter is ignored unless SYSOUT is coded in the operand field of the same DD statement.

The limit for the number of logical records you want as output must include a system overhead factor. Generally, the value you add to the limit is eight times the blocking factor for your data. (For those programmers who need a more precise value: The system overhead is the number of EXCPs issued each time the OPEN or CLOSE macro instruction is issued for the data set.)

When the number specified is reached, an exit provided by the System Management Facilities option is taken to a user supplied routine that determines whether to cancel the job or increase the limit. If the exit routine is not supplied, the job is cancelled.

A discussion of the System Management Facilities option is contained in the publication IBM System/360 Operating System: Concepts and Facilities. Information on user exit routines to be used with the System Management Facilities option is contained in the publication IBM System/360 Operating System: System Programmer's Guide.

Magnetic Tape

You can create an output data set on magnetic tape using the parameters shown in Table 6. The data set information parameters (DSNAME and DISP) let you specify whether your data set is to be temporary or nontemporary. The location parameters (UNIT, VOLUME, and LABEL) let you specify whether your data set is to occupy one volume (one reel of tape), several volumes, or part of a volume and the type of labels you use for those volumes. You can also use special processing options (SEP and AFF) to optimize channel use.

This section is summarized in Table 65 of Appendix D.

Data Set Information

The DSNAME and DISP parameter are used to specify which type of data set you want to create. There are two types of data set: temporary and nontemporary. A temporary data set is one that will be used for the duration of the jobstep or the job only. A nontemporary data set is one that can be used not only by the jobsteps in your job, but by later jobs as well. There are three types of nontemporary data sets: cataloged data sets, kept (or noncataloged) data sets, and members of generation data groups.

The following is a summary of the types of data sets that can be created on magnetic tape:

1. Temporary
   a. For the duration of the job.
   b. For the duration of the jobstep.

2. Nontemporary
   a. Cataloged.
   b. Kept (or noncataloged).
   c. Member of a generation data group.

The formats of the DSNAME and DISP parameters for each type of data set are described below.

Temporary -- For the Duration of the Job: The formats of the DSNAME and DISP parameters are:

```
DSNAME={&&name},DISP=(NEW,PASS,DELETE)
       {&name }
```

In the DSNAME parameter, replace the term "name" with any name not used by another temporary data set in the job. Each name consists of one to eight alphameric or national (@,$,#) characters. However, the first character of the name must be a letter or national character. The system generates a name for the temporary data set, which begins with SYS and includes the jobname, the temporary name assigned in the DSNAME parameter, and other identifying characters. You can retrieve this data set later in the job by coding

```
DSNAME=&&name
```

or

```
DSNAME=&name
```

in a DD statement, using the same name, or by coding:

```
DSNAME=*.stepname.ddname.
```

A double ampersand should be coded preceding the temporary name you assign to a data set. However, a name preceded by a single ampersand is treated as a temporary data set name, as long as no value is assigned to it either on an EXEC statement invoking a procedure, or a PROC statement within a procedure. If a value is assigned to it by one of these means, it is treated as a symbolic parameter. (Symbolic parameters are discussed in the section "The PROC Statement" and in "Part III: Cataloged Procedures.")

In the DISP parameter, the NEW subparameter indicates that you are creating the data set. You do not have to code NEW as long as you indicate its absence with a comma. The PASS subparameter indicates that the temporary data set can be used by later steps in the job. The DELETE subparameter indicates that this data set is to be deleted if the job step abnormally terminates (ABEND). You need not code DELETE because it is always assumed by the system for temporary data sets.

For example, the parameters DSNAME and DISP in the following DD statement indicate that the temporary data set named AREA can be used by later steps in the job:

```
//DD1 DD DSNAME=&&AREA,DISP=(,PASS),...
```

Temporary -- For the Duration of the Job Step: Both the DSNAME and DISP parameters can be omitted to indicate a temporary data set which will be deleted at the end of the job step. You can, if you wish, code the following:

```
DSNAME={&&name} ,DISP=(NEW,DELETE,DELETE)
       {&name }
```

The temporary name in the DSNAME parameter is coded as described for temporary data sets for the duration of the job. All subparameters of the DISP parameter are assumed if omitted. For example, the three following DD statements define the same data set:

```
//DD2 DD DSNAME=&&AREA,DISP=(NEW,DELETE,DELETE),...
```

or

```
//DD2 DD DSNAME=&&AREA,...
```

or

```
//DD2 DD...
```

Note that the location parameters must still be coded in those statements.

Nontemporary -- Cataloged: The formats of the DSNAME and DISP parameter are:

```
DSNAME=dsname,DISP=(NEW,CATLG[,CATLG ])
                              [,DELETE]
                              [,KEEP  ]
```

In the DSNAME parameter, replace the term "dsname" with the qualified name you want to give to the data set. A qualified name is made up of several 1-to-8-character names separated by periods. All but the last of those names correspond to index levels in the catalog. Each name must begin with a letter or national (@,$,#) character. Any letter, number, national character, the hyphen, and the +0(12-0 multipunch) can be used to complete each name. A qualified name can consist of 1 through 44 characters (including periods). For example,

```
DSNAME=AB-5@3.M.$7.R2579AB2
```

Each level of qualification must already exist as an index in the system catalog before you can request the system to catalog the data set. An index level is created using the IEHPROGM utility program. Once the indexes are established, the data set can be cataloged. If the name has no qualifiers (called unqualified name), the system will create the index entry for you. For example,

```
DSNAME=C174B
```

or

```
DSNAME=D
```

In the DISP parameter, the NEW subparameter indicates that you are creating the data set. You do not have to code NEW as long as you indicate its absence by a comma. The CATLG subparameter indicates that the data set is to be cataloged. The third subparameter tells the system what is to be done with the data set if the step abnormally terminates (ABEND). If the data set name is either qualified or unqualified, you can specify CATLG or DELETE. CATLG is assumed if you omit the third subparameter. If the data set is unqualified, you can specify KEEP as the third subparameter. KEEP tells the system that if the step abnormally terminates you want the data set kept intact (but not cataloged) until a later job requests that the data set be deleted or until the expiration date has passed. (You can specify a retention period or expiration date in the LABEL parameter.)

In the following example, a new data set named SYSTEM.FILE1 is to be cataloged. If the step abnormally terminates it will be cataloged.

```
//DD3 DD DSNAME=SYSTEM.FILE1,DISP=(,CATLG),...
```

In the following example, a new data set named AREA#5 is to be cataloged. If the step abnormally terminates, it will be kept.

```
//DD4 DD DSNAME=AREA#5,DISP=(,CATLG,KEEP),...
```

Nontemporary -- Kept: The formats of the DSNAME and DISP parameters are:

```
DSNAME=dsname,DISP=(NEW,KEEP[,KEEP  ])
                           [,DELETE]
                           [,CATLG ]
```

In the DSNAME parameter, replace the term "dsname" with the unqualified name you want to give to the data set. The name can contain from 1 to 8 characters. The first character must be a letter or a national character (@,$,#). Any letter, number, national character, the hyphen, and the +0(12-0 multipunch) can be used to complete the name. For example,

```
DSNAME=@A#753
```

In the DISP parameter, the NEW subparameter indicates that you are creating the data set. You do not have to code NEW as long as you indicate its absence by a comma. The KEEP subparameter indicates that the data set is to be kept intact (but not cataloged) until a later job step or job requests that the data set be deleted or until the expiration date has passed. (You can specify a retention period or expiration data in the LABEL parameter.) The third subparameter tells the system what is to be done with the data set if the step abnormally terminates (ABEND). You can specify KEEP, DELETE, or CATLG. KEEP is assumed if you omit the third subparameter.

In the following example, a new data set named RTW is to be kept. If the step abnormally terminates, it will also be kept.

```
//DD5 DD DSNAME=RTW,DISP=(,KEEP),...
```

In the following example, a new data set named TWO is to be kept. If the step abnormally terminates it will be deleted.

```
//DD6 DD DSNAME=TWO,DISP=(,KEEP,DELETE),...
```

Notes:

- You can give a disposition of PASS to a new data set with an unqualified name. For example,

  ```
  DSNAME=DATA,DISP=(,PASS)
  ```

  Such a data set can become a temporary, cataloged, or kept data set according to the disposition you give to it in a later DD statement. See "Passed Data Sets" in "Retrieving an Existing Data Set." If the data set is not referred to by a later DD statement in the job, it is deleted at the end of the job.

- You can use special characters as part of the unqualified data set name if you enclose the name in apostrophes. For example,

  ```
  DSNAME='AB=750'
  ```

  If the special character is an apostrophe, you must code it as two consecutive apostrophes. For example, code NAT'L as

  ```
  DSNAME='NAT''L'
  ```

  Do not catalog the data set if you enclose the data set name in apostrophes.

Nontemporary -- Member Generation Data Group: A cataloged data set that is periodically processed can be grouped with its earlier generations to form a named generation data group. The entire group has a generation group name. Each member of the group can be addressed by a simple generation number. The last generation produced before the start of your job has a generation number of 0. The generation produced before the last one is -1. The generation you are going to produce in your job is +1. After your job ends, the generation you produced automatically becomes generation 0. If you produce more than one generation in your job, you must name them +1, +2, +3, etc. After your job, the highest generation number you produced, say, +3, becomes generation 0; +2 becomes -1; and +1 becomes -2.

For further information on creating generation data groups, refer to Appendix D of the publication IBM System/360 Job Control Language Reference. The formats of the DSNAME and DISP parameters for creating a new member of a generation data group are:

```
DSNAME=groupname(+number),DISP=(NEW,CATLG[,CATLG ])
                                        [,DELETE]
```

In the DSNAME parameter, replace the term "groupname" with the name of the generation data group. The name can contain from one to eight characters. The first character must be a letter or national character (@,$,#). Any letter, number, national character, the hyphen, and the +0(12-0 multipunch) can be used to complete the name. The groupname can be a qualified name. The qualified name can contain up to 35 characters (including periods). Replace the term "number" with the number of the generation you are creating. If you are only creating one generation in your job, this number will be 1. For example,

```
DSNAME=PAYROLL.PLANT20(+1)
```

or

```
DSNAME=MASTER(+1)
```

If you are creating more than one generation in your job, number each generation consecutively. For example,

```
DSNAME=MASTER(+1)

DSNAME=MASTER(+2)
```

etc.

In the DISP parameter, the NEW subparameter indicates that you are creating the data set. You do not have to code NEW as long as you indicate its absence by a comma. The CATLG subparameter indicates that the new generation is to be cataloged. (This subparameter is required.)

The third subparameter tells the system what is to be done with the data set if the step abnormally terminates (ABEND). You can specify CATLG or DELETE. CATLG is assumed if you omit the third subparameter.

In the following example, two new generations are added to the PAYROLL generation data group. If the step abnormally terminates the new generations will be deleted.

```
//STEP EXEC PGM=...
.
.
.
//DD7 DD DSNAME=PAYROLL(+1),DISP=(,CATLG,DELETE),...
//DD8 DD DSNAME=PAYROLL(+2),DISP=(,CATLG,DELETE),...
.
.
.
```

After your job is successfully completed PAYROLL(+2) becomes PAYROLL(0), and PAYROLL(+1) becomes PAYROLL(-1).

Location of the Data Set

The UNIT, VOLUME and LABEL parameters are used to describe the location of your data set. The UNIT parameter indicates which kind of tape drive you want, and, if your data set is to occupy more than one volume, it indicates how many tape drives are needed. The VOLUME parameter lets you request a scratch volume (or volumes) or a private volume (or volumes). If you request a private volume you can request a specific volume or you can let the system assign you a volume. The LABEL parameter describes the tape label of the volume used for your data set.

UNIT: The format of the UNIT parameter is:

```
                  ⎧unit address⎫ ⎡,unitcount⎤
UNIT=(            ⎨device type  ⎬ ⎢,P       ⎥ [,DEFER])
                  ⎩group name  ⎭ ⎣,        ⎦
```

The first positional subparameter identifies the tape drive you want to use for your data set by its address, or unit name, or group name.

unit address
: is the actual machine address of the tape drive. For example,

  UNIT=240

  You should not specify the address unless you are sure you want it. Do not specify an address if you are going to need more than one drive for the data set. (Multiple drives are requested with the second subparameter of the UNIT parmeter.)

device type
: corresponds to the type of tape drive. Coding a device type provide you with a certain degree of device independence in that your data set may be placed in any number of devices of the same type. For example, if you code

  UNIT=2400-1

your data set will be placed on any 2400 series tape drive with seven-track compatibility and without data conversion. The following device types can be specified.

Device Type	Description
2400	2400 series Nine-Track Magnetic Tape Drive that can be allocated to a data set written or to be written in 800 bpi density when the dual density feature is not installed on the drive, or in 1600 bpi when the dual-density feature is installed in the drive.
2400-1	2400 series Magnetic Tape Drive with Seven-Track Compatibility and without Data Conversion.
2400-2	2400 series Magnetic Tape Drive with Seven-Track Compatibility and Data Conversion.
2400-3	2400 series Nine-Track Magnetic Tape Drive that can be allocated to a data set written or to be written in 1600 bpi density.
2400-4	2400 series Nine-Track Magnetic Tape Drive having an 800 and 1600 bpi capability.

group name
is the name of a collection of devices, selected by your installation during system generation. For example, your installation might select the name TAPE for all tape drives in the configuration. If you do not care which tape drive is used for your data set, you would then code

```
UNIT=TAPE
```

Your manager or supervisor should tell you which group names were generated for your installation.

The second positional subparameter is used only if your data set will occupy more than one volume and if you want more than one of those volumes to be mounted at the same time. This subparameter indicates how many drives are to be used for mounting your data set's volumes.

unit count
indicates how many tape drives you want assigned to the data set. You can specify a maximum of 59 drives per DD statement. Make sure that your system has at least the number of drives you specify. Otherwise an error will result. If you specify fewer drives than the number of volumes in your data set, only the same number of volumes as there are drives can be mounted at the same time. (If you think your data set may use more volumes than you expect, you should indicate the maximum number of volumes that can be used with the volume count subparameter of the VOLUME parameter. Do not request any more tape drives than the maximum number of volumes you specify.) For example, if you write

```
UNIT=(2400,5)
```

five 800 bpi drives will be assigned to your data set.

P
specifies parallel mounting. When you request parallel mounting the system counts the number of serial numbers specified with the VOLUME parameter and assigns to the data set as many tape drives as there are serial numbers. You should only use the P subparameter if you are making specific volume requests in the VOLUME parameter.

If, in addition to indicating specific serial numbers in the VOLUME parameter, you also indicate a maximum number of volumes that can be used by the data set, the number of drives assigned is the number of serial volumes you specified and not the larger maximum number of volumes that can be used.

If your data set required only one tape drive, you can either code 0 or omit this parameter. If you omit this subparameter and the DEFER subparameter follows, you should code a comma to indicate its absence. For example,

```
UNIT=(2400,,DEFER)
```

DEFER is the third positional subparameter. DEFER requests the system to assign the required tape drives to the data set and to defer the mounting of the volume or volumes until the processing program attempts to open the data set. If you are running under MFT or MVT, operating efficiency may decrease if you specify DEFER for a tape data set. This is because if you code DEFER, the system issues a mount message to the operator when the processing program attempts to use the data set and then waits until the volume is mounted. If you do not code DEFER, the mount message is issued when the device is assigned and there is no waiting for the operator. By the time your program tries to use the data set, it is very likely that the operator has already mounted the volume and that the system will not have to wait for him.

In the following example, the UNIT parameter is used to request four seven-track tape drives with data conversion.

```
UNIT=(2400-2,4)
```

In the following DD statement, the unit parameter is used to request as many tape drives of a group named TAPE as there are volumes specifically requested with the volume parameter. Deferred mounting is also requested.

```
//DD9 DD UNIT=(TAPE,P,DEFER),VOLUME=SER=(ABC24,ABC25,ABC29),...
```

As a result three tape drives are assigned to the data set, and all three volumes requested can be mounted at the same time.

Note: The UNIT parameter is ignored if you use the VOLUME parameter to request a volume used by a data set defined earlier in your job, that is, if you specify VOLUME=REF.

VOLUME: The VOLUME parameter lets you request a specific volume for your data set or lets the system assign your data set a suitable volume (nonspecific request). If you make a nonspecific request, you can ask for a scratch volume or for a private volume. If you make a specific request, you get a private volume. Scratch volumes can only be requested for temporary data sets. Private volumes can be requested for either temporary or nontemporary data sets. The following is a summary of the types of volume request that can be made for new data sets on magnetic tape:

1. Nonspecific request

    a. Scratch volume
    b. Private volume

2. Specific request

    a. Private volume

Note: When you ask the system to assign you an available volume (nonspecific request), you can also specify which kind of labels you want that volume to have. You simply indicate the label type with the LABEL parameter and the system will make a volume with that label type available to you.

Nonspecific Request -- Scratch Volume: If you have a temporary data set you can request a scratch volume. The format of the VOLUME parameter depends on whether you need one volume or more than one volume.

1. To request only one scratch volume simply omit the VOLUME parameter. For example, if you have a temporary data set that will last only for the duration of the job step and you want a scratch volume to be mounted on a nine-track (800-bpi) tape drive, code:

   ```
   //DD10 DD UNIT=2400,...
   ```

   If you have a temporary data set named &&AREA that is to be passed to other steps in your job and you want a scratch volume to be mounted on a seven-track drive with data conversion, code:

   ```
   //DD11 DD DSNAME=&&AREA,DISP=(,PASS),UNIT=2400-2,...
   ```

2. If your temporary data set requires more than one scratch volume, code the VOLUME parameter as follows:

   ```
   VOLUME=(,,,volcount)
   ```

   Replace the term "volcount" with the number of volumes required for your data set. For example, if your data set requires four scratch volumes and you want them in succession on one nine-track tape drive, code:

   ```
   //DD12 DD UNIT=2400,VOLUME=(,,,4),...
   ```

   If your temporary data set requires six scratch volumes, and you want to have them mounted two at a time on two nine-track drives, code:

   ```
   //DD13 DD UNIT=(2400,2),VOLUME=(,,,6),...
   ```

Nonspecific Request -- Private Volume: If you make a nonspecific request for a private volume, the system assigns an available volume to you. The private volume is demounted after its use in the job step, unless a MOUNT command was used to mount it or the data set is passed to a later job step.

The format of the VOLUME parameter depends on whether your data set is temporary or nontemporary.

1. The format of the VOLUME parameter for making a nonspecific volume request for temporary data sets is:

   ```
   VOLUME=(PRIVATE[,,,volcount])
   ```

The PRIVATE subparameter is required. If your temporary data set requires more than one volume, replace the term "volcount" with the number of volumes required.

For example, if you have a temporary data set named &&FILE that is to be passed to other steps in your job and you want two private volumes to be mounted in sequential order on a drive of the group named TAPE, code:

```
//DD14 DD DSNAME=&&FILE,DISP=(,PASS),UNIT=TAPE,
//        VOLUME=(PRIVATE,,,2),...
```

If the &&FILE data set only needs one volume, code:

```
//DD15 DD DSNAME=&&FILE,DISP=(,PASS),UNIT=TAPE,
//        VOLUME=PRIVATE,...
```

If the &&FILE data set needs five volumes to be mounted at the same time on five drives, code:

```
//DD16 DD DSNAME=&&FILE,DISP=(,PASS),UNIT=(TAPE,5),
//        VOLUME=(PRIVATE,,,5),...
```

In the preceding examples the volume is not demounted at the end of the step because you coded PASS in the DISP parameter. When the volumes are mounted in sequential order, only the last volume used remains mounted at the end of the step.

If your temporary data set is used only in one job step the volume is demounted at the end of the step. For example,

```
//DD17 DD UNIT=TAPE,VOLUME=PRIVATE
```

or, if your temporary data set needs three volumes, code

```
//DD18 DD UNIT=TAPE, VOLUME=(PRIVATE,,,3)
```

2. The format of the VOLUME parameter for making a nonspecific volume request for a nontemporary data set is:

```
VOLUME=( [PRIVATE,RETAIN] [,,volcount] )
         [,             ]
```

The subparameter PRIVATE is not needed when you have a nontemporary data set because the system always assigns a private volume to nontemporary tape data sets. You only need to code PRIVATE when you also code RETAIN.

RETAIN means that the volume will not be demounted at the end of the step. (RETAIN has the same effect on volume mounting as coding PASS in the DISP parameter.) For example, if you want a nonspecific private volume for a data set named ALPHA, code:

```
//DD19 DD DSNAME=ALPHA,DISP=(,KEEP),UNIT=2400
```

If you want to make sure that the nonspecific volume used for ALPHA remains mounted at the end of the step, code:

```
//DD20 DD DSNAME=ALPHA,DISP=(,KEEP),UNIT=2400,
//        VOLUME=(PRIVATE,RETAIN)
```

If your nontemporary data set needs more than one volume, replace the term "volcount" with the number of volumes needed. For example, if the data set ALPHA BETA needs four volumes to be mounted in succession on two tape drives, code:

```
//DD21 DD DSNAME=ALPHA.BETA,DISP=(,CATLG),UNIT=(TAPE,2),
//        VOLUME=(PRIVATE,RETAIN,,4)
```

Notes:

- The RETAIN subparameter is not needed if the tape volume was mounted with the MOUNT command. A volume thus mounted remains mounted until an UNLOAD command is issued. It is recommended that you use the RETAIN subparameter whenever you want the volume to remain mounted at the end of the step.
- If you specify KEEP for your nontemporary data set, ask the machine operator to tell you the serial number of the volume assigned to your data set. You need the serial number to retrieve your data set in later jobs.

Specific Request -- Private volume: You can request specific volumes either stating the serial number of the volumes you want, or by requesting the same volume used by an earlier data set.

1. The format of the VOLUME parameter for requesting volumes by their serial numbers is:

```
VOLUME=([PRIVATE,RETAIN][,,volcount,]SER=(serial,...))
        [,              ][,        ]
```

The PRIVATE subparameter is not needed when you make a specific volume request, because the system always considers specific tape volumes to be private volumes. You only need to code PRIVATE when you also code RETAIN. RETAIN means that the volume will not be demounted at the end of the step. If your data set has more than one volume, and the volumes are mounted in sequential order, only the last volume mounted is retained. (If more than one tape drive is available to the multivolume data set, the last volumes mounted are retained.)

Replace the term "volcount" with the maximum number of volumes the data set can use. You should only use this subparameter if you think your data set may use more volumes than those named in the SER subparameter. Note that "volcount" is the maximum number of volumes that can be used and not additional volumes. For example, if you name three volumes in the SER subparameter, but you think you may need two more volumes, replace "volcount" with 5. In this case, you are in effect making a specific request for three volumes and a nonspecific request for two volumes.

The SER subparameter indicates that the serial numbers of the volumes you want for your data set will follow. Replace the term "serial" with the 1-to-6 character serial number associated with the volume. If the volume serial number is not 6 characters, it will be padded with trailing blanks by the system. The volume serial number can contain any alphameric and national characters

and the hyphen. If it is necessary to include special characters, other than a hyphen, in the volume serial number, enclose it in apostrophes, for example,

```
VOLUME=SER=(54AB2,'6/10/8')
```

If only one volume is involved you need not code the parentheses, for example,

```
VOL=SER=ABCDEF
```

In the following example you request that volumes AA22 and AB45 be assigned to the A99.F77 data set. Only one tape drive named TAPE will be used and the last volume used is to remain mounted at the end of the step.

```
//DD23 DD DSNAME=A99.F77,DISP=(,CATLG),UNIT=TAPE,
//        VOLUME=(PRIVATE,RETAIN,SER=(AA22,AB45)),...
```

In the following example, the same data set, A99.F77, is defined, but the possibility that it may need one extra volume is indicated.

```
//DD24 DD DSNAME=A99.F77,DISP=(,CATLG,UNIT=TAPE,
//        VOLUME=(PRIVATE,RETAIN,,3,SER=(AA22,AB45)),...
```

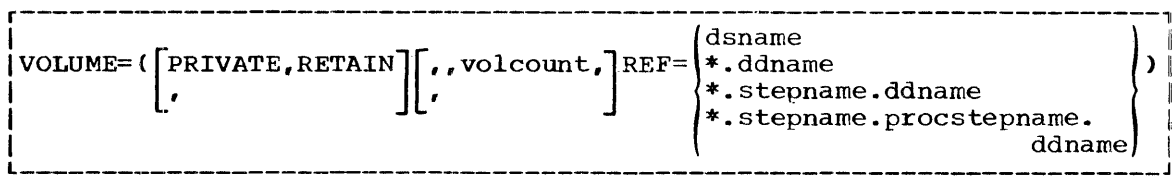
In the following example, you request that volumes ABC, BCD, and CDE be assigned to the RECORD data set. All three volumes are to be mounted in parallel on units of the group named TAPE7.

```
//DD25 DD DSNAME=RECORD,DISP=(,KEEP),UNIT=(TAPE7,P),
//        VOLUME=SER=(ABC,BCD,CDE),...
```

Notes:

- When using various typewriter heads or printer chains, difficulties in volume serial number recognition may arise if you use other than alphameric characters.
- SCRTCH should not be used as a volume serial number, because it is used to notify the operator to mount a non-specific volume.

2. The format of the VOLUME parameter for requesting volumes used by other data sets is:



The PRIVATE and RETAIN parameters are used as described for VOLUME=SER. The REF subparameter designates only one volume. Therefore, if your data set requires more than one volume, you should indicate so with the "volcount" subparameter. Replace "volcount" with the maximum number of volumes for your data set.

The REF subparameter identifies the volume assigned to an earlier data set. If the earlier data set resides on more than one tape volume, only the last volume is assigned to your data set. You can identify the earlier data set by its name or by making a backward reference.

You can identify the data set by its name only if it is a nontemporary cataloged data set or a nontemporary passed data set. If one of these conditions is met, replace the term "dsname" with the data set's name. The data set's name cannot contain special characters, except for periods in the case of qualified names. For example,

```
VOLUME=REF=AB.CD.EF
```

If the data set is not cataloged or passed, or if it has been assigned a temporary name, you can refer to it by making a backward reference to the DD statement that defined it. (This DD statement must be an earlier DD statement in your job.) Replace "ddname" with the name of the DD statement where the earlier data set is defined. Replace "stepname" with the name of the EXEC statement of the step that has the earlier DD statement. If the earlier DD statement is contained in the same job step, omit the stepname, i.e.,

```
VOLUME=REF=*.ddname
```

The following example shows how to code the REF subparameter to use the volume specified in a DD statement in a previous step. The previous step defines a multivolume data set, therefore, only the last volume is assigned to the new data set.

```
//STEP5 EXEC...
//DD1 DD UNIT=2400,VOLUME=SER=(75934,34AB5,1679M)
.
.
.
//STEP7 EXEC...
//DDA DD VOLUME=REF=*.STEP5.DD1,...
.
.
.
```

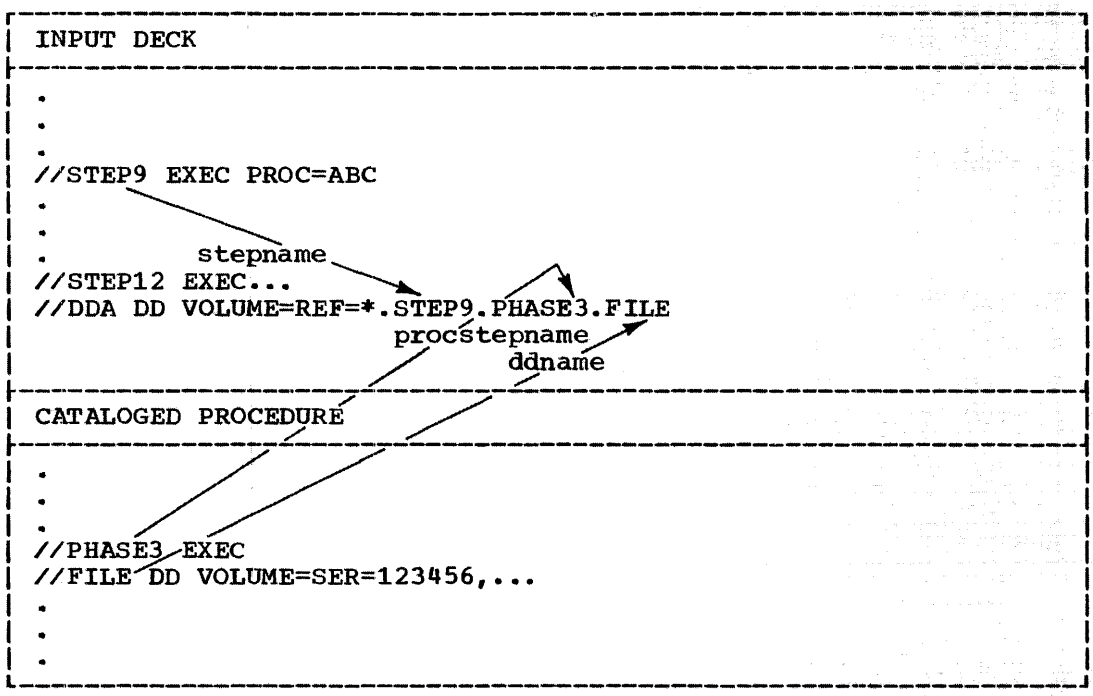
The following example shows how to code the REF subparameter to use the volume specified in a DD statement in a previous step. In addition to the volume used by the previous data set, the new data set may use three more volumes.

```
//STEPC EXEC...
//DD1 DD VOLUME=(PRIVATE,RETAIN,SER=ABC111),...
.
.
.
//DD5 DD VOLUME=(,,,4,REF=*.DD1),...
.
.
.
```

When the earlier DD statement is contained in a cataloged procedure step, you must give both the name of the job step that invokes the procedure and the procedure step name, i.e.,

```
VOLUME=REF=*.stepname.procstepname.ddname
```

The first part of the following example shows your input deck; the second part shows a cataloged procedure called by the STEP9 EXEC statement in your job.



If you want to use the volume assigned to a kept data set defined in another job, you must use VOLUME=SER.

When you request a volume using the REF subparameter, the system obtains unit type information from the referenced data set. The UNIT parameter is ignored if you code it in the same DD statement as REF.

LABEL: The LABEL parameter is used to describe the type of label of the volume you selected with the VOLUME parameter. It is also used to indicate how many data sets, if any, precede your data set on the volume. The LABEL parameter is also used to assign a retention period and password protection to your data set, and to indicate whether this data set is to be used for input or output exclusively. The format of the LABEL parameter is:

```
LABEL=([sequence] [,SL  ] [,PASSWORD] [,IN  ] [,] [EXPDT=yyddd])
                  [,SUL ]             [,OUT ]      [RETPD=nnnn ]
                  [,NSL ]
                  [,NL  ]
                  [,BLP ]
                  [,    ]
```

The first positional subparameter indicates the order of your data set on the tape volume. You only have to specify a sequence number if you make a specific volume request (VOL=REF or VOL=SER) and if your data set is not the first data set on the volume. If you do not specify a sequence number and there are other data sets on the volume, your data set will be written over the existing data sets. (If the existing data set was assigned a retention period or password protection, your data

set will not be written and an error will result.) Replace the term "sequence#" with the sequence number of your data set. For example, if your data set is to be the fifth data set on the volume, code

```
LABEL=5
```

Note: If you request the system to bypass label processing (by specifying BLP in the second positional subparameter of the LABEL parameter), the system treats anything within tape marks as a data set. Therefore, if you want your data set to be written in the proper sequence, you must include all header and trailer labels and data sets that precede your data set in the sequence number subparameter.

The second positional subparameter indicates the type of labels used for the specific volume you requested, or, if you are requesting a nonspecific volume, the type of labels you want on that volume. You can omit this subparameter if standard labels are used.

You must specify one of the following:

SL
: if the data set has standard labels. If you specify SL (or omit the second positional subparameter) the system can ensure that the specific volume you requested is mounted, because the volume serial number is written on the label. If you make a nonspecific request and the operator mounts a tape that does not have standard labels, the system asks the operator for the volume serial number and your name so that it can include this information on the new standard label.

SUL
: if the data set has both standard and user labels. If you specify SUL, the system can ensure that the specific volume you requested is mounted because the volume serial number is written on the label.

NL
: if the data set has no labels. The operator must ensure that the specific volume requested is mounted, because the volume has no labels to indicate the serial number and the system cannot verify this information. If you make a nonspecific request and the operator mounts a volume with standard labels, you may use the volume provided: (1) the expiration date of any existing data set on the volume has passed, and (2) there are no password protected data sets on the volume. If either condition is not met, the system asks the operator to mount another volume.

NSL
: if the data set has nonstandard labels. Nonstandard labels are processed by installation-written routines. These routines must ensure that the specific volume requested is mounted. If you make a nonspecific request and the operator mounts a volume with standard labels, you may use the volume provided: (1) the expiration data of any existing data set has passed, and (2) there are no password protected data sets on the volume. If either condition is not met, the system asks the operator to mount another volume.

BLP
: if you want to bypass label processing, BLP is useful if you want to use a blank tape or if you want to overwrite a seven-track tape that differs from your current parity or density specification. Your manager or supervisor should tell you if your system has the BLP facility. If it does not and you code BLP, the system assumes NL.

The type of label used for tape volumes is determined by your installation. Most installations use only one type of labels. Check with your manager or supervisor before you specify a label type other than the type used in your installation.

PASSWORD is the third positional subparameter. It tells the system that the data set you are creating cannot be used by another job step or job unless the operator can supply the system with the correct password. Password protected data sets must have standard labels, that is you must code the SL subparameter (or omit the second positional subparameter). For example:

```
//DD26 DD DSNAME=OPEN.SESAME,DISP=(,CATLG,DELETE),UNIT=2400,
//        VOLUME=SER=SECRET,LABEL=(,,PASSWORD),...
```

Before or after you create the data set you must tell the system programmer the name of your data set and the password. He will create an entry in a data set named PASSWORD containing those two items. Whenever a request is made for your data set the system will verify the password supplied by the operator against the entry in the PASSWORD data set.

The fourth positional subparameter (OUT) is used only if you coded your program in the assembler language or in FORTRAN. If you coded in the assembler language it allows you to override the specification of the OUTIN parameter in the OPEN macro instruction (for BSAM data sets only). If OUTIN is specified and you want the data set processed for output only code

```
LABEL=(,,,OUT)
```

If you are a FORTRAN user, OUT means that the set is to be processed for output only.

You can assign a retention period (length of time during which the data set cannot be changed or deleted without the operator's permission) to your nontemporary data set with either the RETPD=nnnn or EXPDT=yyddd subparameter. These subparameters are keyword subparameters. You must code the one you choose after the last positional subparameter you code. For example:

```
LABEL=(5,RETPD=34)
LABEL=(,SUL,EXPDT=69251)
LABEL=(3,,PASSWORD,RETPD=100)
LABEL=(,NSL,,OUT,EXPDT=7100)
LABEL=RETPD=170
```

The RETPD subparameter expresses the retention period in terms of the number of days you want the data set retained. The format of RETPD is:

```
RETPD=nnnn
```

Replace "nnnn" with a number from 1 to 9999.

The EXPDT subparameter states the expiration data of your data set. The format of EXPDT is:

```
EXPDT=yyddd
```

Replace "yyddd" with the 2-digit year number and the 3-digit day number. For example, January 1, 1971 is coded as

```
LABEL=EXPDT=71001
```

and February 1,1971 is coded as

```
LABEL=EXPDT=71032
```

(Many calendars indicate the day number for each day of the year.)

If neither RETPD or EXPDT is specified, a retention period of zero days is assumed; that is, your data set can be modified or deleted at anytime. Do not specify a retention period for temporary data sets, because they are always deleted at the end of the job step or the job.

Data Attributes

The DCB parameter allows you to specify data set attributes for your data set when your program is to be executed rather than when it is compiled. Any applicable attributes not specified in your program must be specified with the DCB parameter. However, in most cases, your compiler provides a default value for an attribute if you do not specify it in the program or in the DD statement. Other attributes are always given a fixed value by the compiler and you do not have to specify them at all. For example, you can select a buffering technique with the assembler, but all other IBM compilers select one for you when it is needed.

DCB: You can use the DCB parameter to directly specify the attributes of your data set or to copy those attributes specified in a DD statement for another data set.

The format of the DCB parameter for specifying the attributes is:

```
DCB=(list of attributes)
```

The attributes in the list are coded in the form of keyword subparameters separated by commas; for example,

```
DCB=(BLKSIZE=300,LRECL=100)
```

The valid subparameters that can be used with each compiler for magnetic tape data sets are shown in Table 9. Underscored items are those default values selected if you omit the subparameter. Default values are not shown where the attribute can be specified either in your program or in the DD statement. If values for a given subparameter are not shown, they are either specified in your source program or given a fixed value by the compiler. A glossary of DCB parameters is given in Appendix B. Code only those parameters that apply to your compiler as shown in Table 9.

Table 9. DCB Subparameters for Creating a Data Set on Magnetic Tape

DCB Subparameter	ALGOL	Assembler	COBOL E	COBOL F	ANS COBOL [4]	FORTRAN E	FORTRAN G & H	PL/I F
BFALN=		F or D [1,2]						
BFTEK=		S, E, or A (QSAM only) [1,2]						
BLKSIZE=	number of bytes	number of bytes [1]		number of bytes [1]	number of bytes [1]	number of bytes	number of bytes	number of bytes [1]
BUFL=		number of bytes [1]						
BUFNO=		number of buffers [1]	number of buffers [1]	number of buffers [1]	number of buffers [1]	1 or 2	1 or 2	number of buffers [1]
DEN=		0,1,2, or 3 [1,3]	0,1, or 2 [3]	0,1,2, or 3 [3]	0,1,2, or 3 [3]	0,1,2, or 3 [3]	0,1,2, or 3 [3]	0,1,2, or 3 [3]
EROPT=		ABE [1]	ABE	ABE	ABE			
HIARCHY=		0 or 1 [1]						
LRECL=	number of bytes	number of bytes [1]				number of bytes	number of bytes	number of bytes [1]
NCP=		number of channel programs (BSAM only) [1]						number of channel programs (BSAM only) [1]
OPTCD=		[C][T] [1]	[C]	[C]	[C]	[C]	[C]	[C] [1]
RECFM=	F[B][BS][A]	U[A/M] or V[B/S/BS][A/M] or F[B/S/BS][A/M] [1]				Formatted: U[A] V[B][A] or F[B][A/M] Unformatted: VS[B][A/M]	Formatted: U[A/M] or V[B][A/M] or F[B][A/M] Unformatted: VS[B][A/M]	U[A/M] or V[B/S/BS][A/M] or F[B/S/BS][A/M] [1]
TRTCH=		C,E,ET, or T [1]	C,E,ET, or T	C,E,ET, or T	C,E,ET, or T	C,E,ET, or T	C,E,ET, or T	C,E,ET, or T

[1] This function can be specified in your program rather than in the DD statement.

[2] For QSAM, you must specify both BFALN and BFTEK on the DD statement or omit both.

[3] If DEN is omitted, 2 is assumed for 7-track, 2 is assumed for 9-track (one density), and 3 is assumed for 9-track (dual density).

[4] American National Standard COBOL.

The format of the DCB parameter for copying the DCB parameter of a previous DD statement in your job is:

```
                  ⎧*.ddname                          ⎫
DCB=(             ⎪*.stepname.ddname                 ⎪[,list of attributes])
                  ⎩*.stepname.procstepname.ddname    ⎭
```

Replace "ddname" with the name of the DD statement whose DCB parameter you want to copy. Replace "stepname" with the name of the EXEC statement of the step that contains that DD statement. If the DD statement you want to copy is contained in the same job step, omit the stepname, i.e., DCB=*.ddname. The following example shows how to code the DCB parameter to copy the DCB parameter of a DD statement in previous step:

```
//STEP2 EXEC...
//DD1 DD...,DCB=(BLKSIZE=1600,LRECL=80)
.
.
.
//STEP4 EXEC...
//COPY DD...,DCB=*.STEP2.DD1
.
.
.
```

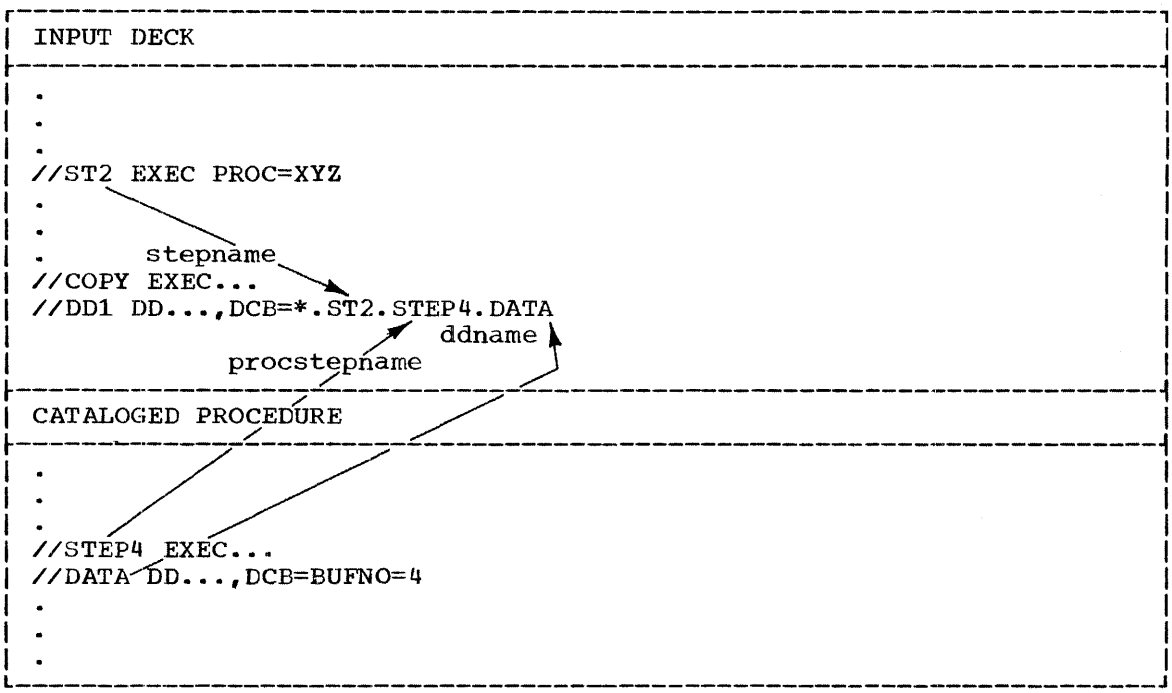
The following example shows how to code the DCB parameter of a previous DD statement in the same step:

```
.
.
.
//STEPD EXEC...
//DDA DD...,DCB=RECFM=F
.
.
.
//DDC DD...,DCB=*.DDA
.
.
.
```

If you want to copy the DCB parameter of a DD statement contained in a cataloged procedure step, you must give both the name of the job step that invokes the procedure and the procedure step name, i.e.,

```
DCB=*.stepname.procstepname.ddname
```

The first part of the following example shows your input deck; the second part shows a cataloged procedured called by the ST2 EXEC statement in your deck.



If you want to modify the DCB subparameters you are copying add the new subparameters to the reference. The subparameters you specify will override the corresponding copied subparameters. For example,

```
.
.
.
//OUT EXEC...
//TAPE1 DD...,DCB=(BLKSIZE=160,LRECL=80,OPTCD=C,DEN=1
//TAPE2 DD...,DCB=(*.TAPE1,DEN=2,RECFM=V)
.
.
.
```

Special Processing Options

There are two special processing options for data sets on magnetic tape:

1. You can request channel separation from other data sets in the same job step using either the SEP or the AFF parameter.

2. You can suppress I/O operations on your data set using the DUMMY parameter.

SEP: When two or more data sets are to be used in a job step, processing time may be shortened by requesting that the system transmit data sets over separate channels. For example, it would be faster to have your input data set and your output data set on separate channels than to have them on the same channel.

You can request channel separation for data sets in each job step using the SEP parameter. If possible, the system will honor this request. It may not always be possible to honor the request for separation because given devices may not be available for allocation when the job step is executed or because there may not be enough devices of a given type on different channels for all the data sets that request separation. If channel separation is not requested, the system will assign any available channel that has the device specified with UNIT parameter.

The format of the SEP parameter is:

```
SEP=(ddname,...)
```

Replace the terms "ddname" with the names of up to eight earlier DD statements in the same job step. The earlier DD statements can define any type of data set; new or existing, on magnetic tape or direct access, temporary or nontemporary, etc.

In the following example, you request that the data set defined by the FOUR DD statement be assigned to a channel other than the ones assigned to the data sets defined by the ONE and TWO DD statement. The data sets defined by ONE and TWO may or may not be on the same channel. The data set defined by THREE may or may not be on the same channel as any of the other three data sets.

```
//PROG1 EXEC PGM=PROCESS
//ONE DD UNIT=2400,VOLUME=PRIVATE,LABEL=(,NL)
//TWO DD DSNAME=SYSTEM.USER.ABC,DISP=OLD
//THREE DD DSNAME=HELP,DISP=(,KEEP),UNIT=2314,
//         VOLUME=SER=AFL,LABEL=RETPD=31
//FOUR DD DSNAME=END,DISP=OLD,UNIT=2400-2,
//         VOLUME=SER=TAPE75,SEP=(ONE,TWO)
```

Using the preceding example, if you want to request that the data set defined by the THREE DD statement be on a channel other than the one assigned to the data set defined by the TWO DD statement code:

```
//PROG1 EXEC PGM=PROCESS
//ONE DD UNIT=2400,VOLUME=PRIVATE,LABEL=(,NL)
//TWO DD DSNAME=SYSTEM.USER.ABC,DISP=OLD
//THREE DD DSNAME=HELP,DISP=(,KEEP),UNIT=2314,
//         VOLUME=SER=AFL,LABEL=RETPD=31,SEP=TWO
//FOUR DD DSNAME=END,DISP=OLD,UNIT=2400-2,
//         VOLUME=SER=TAPE75,SEP=(ONE,TWO)
```

As a result of this example, the data sets defined by ONE and TWO may or may not be on the same channel. The data set defined by THREE and FOUR may or may not be on the same channel, but neither one will be on the same channel as the data set defined by TWO. Also the data set defined by FOUR will not be on the same channel as the data set defined by ONE.

Remember that the request for channel separation made with the SEP parameter is only a suggestion for more efficient operation, and not a requirement. If your data set must be on separate channels specify unit addresses with the UNIT parameter. You can obtain a configuration chart with all the unit addresses from your manager or supervisor.

AFF: The AFF parameter provides a shortcut method of requesting channel separation. When two or more data sets in the same job step have the same channel separation requirements, you can code the SEP parameter for the first data set, and then request the same separation using the AFF parameter for the later data sets. The AFF parameter tells the system that you want the data set defined in this DD statement to have the same channel separation as the data set defined in the named DD statement.

The format of the AFF parameter is:

```
AFF=ddname
```

Replace "ddname" with the name of the earlier DD statement that requests the desired channel separations. This data set and the earlier data set may or may not be on the same channel.

In the following example, the data sets defined by DD1 and DD2 may or may not be on the same channel. The data sets defined by DD3 and DD5 may or may not be on the same channel, but neither one will be on the same channel or channels as the data sets defined by DD1 or DD2. The data set defined by DD4 will not be on the same channel as the data set defined by DD3, but it may be on the same channel as the data sets defined by DD1, DD2, or DD5.

```
//STEP   EXEC   PGM=METHOD2
//DD1    DD     ...
//DD2    DD     ...
//DD3    DD     SEP=(DD1,DD2),...
//DD4    DD     SEP=DD3,...
//DD5    DD     AFF=DD3,...
```

The requests for channel separation and affinity are only suggestions to the system. They will be honored if the appropriate devices are available.

DUMMY: The DUMMY parameter allows you to bypass I/O operations and data set disposition. When your processing programs ask to write the dummy data set, the write request is recognized, but no data is transmitted. This facility is particularly useful when you are debugging your program because it saves you processing time. For example, you can suppress the writing of a data set until you are sure your program is going to produce meaningful output.

The DUMMY parameter is a positional parameter. That means that it must be coded first in the operand field. For example:

```
//OUTPUT DD DUMMY,UNIT=2400-1,
//DD7 DD DUMMY,DSNAME=&&JOE,DISP=(,PASS),UNIT=TAPE7,
//       LABEL=(,SUL),DCB=TRTCH=ET
```

If you wish, you can omit all other parameters required to define your data set except DCB. (If you were not going to write the DCB parameter you do not have to write it.) For example, the two DD statements shown above could be coded as follows:

```
//OUTPUT DD DUMMY
//DD7 DD DUMMY,DCB=TRTCH=ET
```

When you are ready to perform I/O operations on your data set, simply omit the DUMMY parameter and code all parameters required to write the data set. For example,

```
//OUTPUT DD UNIT=2400-1,
//DD7 DD DSNAME=&&JOE,DISP=(,PASS),UNIT=TAPE7,
//       LABEL=(,SUL),DCB=TRTCH=ET
```

Direct Access Devices

You can create five different kinds of data set on direct access devices:

- Sequential data sets.
- Direct data sets.
- Partitioned data sets.
- Generation data sets.
- Indexed sequential data sets.

The parameters shown in Table 6 are used to create those data sets. The definition of an indexed sequential data set may require up to three DD statements, and is discussed in Appendix A. The remaining four kinds of data sets are defined with only one DD statement. The way you use certain parameters indicates which kind of data set you are defining. These differences are summarized below and will be pointed out again when the parameters are discussed in the remainder of this topic.

- Sequential data set -- You use basically the same parameters used to define a data set on magnetic tape and you must also allocate space for the data set using one of the following parameters: SPACE, SPLIT, or SUBALLOC.

- Direct data set -- You use the same parameters as for a sequential data set, with the exception of the SPLIT parameter. If not specified in your program, you must write DSORG=DA or DSORG=DAU in the DCB parameter.

- Partitioned data set -- You use the same parameters as for a sequential data set, with the exception of the SPLIT parameter. You must code the number of directory blocks in the SPACE or SUBALLOC parameters. You can define the first member of the data set at the same time you allocate the partitioned data set. To do this use a special form of the DSNAME parameter, i.e.,

  ```
  DSNAME=dsname(membername)
  ```

  To learn how to add more members to a partitioned data set, refer to the section "Extending an Existing Data Set."

- Generation data set -- You use the same parameters as for a sequential, partitioned, direct, or indexed sequential (if the data set is defined on one DD statement) data set. You must use a special form of the DSNAME parameter, i.e.,

  ```
  DSNAME=groupname(+number)
  ```

  Generation data sets must be cataloged, i.e.,

  ```
  DISP=(,CATLG)
  ```

For further information on the different types of data set refer to "Section 3: Data Management" in IBM System/360 Operating System: Concepts and Facilities. Programs written in ALGOL or FORTRAN H cannot use direct data sets. Partitioned data sets can only be used by programs written in the assembler language. (Programs written in COBOL or FORTRAN can use partitioned data sets in a restricted way. For further information refer to the language Programmer's Guide.)

This section is summarized in Table 66 of Appendix D.

Data Set Information

The DSNAME and DISP parameter are used to specify which type of data set you want to create. There are two types of data sets: temporary and nontemporary. A temporary data set is one that will be used for the duration of the jobstep or the job only . A nontemporary data set is one that can be used not only by the jobsteps in your job, but by later jobs as well. There are three types of nontemporary data sets: cataloged data sets, kept (or noncataloged) data sets, and members of generation data groups.

The following is a summary of the types of data sets that can be created on direct access devices:

1. Temporary

   a. For the duration of the job (sequential, direct or partitioned).
   b. For the duration of the jobstep (sequential, direct or partitioned).

2. Nontemporary

   a. Cataloged (sequential, direct, or partitioned).
   b. Kept (or noncataloged) (sequential, direct or partitioned).
   c. Member of a generation data group.

The formats of the DSNAME and DISP parameters for each type of data set are described below.

Temporary -- For the Duration of the Job: The formats of the DSNAME and DISP parameters are:

```
DSNAME={&&name                 },DISP=(NEW,PASS,DELETE)
       {&name                  }
       {&&name(membername)     }
       {&name(membername)      }
```

In the DSNAME parameter, replace the term "name" with any name not used by another temporary data set in the job. Each name consists of one to eight alphameric or national (@,$,#,) characters. However, the first character of the name must be a letter or national character. If you are creating the first member of a partitioned data set, add the member name after the temporary name. The member name is enclosed in parentheses and consists of one to eight alphameric or national characters. The first character must be a letter or national character.

The system generates a name for the temporary data set, which begins with SYS and includes the jobname, the temporary name assigned in the DSNAME parameter, and other identifying characters. You can retrieve this data set later in the job by coding

```
   DSNAME=&&name
   DSNAME=&name
   DSNAME=&&name(membername)
or
   DSNAME=&name(membername)
```

in a DD statement, using the same name, or

```
   DSNAME=*.stepname.ddname
```


A double ampersand should be coded preceding the temporary name you assign to a data set. However, a name preceded by a single ampersand is treated as a temporary data set name, as long as no value is assigned to it either on an EXEC statement invoking a procedure, or a PROC statement within a procedure. If a value is assigned to it by one of these means, it is treated as a symbolic parameter. (Symbolic parameters are discussed in the section "The PROC Statement" and in "Part III: Cataloged Procedures.")

In the DISP parameter, the NEW subparameter indicates that you are creating the data set. You do not have to code NEW as long as you indicate its absence by a comma. The PASS subparameter indicates that the temporary data set can be used by later steps in the job. The DELETE subparameter indicates that this data set is to be deleted if the job step abnormally terminates (ABEND). You need not code DELETE because it is always assumed by the system for temporary data sets.

For example, the parameters DSNAME and DISP in the following DD statement indicate that the temporary data set named AREA can be used by later steps in the job.

```
//DD1 DD DSNAME=&&AREA,DISP=(,PASS),...
```

In the following example the DSNAME parameter defines the first member of a temporary data set named PDS.

```
//DDA DD DSNAME=&&PDS(ONE),DISP=(,PASS),...
```

Temporary -- For the Duration of the Job Step: Both the DSNAME and DISP parameters can be omitted to indicate a temporary data set which will be deleted at the end of the job step. You can, if you wish, code the following:

```
DSNAME={&&name                 }[,DISP=(NEW,DELETE,DELETE)]
       {&name                  }
       {&&name(membername)     }
       {&name(membername)      }
```

The temporary name in the DSNAME parameter is coded as described for temporary data sets for the duration of the job. All subparameters of

the DISP parameter are assumed if omitted. For example, the three following DD statements define the same data set:

```
//DD2 DD DSNAME=&&AREA,DISP=(NEW,DELETE,DELETE),...
```

or

```
//DD2 DD DSNAME=&&AREA,...
```

or

```
//DD2 DD...
```

Note that the location parameters must still be coded in those statements.

Nontemporary -- Cataloged: The formats of the DSNAME and DISP parameters are:

```
DSNAME= {dsname,            },DISP=(NEW,CATLG[,CATLG ])
        {dsname(membername)}                 [,DELETE]
                                             [,KEEP  ]
```

In the DSNAME parameter, replace the term "dsname" with the qualified name you want to give to the data set. A qualified name is made up of several names that correspond to index levels in the catalog; each 1-to-8 character name is separated by a period. Each name must begin with a letter or national (@,$,#) character. Any letter, number, nation be used to complete each name. A qualified name can consist of 1 through 44 characters (including periods). For example,

```
DSNAME=AB.@3.M.$7.R2579AB2
```

Each level of qualification must already exist as an index in the system catalog before you can request the system to catalog the data set. An index level is created using the IEHPROGM utility program. Once the indexes are established, the data set can be cataloged. If the name has no qualifiers (called an unqualified name), the system will create the index entry for you. For example,

```
DSNAME=C174B
```
or
```
DSNAME=D
```

If you are creating the first member of a partitioned data set, add the member name after the data set name. The member name is enclosed in parentheses and consists of one to eight alphameric or national characters. For example,

```
DSNAME=DATA.FILE(G794)
```
or
```
DSNAME=ALPHA(MW)
```

In the DISP parameter, the NEW subparameter indicates that you are creating the data set. You do not have to code NEW as long as you indicate its absence by a comma. The CATLG subparameter indicates that the data set is to be cataloged. The third subparameter tells the system what is to be done with the data set if the step abnormally terminates (ABEND). If the data set name is either qualified or unqualified, you can specify CATLG or DELETE. CATLG is assumed if you omit the third subparameter. If the data set name is unqualified, you

can specify KEEP as the third subparameter. KEEP tells the system that if the step ABENDs you want the data set kept intact (but not cataloged) until a later job requests that the data set be deleted or until the expiration date has passed. (You can specify a retention period or expiration date in the LABEL parameter.)

In the following example, a new partitioned data set named DATE.FILE is to be cataloged. Its first member, JAN, is being created at the same time.

```
//DD20 DD DSNAME=DATE.FILE(JAN),DISP=(,CATLG),...
```

If you wish to include the first member at a later time, simply code:

```
//DD21 DD DSNAME=DATE.FILE,DISP=(,CATLG),...
```

In the following example, a new data set named SYSTEM.FILE1 is to be cataloged. If the step abnormally terminates it will also be cataloged.

```
//DD3 DD DSNAME=SYSTEM.FILE1,DISP=(,CATLG),...
```

In the following example, a new data set named AREA#5 is to be cataloged. If the step abnormally terminates, it will be kept.

```
//DD4 DD DSNAME=AREA#5,DISP=(,CATLG,KEEP),...
```


Nontemporary -- Kept: The formats of the DSNAME and DISP parameters are:

```
DSNAME={dsname              },DISP=(NEW,KEEP[,KEEP  ])
       {dsname(membername)}             [,DELETE]
                                        [,CATLG ]
```

In the DSNAME parameter, replace the term "dsname" with the unqualified name you want to give to the data set. The name can contain from 1 to 8 characters. The first character must be a letter or a national character (@,$,#). Any letter, number, national character, the hyphen and the +0(12-0 multipunch) can be used to complete the name. For example,

```
DSNAME=@A#753
```

If you are creating the first member of a partitioned data set, add the member name after the data set name. The member name is enclosed in parentheses and consists of one to eight alphameric or national characters. The first character must be a letter or nation character. For example,

```
DSNAME=STORE($FILE)
```

In the DISP parameter, the NEW subparameter indicates that you are creating the data set. You do not have to code NEW as long as you indicate its absence by a comma. The KEEP subparameter indicates that the data set is to be kept intact (but not cataloged) until a later job step or job requests that the data set be deleted or until the expiration date has passed. (You can specify a retention period or expiration date in the LABEL parameter.) The third subparameter tells the system what is to be done with the data set if the step abnormally terminates (ABEND). You can specify KEEP, DELETE or CATLG. KEEP is assumed if you omit the third subparameter.

In the following example, a new data set named RTW is to be kept. If the step ABENDs, it will also be kept.

```
//DDB DD DSNAME=RTW,DISP=(,KEEP),...
```

In the following example a new partitioned data set named RRC is to be kept. Its first member, ACS, is being created at the same time.

```
//DDB DD DSNAME=RRC(ACS),DISP=(,KEEP),...
```

In the following example, a new data set named TWO is to be kept. If the step ABENDs it will be cataloged.

```
//DD6 DD DSNAME=TWO,DISP=(,KEEP,CATLG),...
```

Notes:

- You can give a disposition of PASS to a new data set with an unqualified name. For example,

  ```
  DSNAME=DATA,DISP=(,PASS)
  ```

  Such a data set can become a temporary, cataloged, or kept data set according to the disposition you give to it in a later DD statement. See "Passed Data Sets" in "Retrieving an Existing Data Set." If the data set is not referred to by a later DD statement in the job, it is deleted at the end of the job.

- You can use special characters as part of the qualified data set name if you enclose the name in apostrophes. For example,

  ```
  DSNAME='AB=750'
  ```

  If the special character is an apostrophe, you must code it as two consectutive apostrophes. For example, code NAT'L as

  ```
  DSNAME='NAT''L'
  ```

  Do not use special characters if the data set is to be cataloged or if it contains a membername.

Nontemporary -- Member Generation Data Group: A cataloged data set that is periodically processed can be grouped with its earlier generations to form a named generation data group. The entire group has a generation group name. Each member of the group can be addressed by a simple generation number. The last generation produced before the start of your job has a generation number of 0. The generation produced before the last one is -1. The generation you are going to produce in your job is +1. After your job ends, the generation you produced automatically becomes generation 0. If you produce more than one generation in your job, you must name them +1, +2, +3, etc. After your job, the highest generation number you produced, say +3, becomes generation 0, +2 becomes -1 and +1 becomes -2.

For further information on creating generation data groups, refer to Appendix D of the publication IBM System/360 Operating System Job Control language Reference. The formats of the DSNAME and DISP parameters for creating a new member of a generation data group are:

```
DSNAME=groupname(+number),DISP=(NEW,CATLG[,CATLG ])
                                       [,DELETE]
```

In the DSNAME parameter, replace the term "groupname" with the name of the generation data group. The name can contain from one to eight characters. The first character must be a letter or national character (@,$,#). Any letter, number, national character, the hyphen, and the +0 (12-0 multipunch) can be used to complete the name. The groupname can be a qualified name. The qualified name can contain up to 35 characters (including periods). Replace the term "number" with the number of the generation you are creating. If you are only creating one generation in your job, this number will be 1. For example,

```
   DSNAME=MASTER(+1)
or
   DSNAME=A.B(+1)
```

If you are creating more than one generation in your job, number each generation consecutively. For example, DSNAME=MASTER(+1), DSNAME=MASTER(+2), etc.

In the DISP parameter, the NEW subparameter indicates that you are creating the data set. You do not have to code NEW as long as you indicate its absence by a comma. The CATLG subparameter indicates that the new generation is to be cataloged. (This subparameter is required.) The third subparameter tells the system what is to be done with the data set if the step abnormally terminates (ABEND). You can specify CATLG or DELETE. CATLG is assumed if you omit the third subparameter.

In the following example, two new generations are added to the DATE.FILE generation data group. If the step abnormally terminates the new generations will be deleted.

```
//STEP EXEC PGM=...
.
.
.
//DD7 DD DSNAME=DATE.FILE(+1),DISP=(,CATLG,DELETE),...
//DD8 DD DSNAME=DATE.FILE(+2),DISP=(,CATLG,DELETE),...
.
.
.
```

After your job is successfully completed DATE.FILE(+2) becomes DATE.FILE(0), and DATE.FILE(+1) becomes DATE.FILE(-1).

Location of the Data Set

The UNIT, VOLUME and LABEL parameters are used to describe the location of your data set. The UNIT parameter indicates the kind of direct access device you want, and, if your data set is to occupy more than one volume, it indicates how many units are needed. The VOLUME parameter lets you request a public volume (or volumes), or a private volume (or volumes). You can request a specific volume or you can let the system assign you a volume. The LABEL parameter describes the tape label of the volume used for your data set. You must allocate space to a sequential data set using either the SPACE, SPLIT or SUBALLOC parameter; to a partitioned or direct data set using either the SPACE or SUBALLOC parameter; and to a generation data set using the SPACE parameter.

UNIT: The format of the UNIT parameter is:

```
                  ⎧unit address⎫ ⎡,unitcount⎤
UNIT=(            ⎨device type  ⎬ ⎢,P       ⎥ [,SEP=(ddname,...)])
                  ⎩groupname    ⎭ ⎣,        ⎦
```

The first positional subparameter identifies the direct access device you want to use for you data set by its address, or unit name, or group name.

unit address
is the actual machine address of the direct access device. For example, UNIT=190. You should not specify the address unless you are sure you want it. Do not specify the address if you are going to need more than one unit for the data set. (Multiple units are requested with the second subparameter of the UNIT parameter.) To request a specific bin on a specific 2321, you should write

UNIT=address/bin

where "bin" is a number from 0 to 9. For example,

UNIT=293/7

indicates that you want bin 7 of the 2321 unit located at address 293. If you specify

UNIT=293

you are requesting any one of the available bins on that device.

device type
corresponds to the type of device. Coding the device type provides you with a certain degree of device independence in that your data set may be placed in any number of devices of the same type. For example, if you code UNIT=2311, your data set will be placed on any 2311 Disk Storage Drive. The following device types can be specified.

Device Type	Description
2301	2301 Drum Storage Unit
2302	2302 Disk Storage Drive
2303	2303 Drum Storage Unit any 2311 Disk Storage Drive
2311	Any 2311 Disk Storage Drive
2314	2314 Storage Facility
2321	Any bin mounted on a 2321 data cell drive

group name
is the name of a collection of devices, selected by your installation during system generation. For example, your installation might select the name DISK for all disk drives in the configuration. If you do not care which drive is used for your data set, you should code UNIT=DISK. Your manager or supervisor should tell you which group names were generated for your installation.

The second positional subparameter is used only if your data set will occupy more than one volume and if you want more than one of these volumes to be mounted at the same time. This subparameter indicates how many drives are to be used for mounting your data set's volumes.

unit count
indicates how many units you want assigned to the data set. You can specify a maximum of 59 units per DD statement. Make sure that your system has at least the number of units you specify. Otherwise an error will result. If you specify fewer units than the number of volumes in your data set, only the same number of volumes as there are units can be mounted at the same time. (If you request fewer units than volumes code the PRIVATE subparameter in the VOLUME parameter.) If you specify the same number of units as there are volumes, all volumes can be mounted at the same time.

(If you think your data set may use more volumes than you expect, you should indicate the maximum number of volumes that can be used with the volume count subparameter of the VOLUME parameter. Do not request any more units than the maximum number of volumes you specify.) For example, if you code UNIT=(2301,2), your data set will be assigned two 2301 drum storage units.

P
specifies parallel mounting. When you request parallel mounting the system counts the number of serial numbers specified with the VOLUME parameter. If, in addition to indicating specific serial numbers in the VOLUME parameter, you also indicate a maximum number of volumes that can be used by the data set, the number of units assigned is the number of volumes that can be used.

SEP
is the last positional subparameter. It is used when you want a different direct access device assigned to your data set from the ones assigned to previous data sets in the same job step. This parameter is used only when a previous data set made a nonspecific volume request for the same type of device, because the system can assign more than one data set to the same direct access device. (Note that you would be contradicting yourself if you made specific requests to place two or more data sets on the same volume and then requested unit separation for those data sets.) To identify the data sets that should not be assigned the same device as this data set, follow SEP= by a list of up to eight ddnames of the DD statements that define these data sets. The listed DD statements must precede this statement and must be contained in the same job step. The list of ddnames must be enclosed in parentheses unless there is only one ddname. If one of the listed DD statements defines a dummy data set, the system ignores the unit separation request for that data set.

In the following example the DDAA DD statement defines a temporary data set that needs 10 tracks on a volume that can be mounted on a 2311. The DDBB DD statement defines another temporary data set that needs 50 tracks on a volume that can also be mounted on a 2311. The DDCC DD statement also defines a temporary data set that needs 20 tracks on the same type of volume but requests that the volume assigned to it be mounted on a different 2311 unit than those assigned to the first two data sets. As a result of these DD statements the first two data sets may or may not use the same 2311, but the third data set will not use the same 2311 as either of the first two data sets.

```
//STEPB   EXEC...
//DDAA    DD  UNIT=2311,SPACE=(TRK,10)
//DDBB    DD  UNIT=2311,SPACE=(TRK,50)
//DDCC    DD  UNIT=(2311,SEP=(DDAA,DDBB)),SPACE=(TRK,20)
```

In PCP the system ignores a unit separation request if the request conflicts with another unit separation request or if sufficient devices

are not available to satisfy the request. In MFT and MVT the system issues a message to the operator if a unit separation request cannot be satisfied. The operator decides if the system should wait for devices to become available, if the unit separation request should be ignored, or if the job should be canceled.

Do not confuse the SEP subparameter of the UNIT parameter with the SEP parameter. The former requests unit separation while the latter requests channel separation.

Note: You need not code the UNIT parameter when:

- The data set is to use the same volume assigned to an earlier data set, that is, if you specify VOLUME=REF.
- The data set is to share space or cylinders with an earlier data set, that is, if you specify SUBALLOC or SPLIT.

In both cases, you can use the UNIT parameter if you want more units assigned.

VOLUME: The VOLUME parameter lets you request a specific volume for your data set or lets the system assign your data set a suitable volume (nonspecific request). When you make a nonspecific request the system will look for a volume that both fits the device specified with the UNIT parameter and has the amount of space you specified with the SPACE, SPLIT, or SUBALLOC parameter available, and will then allocate that amount of space to your data set. When you make a specific request, you must make sure that the volume you requested has enough space for your data set. If it does not have enough space the job step will ABEND.

You can request a public or a private volume with either a specific or nonspecific request. A volume is designated public or private by the installation. A public volume is one that can be allocated to a temporary data set when a nonspecific volume request is made and PRIVATE is not coded in the VOLUME parameter. A private volume is one that cannot be allocated to a temporary data set when a nonspecific volume request is made and PRIVATE is not coded in the VOLUME parameter. The key word in the preceding definitions is "nonspecific." If you make a specific request you can get any volume - public or private - allocated to your data set provided there is enough space on that volume. However, if you make a specfic request for a public volume, remember that temporary data sets that make nonspecific requests can be allocated on the same volume. (If you want more information on the different types of volume and how they are allocated read the description of the VOLUME parameter in IBM System/360 Operating System: Job Control Language Reference. A distinction is made in that publication between public and storage volumes; for the purposes of this publication both are called public volumes.)

The following is a summary of the types of volume requests that can be made for new data sets on direct access devices:

1. Nonspecific request
   a. Public volume
   b. Private volume
2. Specific request
   a. Public volume
   b. Private volume

Before discussing the four types of requests in detail, a word must be said on multivolume data sets on direct access devices. Normally, all direct access data sets are allocated on only one volume. This is because the amount of space (primary quantity) you specify with the SPACE, SPLIT, or SUBALLOC parameter is allocated entirely on one volume. There is, however, a way of extending your data set to more volumes. You can allocate a "secondary quantity" to your data set in the SPACE, SPLIT, or SUBALLOC parameter. When the space you requested as "primary quantity" is used up, the system allocates the amount you request as "secondary quantity" to your data set. If the "secondary quantity" is used up, the system again allocates your data set another "secondary quantity", and so on. The system will try to allocate the "secondary quantity" on the same volume as the "primary quantity". However, if there is not enough space available on the first volume the system will allocate the "secondary quantity" on another volume provided you request more than one volume with the VOLUME parameter. If you do not request more volumes or if the data set exceeds the volumes you request, the job step abnormally terminates (ABEND). You request more volumes either by specifying more than one serial number with the SER subparameter or by using the volcount subparameter, or both. There is no point in requesting more than one volume if you do not specify a "secondary quantity" because the "primary quantity" must be completely allocated on one volume. Also, you do not have to request more than one volume if you know that any "secondary quantity" allocated will fit in the same volume as the "primary quantity".

Partitioned data sets must be contained on only one volume.

Nonspecific Request -- Public Volume: The format of the VOLUME parameter depends on whether you need one volume or more than one volume.

1. If your data set needs space on only one public volume, omit the VOLUME parameter. For example, if you have a temporary data set that needs 10 tracks on a 2314 volume, write:

   ```
   //DASD1 DD UNIT=2314,SPACE=(TRK,10)
   ```

   If you have a data set named AREA that is to be cataloged and you want 5 cylinders on a 2311 volume, write:

   ```
   //DASD2 DD DSNAME=AREA,DISP=(,CATLG),UNIT=2311,SPACE=(CYL,5)
   ```

2. If your data set may require more than one volume, code:

   ```
   VOLUME=(,,,volcount)
   ```

   Replace the term "volcount" with the number of volumes required for your data set. You should request the same number of devices with the "unit count" subparameter of the UNIT parameter as you request volumes. If the number of volumes requested exceeds the number of devices, you should code PRIVATE as described in "Nonspecific Request/Private Volume."

   For example, if you have a temporary data set named &&DATA that is to be passed to other steps in your job and you are requesting 300

tracks of a 2311 as primary quantity and 100 tracks as secondary quantity, code:

```
//DASD3 DD DSNAME=&&DATA,DISP=(,PASS),UNIT=(2311,2),
//         VOLUME=(,,,2),SPACE=(TRK,(300,100))
```

If you expect the same data set (&&DATA) to need a maximum of four volumes, code:

```
//DASD4 DD DSNAME=&&DATA,DISP=(,PASS),UNIT=(2311,4),
//         VOLUME=(,,,4),SPACE=(TRK,(300,100))
```

Nonspecific Request -- Private Volume: If you make a nonspecific request for a private volume the system assigns an available volume to you. Later data sets that make a nonspecific request will not be assigned to this volume.

The format of the VOLUME parameter for making a nonspecific volume request for a private volume is:

```
VOLUME=(PRIVATE[,RETAIN][,,volcount])
        [,    ]
```

The PRIVATE subparameter is required. The RETAIN subparameter means that the volume will not be demounted at the end of the step. (RETAIN has the same effect on volume mounting as coding PASS in the DISP parameter.) Private volumes are demounted at the end of the step unless it is an unremovable volume, such as a 2301, or unless it has been designated by your installation to be a permanently resident volume, for example, the system residence volume. It is recommended that you use the RETAIN subparameter whenever you want the volume to remain mounted at the end of the step.

For example, if you want 20 tracks of a nonspecific private 2314 volume for a temporary data set, code:

```
//DASD4 DD UNIT=2314,VOLUME=PRIVATE,SPACE=(TRK,20)
```

If you want five 100-byte blocks allocated to a data set named JOE.DOE, and you want the volume to remain mounted at the end of the step, code:

```
//DASD5 DD DSNAME=JOE.DOE,DISP=(,CATLG),UNIT=DISK,
//         VOLUME=(PRIVATE,RETAIN),SPACE=(100,5)
```

If your data set needs more than one volume replace the term "volcount" with the number of volumes needed. If the number of volumes is equal to the number of devices requested in the unit parameter all volumes are mounted at the same time. When the number of volumes is greater than the number of devices requested, and all the mounted volumes are used up, the system demounts one of the volumes and then mounts another volume in its place so that processing can continue. If you specify RETAIN, the volumes mounted at the end of the step remain mounted.

For example, if a temporary data set has a primary allocation of 1500 tracks and secondary allocation of 1500 tracks and you think it may need up to five private volumes to be mounted on two 2311 drives, code

```
//DASD6 DD UNIT=(2311,2),VOLUME=(PRIVATE,,,5),SPACE=(TRK,(1500,1500))
```

Using the preceding example, if you want the last two volumes used to remain mounted at the end of the step, code:

```
//DASD7 DD UNIT=(2311,2),VOLUME=(PRIVATE,RETAIN,,5),
//         SPACE=(TRK,(1500,1500))
```

Note: If you specify KEEP for your nontemporary data set, ask the machine operator to tell you the serial number of the volume assigned to your data set. You need the serial number to retrieve your data set in later jobs.

Specific Request -- Public Volume: You can request specific volumes by either stating the serial number of the volumes you want, or by requesting the same volume used by an earlier data set.

1. The format of the VOLUME parameter for requesting volumes by their serial numbers is :

```
VOLUME=([,,,volcount,]SER=(serial,...))
```

Replace the term "volcount" with the maximum number of volumes the data set can use. You should only use this subparameter if you think your data set may use more volumes than those named in the SER subparameter. Note that "volcount" is the maximum number of volumes that can be used and not additional volumes. For example, if you name three volumes in the SER subparameter, but you think you may need two more volumes, replace "volcount" with 5. In this case, you are in effect making a specific request for three volumes and a nonspecific request for two volumes. You should request the same number of devices with the "unit count" subparameter of the UNIT parameter as you request volumes. If the number of volumes requested exceeds the number of devices, you must code PRIVATE as described in "Specific Request -- Private Volume".

The SER subparameter indicates that the serial numbers of the volumes you want for your data set will allow. . Replace the term "serial" with the 1-to-6 character serial number associated with the volume. If the volume serial number is not 6 characters, it will be padded with trailing blanks by the system. The volume serial number can contain any alphameric and national characters and the hyphen. If it is necessary to include special characters, other than a hyphen, in the volume serial number, enclose it is apostrophes, for example,

```
VOLUME=SER=(54AB2,'6/10/8')
```

If only one volume is involved you need not code the parentheses, for example,

```
VOL=SER=ABCDEF
```

In the following example, two cylinders of a 2314 volume named 231405 are requested for a temporary data set:

```
//DASD10 DD UNIT=2314,VOLUME=SER=231405,SPACE=(CYL,2)
```

In the following example you request that volumes ABC111 and ABC222 be assigned to the FILE.ONE data set. Two direct access devices of the group named DISK will be used. The data set has a primary allocation of 1000 tracks and a secondary allocation of 500 tracks.

```
//DASD8 DD DSNAME=FILE.ONE,DISP=(,CATLG),UNIT=(DISK,2),
//         VOLUME=SER=(ABC111,ABC222),SPACE=(TRK,(1000,500))
```

In the following example, the same data set, FILE.ONE is defined, but the possibliity that it may need one extra volume is indicated. In this case, three devices must be assigned.

```
//DASD9 DD DSNAME=FILE.ONE,DISP=(,CATLG),UNIT=(DISK,3),
//         VOLUME=(,,,3,SER=(ABC111,ABC222)),SPACE=(TRK,(1000,500))
```

In the following example, you request that volumes 125, 769 and 842 be assigned to the BETA data set. All three volumes are to be mounted in parallel on units named DASD. A primary allocation of 2000 tracks and a secondary allocation of 500 tracks are made

```
//DASD11 DD DSNAME=BETA,DISP=(,KEEP),UNIT=(DASD,P),
//          VOLUME=SER=(125,769,842),SPACE=(TRK,(2000,500))
```

Notes:

- Do not request parallel mounting when you code the "volcount" subparameter unless you code PRIVATE.
- When using various typewriter heads or printer chains, difficulties in volume serial number recognition may arise if you use other than alphameric characters.
- SCRTCH should not be used as a volume serial number, because it is used to notify the operator to mount a non-specific volume.

2. The format of the VOLUME parameter for requesting volumes used by other data sets is:

```
VOLUME=([,,,volcount,]REF={dsname                              })
                           {*.ddname                           }
                           {*.stepname.ddname                  }
                           {*.stepname.procstepname.ddname     }
```

The REF subparameter identifies the volumes assigned to an earlier data set. If the earlier data set resides on more than one volume, all the volumes are assigned to your data set. If your data set needs more volumes, you should indicate so with the "volcount" subparameter. Replace "volcount" with the maximum number of volumes that can be used for your data set. When you use the REF subparameter, the system obtains UNIT information from the definition of the previous data set. However, if you are specifying additional volumes, you must use the UNIT parameter to specify the same number of units. For example,

```
VOLUME=(,,,5,REF=ABC.DATA),UNIT=(,5)
```

If you specify additional volumes but do not want to specify additional units, you must code PRIVATE as described in "Specific Request -- Private Volume."

You can use the REF subparameter to identify the earlier data set by its name or by making a backward reference. You can identify the data set by its name only if it is a nontemporary cataloged data set or a nontemporary passed data set. If one of these conditions is met, replace the term "dsname" with the data set's name. The data set's name cannot contain special characters, except for periods in the case of qualified names. For example,

```
VOLUME=REF=AB.CD.EF
```

If the data set is not cataloged or passed, or if it has been assigned a temporary name, you can refer to it by making a backward reference to the DD statement that defined it. (This DD statement must be an earlier DD statement in your job.) Replace "ddname" with the name of the DD statement where the earlier data set is defined. Replace "stepname" with the name of the EXEC statement of the step that has the earlier DD statement. If the earlier DD statement is contained in the same job step, omit the stepname, i.e.,

```
VOLUME=REF=*.ddname
```

The following example shows how to code the REF subparameter to use the volume specified in a DD statement in a previous step.

```
//STEP5 EXEC...
//DD1 DD UNIT=2314,VOLUME=SER=RFS167
.
.
.
//STEP7 EXEC...
//DDA DD VOLUME=REF=*.STEP5.DD1,...
.
.
.
```

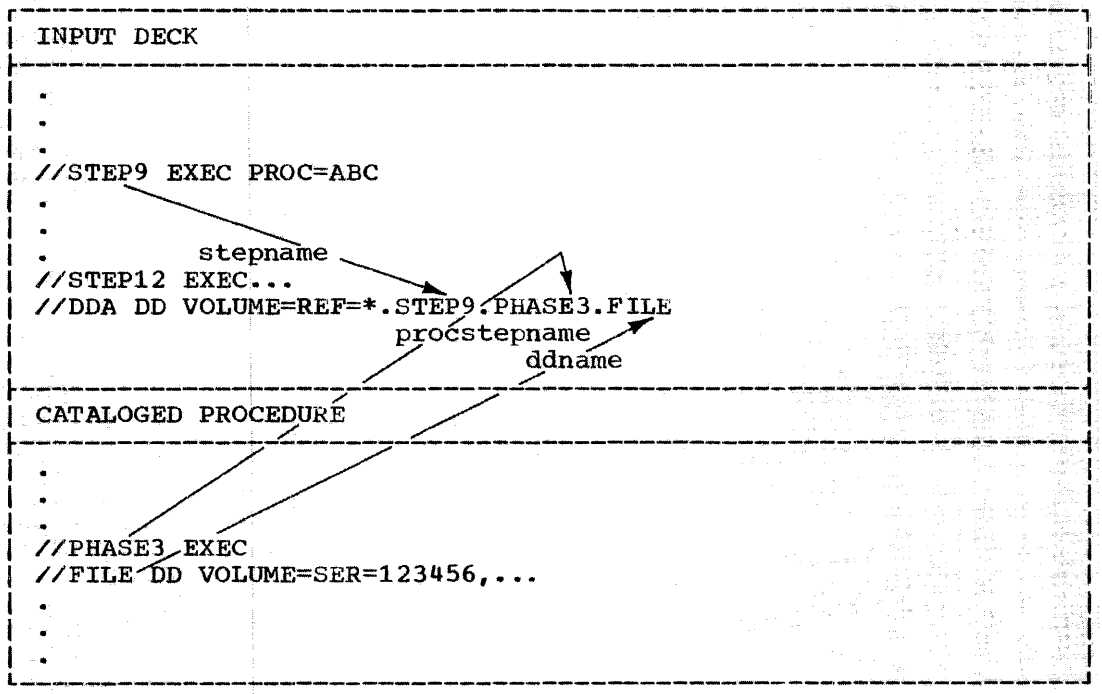
The following example shows how to code the REF subparameter to use the volume specified in a DD statement in a previous step. In addition to the volumes used by the previous data set, the new data set may use three more volumes. The UNIT parameter is used to override the number of units assigned in the first DD statement.

```
//STEPC EXEC...
//DD1 DD VOLUME=SER=(123456,789012),UNIT=(2311,2),...
.
.
.
//DD5 DD VOLUME=(,,,5,REF=*.DD1),UNIT=(,5),...
.
.
.
```

When the earlier DD statement is contained in a cataloged procedure step, you must give both the name of the job step that invokes the procedure and the procedure step name, i.e.,

```
VOLUME=REF=*.stepname.procstepname.ddname
```

The first part of the following example shows your input deck; the second part shows a cataloged procedure called by the STEP9 EXEC statement in your job.



If you want to use the volume assigned to a kept data set defined in another job, you must use VOLUME=SER.

Specific Request -- Private Volume: You can request specific volumes by either stating the serial number of the volumes you want, or by requesting the same volume used by an earlier data set.

1. The format of the VOLUME parameter for requesting volumes by their serial numbers is:

```
VOLUME=(PRIVATE[,RETAIN][,,volcount,]SER=(serial,...))
               [,      ]
```

The PRIVATE subparameter is required. The RETAIN subparameter means that the volume will not be demounted at the end of the step. (RETAIN has the same effect on volume mounting as coding PASS in the DISP parameter.) Private volumes are demounted at the end of the step unless it is an unremovable volume, such as a 2301, or unless it has been designated by your installation to be a permanently resident volume, for example, the system residence volume. It is recommended that you use the RETAIN subparameter whenever you want the volume to remain mounted at the end of the step.

For example, if you want 10 cylinders of a private 2314 volume named SYS333 for a temporary data set, code:

```
//DASD14 DD UNIT=2314,VOLUME=(PRIVATE,SER=SYS333),SPACE=(CYL,10)
```

If you want 20 tracks of a private 2311 volume named SYS007 for a kept data set named MIKE, and you want the volume to remain mounted at the end of the step, code:

```
//DASD15 DD DSNAME=MIKE,DISP=(,KEEP),UNIT=2311
//          VOLUME=(PRIVATE,RETAIN,SER=SYS007),SPACE=(TRK,20)
```

If your data set needs more than one volume, replace the term "volcount" with the number of volumes needed. If the number of volumes is equal to the number of devices requested in the unit parameter all volume are mounted at the same time. When the number of volumes is greater than the number of devices requested, and all the mounted volumes are used, the system demounts one of the volumes and then mounts another volume in its place so that processing can continue. If you specify RETAIN, the volumes mounted at the end of the step remain mounted.

The SER subparameter is coded as described for "Specific Request -- Public Volume."

In the following example, 50 tracks of a private 2301 volume named 2301AA are requested for a cataloged data set named RAY.

```
//DASD16 DD DSNAME=RAY,DISP=(,CATLG),UNIT=2301,
//          VOLUME=(PRIVATE,SER=2301AA),SPACE=(TRK,50)
```

In the following example, a temporary data set has a primary allocation of 500 tracks and a secondary allocation of 250 tracks. These tracks should be allocated on a 2311 volume named RFS300, but two more volumes may be needed. These volumes are to be mounted in succession on one 2311 drive.

```
//DASD17 DD UNIT=2311,VOLUME=(PRIVATE,,,3,SER=RFS300),
//          SPACE=(TRK,(500,250))
```

Using the preceding example, if you want the last volume used to remain mounted at the end of the job, code

```
//DASD17 DD UNIT=2311,VOLUME=(PRIVATE,RETAIN,,3,SER=RFS300),
//          SPACE=(TRK,(500,250))
```

In the following example a temporary data set named &&JIM has a primary allocation of 2500 tracks and a secondary allocation of 1500 tracks. These tracks should be allocated on two 2314 volumes named A17932 and A17936 that will mounted in parallel.

```
//DASD18 DD DSNAME=&&JIM,DISP=(,PASS),UNIT=(2314,P),
//          VOLUME=(PRIVATE,SER=(A17932,A17936)),
//          SPACE=(TRK,(2500,1500))
```

Using the preceding example, you may wish to indicate that a maximum of four volumes will be used. Volumes A17932 and A17936 will still be mounted in parallel. When an additional volume is needed, the system will demount a volume and replace it with another. Volumes are not demounted at the end of the job step because you coded DISP=(,PASS).

```
//DASD19 DD DSNAME=&&JIM,DISP=(,PASS),UNIT=(2314,P),
//          VOLUME=(PRIVATE,,,4,SER=(A17932,A17936),SPACE=(TRK,
//          (2500,1500))
```

2. The format of the VOLUME parameter for requesting volumes used by <u>other data sets</u> is:

```
                                           ┌dsname                    ┐
                                           │*.ddname                  │
VOLUME=(PRIVATE[,RETAIN][,,volcount,]REF=  │*.stepname. ddname        │)
               [,      ]                   │*.stepname.procstepname.  │
                                           └                    ddname┘
```

The PRIVATE subparameter is required. The RETAIN subparameter means that the volume will not be demounted at the end of the step. (RETAIN has the same effect on volume mounting as coding PASS in the DISP parameter.) Private volumes are demounted at the end of the step unless it is an unremovable volume, such as a 2301 drum, or unless it has been designated by your installation to be a permanently resident volume, for example, the system residence volume. It is recommended that you use the RETAIN subparameter whenever you want the volume to remain mounted at the end of the step.

The REF subparameter identifies the volumes assigned to an earlier data set. If the earlier data set resides on more than one volume, all the volumes are assigned to your data set. If your data set needs more volumes, you should indicate so with the "volcount" subparameter. Replace "volcount" with the maximum number of volumes that can be used for your data set. When you use the REF subparameter the system obtains UNIT information from the previous data set. However, you can use the UNIT parameter to assign additional units. For example,

```
VOLUME=(PRIVATE,,,4,REF=SYST.NINE),UNIT=(,3)
```

The REF subparameter is coded as described for "Specific Request -- Public Volume."

The following example shows how to code the REF subparameter to use the volume specified in a DD statement in a previous step.

```
//STEP5 EXEC...
//DD1 DD UNIT=2301,VOLUME=SER=75934
.
.
.
//STEP7 EXEC...
//DDA DD VOLUME=(PRIVATE,REF=*.STEP5.DD1),...
.
.
.
```

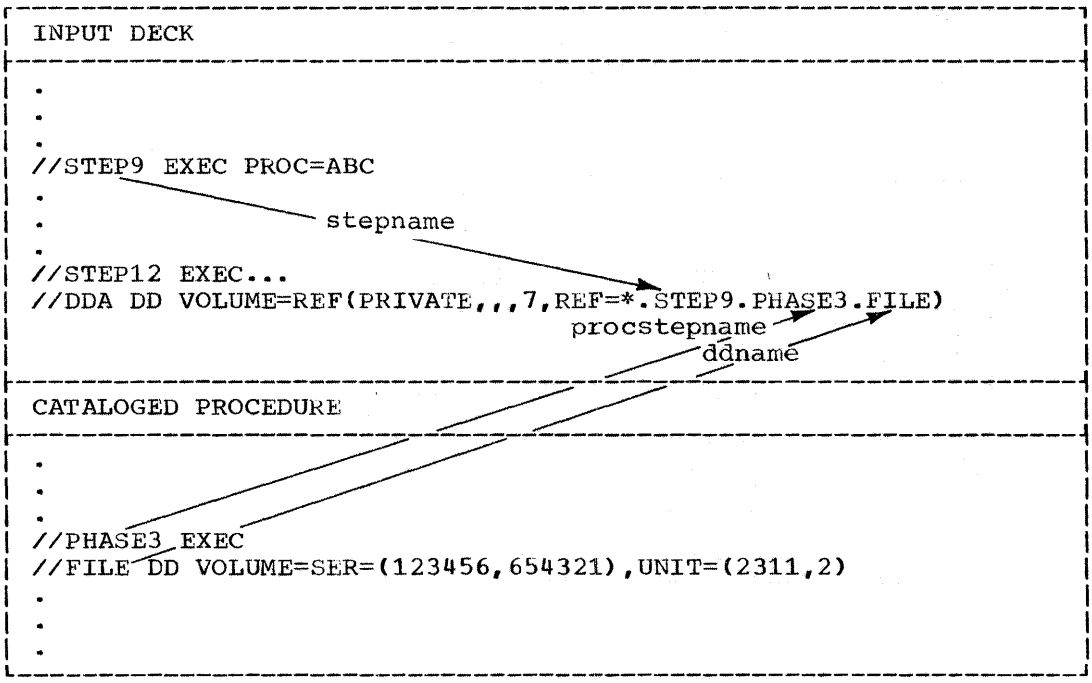
The following example shows how to code the REF subparameter to use the volumes specified in a DD statement in a previous step. In addition to the volumes used by the previous data set, the new data set may use three more volumes. The UNIT parameter is used to request one additional unit.

```
//STEPC EXEC...
//DD1 DD VOLUME=(PRIVATE,RETAIN,SER=(ABC111,ABC321)),
//       UNIT=(2314,2)
.
.
.
//DD5 DD VOLUME=(PRIVATE,,,5,REF=*.DD1),UNIT=(,3),...
.
.
.
```

When the earlier DD statement is contained in a cataloged procedure step, you must give both the name of the job step that invokes the procedure and the procedure step name, i.e.,

VOLUME=REF=*.stepname.procstepname.ddname

The first part of the following example shows your input deck; the second part shows a cataloged procedure called by the STEP9 EXEC statement in your job. The new data set requests a maximum of 7 volumes.



If you want to use the volume assigned to a kept data set defined in another job, you must use VOLUME=SER.

LABEL: The LABEL parameter is used to describe the type of label of the volume you selected with the VOLUME parameter. (The labels of direct access data set control blocks (DSCB). The DSCBs are part of the volume table of contents (VTOC) which is the volume label.) The LABEL parameter is also used to assign a retention period and password protection to your data set, and to indicate whether this data set is to be used for input or output exclusively. If the volume has standard labels you may omit this parameter. The format of the LABEL parameter is:

The first subparameter indicates the type of labels used for the specific volume you requested, or, if you are requesting a nonspecific volume, the type of labels you want on that volume. You can omit this subparameter if standard labels are used. You must specify one of the following:

SL
if the data set has standard labels.

SUL
if the data set has both standard and user labels.

The type of label used is determined by your installation. Most installations use only one type of labels. Check with your manager or supervisor before you specify a label type other than the type used in your installation.

PASSWORD is a positional subparameter. It tells the system that the data set you are creating cannot be used by another job step or job unless the operator can supply the system with the correct password. Password protected data sets must have standard labels, that is you must code the SL subparameter (or omit the second positional subparameter), for example:

```
//DD26 DD DSNAME=HUSH.HUSH,DISP=(,CATLG,DELETE),UNIT=2301,
//        VOLUME=SER=SECRET,LABEL=(,,PASSWORD),...
```

Before or after you create the data set you must tell the system programmer the name of your data set and the password. He will create an entry in a data set named PASSWORD containing those two items. Whenever a request is made for your data set the system will verify the password supplied by the operator against the entry in the PASSWORD data set.

The next positional subparameter (IN or OUT) is used only if you coded your program in the assembler language or in FORTRAN. If you coded in the assembler language it allows you to override the specification of the OUTIN parameter in the OPEN macro instruction (for BSAM data sets only). If OUTIN is specified and you want the data set processed for output only code:

```
LABEL=(,,,OUT)
```

If you are a FORTRAN user, OUT means that the data set is to be processed for output only; IN means that the data set is to be processed for input only.

You can assign a retention period (length of time during which the data set cannot be changed or deleted) to your nontemporary data set with either the RETPD=nnnn or EXPDT=yyddd subparameter. These subparameters are keyword subparameters. You must code the one you choose after the last positional subparameter you code. For example,

```
LABEL=(,,,IN,RETPD=34)
LABEL=(,SUL,EXPDT=69251)
LABEL=(,,PASSWORD,RETPD=100)
LABEL=(,SUL,,OUT,EXPDT=7100)
LABEL=RETPD=170
```

The RETPD subparameter expresses the retention period in terms of the number of days you want the data set retained. The format of RETPD is:

```
RETPD=nnnn
```

Replace "nnnn" with a number from 1 to 9999.

The EXTPD subparameter states the expiration data of your data set. The format of EXPDT is:

```
EXPDT=yyddd
```

Replace "yyddd" with the 2-digit year number and the 3-digit day number. For example, January 1, 1971 is coded as

```
LABEL=EXPDT=71001
```

and February 1, 1971 is coded as

```
LABEL=EXPDT=71032
```

(Many calendars indicate the day number for each day of the year.)

If neither RETPD or EXPDT is specified, a retention period of zero days is assumed. That is, your data set can be modified or deleted at anytime. Do not specify a retention period for temporary data sets, because they are always deleted at the end of the job step or the job.

Size of the Data Set

You must specify the size of your data set using the SPACE, SPLIT or SUBALLOC parameter. The parameter you use depends on the type of data set.

Data set	You can use
Sequential	SPACE, SPLIT, or SUBALLOC
Direct	SPACE, or SUBALLOC
Partitioned	SPACE, or SUBALLOC
Generation	SPACE

The SPACE parameter lets you specify the space required for your data set. The SPLIT and SUBALLOC parameters are used when your data set is sharing space with other data sets.

The SPLIT parameter is used when your step has two or more data sets having corresponding records. You can minimize access arm movemant by defining split cylinders. In the split-cylinder mode each data set is given a certain number of tracks on each cylinder allocated. You must

use the SPLIT parameter in each of the DD statements that define the data sets that are to split the cylinders.

The SUBALLOC parameter lets you use the technique of suballocation. Suballocation allows you to reserve a contiguous area of space on one volume and to place a series of data sets in a certain sequence in that area. Syballocation allows you to mimimize access-arm movement when data sets are processed serially. DD statements for the data sets that are to share that area must use the SUBALLOC parameter.

The SPACE, SPLIT and SUBALLOC parameters let you specify the amount of space needed for your data set as a primary quantity. If you think your data set may need some extra space either during the current job step or at some future time, you can specify a secondary quantity. If your data set exceeds the space in the primary quantity, the system allocates the space in the secondary quantity. If the secondary quantity runs out, another secondary quantity is allocated, and so on. The primary quantity is completely allocated on one volume. The system will try to allocate the secondary quantity on the same volume. If there is not enough space left on that volume, the system will allocate the secondary quantity on another volume provided you requested more than one volume with the VOLUME parameter. If you do not request more volumes or if the data set exceeds the volumes you requested the job step ABENDs. The job step will also ABEND if the primary quantity does not fit on the volume or if the data set exceeds the primary quantity and you did not specify a secondary quantity. (If you made a specific volume request with the VOLUME parameter, you must make sure there is enough space on the volume requested for the primary quantity. If there isn't, your step will ABEND. If you made a nonspecific volume request, the system will find a volume that has enough space for the primary quantity.)

In the SPACE and SUBALLOC parameters, you can specify the space required for your data set in units of tracks, cylinders, or blocks. In the SPLIT parameter you specify it in units of cylinders, or blocks.

Table 10 shows the capacities of the different direct access device. Table 11 shows the number of given-length records that can fit into a track. Table 11 is divided into two parts since the capacity varies depending on whether the record's format has keys. (For more information on direct access data sets refer to Introduction to IBM System/360: Direct Access Storage Devices Organization Methods, GC20-1649.)

The following example illustrates how to use the tables. Suppose you are writing a data set on a 2311 volume. This data set consists of 80-byte records grouped in blocks of 5 records. Therefore, each block has 400 bytes. You expect to write 2000 blocks, for a total of 800,000 bytes. A 2311 volume has over 7 million bytes, therefore, your data set will fit in one volume. According to Table 11, you can write seven 400-byte blocks on each track. You can allocate the space in one of the three following ways:

1. Units of blocks: You request 2000 400-byte blocks as follows:

   (400,2000)

2. Units of tracks: There are 2000 blocks and you are writing 7 blocks per track:

   2000/7=286 tracks (round up)

   You request 286 tracks as follows:

   (TRK,286)

3. Units of cylinders: There are 10 tracks in each 2311 cylinder

   286/10=29 cylinders (round up)

   You request 29 cylinders as follows:

   (CYL,29)

If you think your data set may need 50 more blocks you can make a secondary allocation as follows:

1. Units of blocks: A secondary allocation of 50 blocks is requested as follows:

   (400,(2000,50))

2. Units of tracks:

   50/7=8 tracks (round up)

   You request the secondary allocation as follows:

   (TRK,(286,8))

3. Units of cylinders:

   8/10=1 (round up)

   You request the secondary allocation as follows:

   (CYL,(29,1))

For most efficient performance, request space in units of cylinders. However, if you need to conserve space, be sure the cylinder is reasonably full. For example, it would be waste of space to request one cylinder on a 2302 for a data set that only needs two 2302 tracks.

For greater device independence, request space in units of blocks. The system calculates for you number of tracks or cylinders needed. The following considerations apply when you are requesting space in terms of blocks:

- The blocksize cannot exceed 65,535.
- If you have variable-length records (or blocks) give the average record (or block) length as the block size. You must specify the maximum record length in the BLKSIZE subparameter of the DCB parameter.
- If your blocks are written with keys, give the key length in the KEYLEN subparameter of the DCB parameter.

In addition to the primary quantity, partitioned data sets require a directory quantity. (A secondary quantity can also be specified for partitioned data sets.) The directory quantity is the number of 256-byte blocks that are to be contained in the directory of the partitioned data set. Each block contains from 3 to 21 entries. There must be one entry for each member name and for each alias in the partitioned data set. The length of the entries depends on the length of a user's data field. For example, if your partitioned data set has 20 members and each member has aliases, you need 60 entries. If you can fit five entries into each directory block, you need 12 directory blocks. Therefore you must specify 12 as the directory quantity. The directory quantity is always the same whether you specify your space allocation in blocks, tracks, or cylinders. For example,

```
(400,(2000,50,12))
(TRK,(286,8,12))
(CYL,(29,1,12))
```

For more information on partitioned data sets, refer to IBM System/360 Operating System: Supervisor and Data Management Services.

Table 10. Direct Access Capacities

Device	Storage Medium	Cylinders	Tracks Per Cylinder	Bytes Per		
				Track	Cylinder	Device (in millions)
2301	Drum	25*	8	20,483	4.09 (million)	4.09
2302	Disk	Model 3: 492 Model 4: 984	46	4,984	229,264	Model 3: 112.79 Model 4: 225.59
2303	Drum	80	10	4,892	48,920	3.9
2311	Disk	200	10	3,625	36,250	7.25
2314 (each volume)	Disk	200	20	7,294	145,880	29.17
2321	Strip of Tape	980**	20	2,000	40,000	39.2

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

Table 11. Track Capacity

Maximum Bytes per Record Formatted without Keys						Records per Track	Maximum Bytes per Record Formatted with Keys					
2311	2314	2302	2303	2301	2321		2311	2314	2302	2303	2301	2321
3625	7294	4984	4892	20483	2000	1	3605	7249	4964	4854	20430	1984
1740	3520	2403	2392	10175	935	2	1720	3476	2383	2354	10122	920
1131	2298	1570	1558	6739	592	3	1111	2254	1550	1520	6686	576
830	1693	1158	1142	5021	422	4	811	1649	1139	1104	4968	406
651	1332	912	892	3990	320	5	632	1288	893	854	3937	305
532	1092	749	725	3303	253	6	512	1049	730	687	3250	238
447	921	634	606	2812	205	7	428	877	614	568	2759	190
384	793	546	517	2444	169	8	364	750	527	479	2391	154
334	694	479	447	2157	142	9	315	650	460	409	2104	126
295	615	425	392	1928	119	10	275	571	406	354	1875	103
263	550	381	346	1741	101	11	244	506	362	308	1688	85
236	496	344	308	1585	86	12	217	452	325	270	1532	70
213	450	313	276	1452	73	13	194	407	294	238	1399	58
193	411	286	249	1339	62	14	174	368	267	211	1286	47
177	377	264	225	1241	53	15	158	333	245	187	1188	38
162	347	244	204	1155	44	16	143	304	224	166	1102	29
149	321	225	186	1079	37	17	130	277	206	148	1026	21
138	298	209	169	1012	30	18	119	254	190	131	959	15
127	276	196	155	952	24	19	108	233	176	117	899	9
118	258	183	142	897	20	20	99	215	163	104	844	
109	241	171	130	848	15	21	90	198	152	92	795	
102	226	161	119	804	10	22	82	183	142	81	751	
95	211	151	109	763	6	23	76	168	132	71	710	
88	199	143	100	726		24	69	156	123	62	673	
82	187	135	92	691		25	63	144	116	54	638	
77	176	127	84	659		26	58	133	108	46	606	
72	166	121	77	630		27	53	123	102	39	577	
67	157	114	70	603		28	48	114	95	32	550	
63	148	108	64	577		29	44	105	89	26	524	
59	139	102	58	554		30	40	96	83	20	501	

SPACE: The SPACE parameter lets you request space for your data set in one of two ways:

1. You can request the quantity of space and let the system assign specific tracks, or

2. You can request specific tracks.

The system assigns tracks: The format of the SPACE parameter for letting the system assign specific tracks is:

```
SPACE=( {TRK         }, (primary quantity [,secondary quantity]
        {CYL         }                     [,                  ]
        {block length}
                                                          [,CONTIG]
                        [,directory]) [,RLSE] [,MXIG  ] [,ROUND])
                                      [,    ] [,ALX   ]
                                              [,      ]
```

The first positional parameter indicates whether you are requesting space in tracks (TRK), cylinders (CYL), or blocks. If you are using blocks, replace the term "block length" with average block length of your data. Replace the term "primary quantity" with the amount of space you desire in the units you have chosen. The system tries to allocate the primary quantity in contiguous tracks or cylinders. If contiguous space is not available, the system satisfies the space request with up to five noncontiguous blocks (extents) of space. You can override these system actions by coding the CONTIG, MXIG, or ALX subparameter. If you want additional space allocated to your data set when the primary quantity runs out, specify a "secondary quantity." As with the primary quantity, the system will try to allocate the secondary quantity in contiguous tracks or cylinders. If this is not possible, the system will allocate the secondary quantity in up to five extents.

If you are defining a partitioned data set, you must define a "directory quantity." Replace the term "directory quantity" with the number of 256-byte blocks you need for the directory. If you code a directory quantity but do not request a secondary quantity, indicate the absence of a secondary quantity with a comma.

If you are only coding the primary quantity you need not enclose it in parentheses. For example:

SPACE=(TRK,200)

In the following example you request 30 cylinders for your data set. If the data set needs more space you want the system to allocate 5 more cylinders at a time:

SPACE=(CYL,(30,5))

In the following example you request 300 2000-byte blocks for your partitioned data set. The directory needs 50 256-byte blocks.

SPACE=(2000,(300,,50))

Using the previous example, if you also want to make a secondary allocation of 10 2000-byte blocks code:

SPACE=(2000,(300,10,50))

The RLSE positional subparameter allows you to release unused space when you are finished writing the data set. If space was requested in terms of tracks, space is released starting with the first unused track. If space was requested in terms of cylinders, space is released starting with the first unused cylinder. If space was requested in terms of blocks, any unused tracks are released. Any released space is available

for allocation to other data sets. You can use the RLSE subparameter and still be able to update your data set at a later time if you specify a secondary quantity for the data set. For example, if you code

```
SPACE=(CYL,(10,2),RLSE)
```

10 cylinders will be allocated to your data set. If you use only 9 cylinders, the remaining cylinder is released. If during a later job step or job you want to update the data set, two cylinders (secondary allocation) will be allocated to your data set each time you need more space. If you do not code the RLSE subparameter, and other subparameters follow, you must indicate its absence by a comma.

The next positional subparameter allows you to override the system's allocation technique. The system tries to allocate space in contiguous track or cylinders. If contiguous space is not available, the system satisfies the space request with up to five noncontiguous blocks of space (extents). You can choose one of the following to override these actions.

CONTIG (contiguous)
specifies that the space allocated to the data set must be contiguous. If the request cannot be satisfied, the job is terminated. For example,

```
SPACE=(TRK,(50,,2),,CONTIG)
```

requests 50 contiguous tracks for your partitioned data set.

MXIG (maximum contiguous)
requests the largest area of contiguous space available on the volume that is at least as large as the primary quantity. For example, if you code

```
SPACE=(CYL,10,,MXIG)
```

and there are three blocks of unused space on the volume -- a 10 cylinder block, a 12 cylinder block, and a 20 cylinder block -- the system will assign your data set the 20 cylinder block even if you only requested ten. MXIG is useful when your data set is likely to exceed the primary quantity requested.

ALX (all extents)
requests up to five different areas of contiguous space. Each area is at least as large as the primary quantity. The system allocates as many areas as are available. For example, if you code

```
SPACE=(TRK,(100,20),,ALX)
```

and there are four blocks of unused space on the volume -- a 120-track block, a 200-track block, a 35-track block, and a 155-track block -- the system will assign the 120-200- and 155-track blocks to your data set. If your data set still needs more space, a secondary allocation of 20 tracks is made on the remaining 35 tracks. If more space is still needed, the system will allocate 20 more tracks on another volume, provided you request more than one volume with the volume parameter. ALX is useful when your data set requires an unusually large amount of space.

Because of the unpredictably large amounts of space that can be allocated to a data set with the MXIG or ALX subparameter, RLSE is often coded at the same time to return all unused space to the system. RLSE can also be coded with CONTIG.

If you request space on a specific volume, you must determine whether there is enough contiguous space for the primary allocation, because if there isn't your step will abnormally terminate. If you make a nonspecific request, the system will ensure that you get enough space on a volume for your primary allocation. In either case, if you need a very large amount of space (especially if you are using ALX) you should consider requesting a large secondary quantity at the same time. A secondary allocation will let the system obtain space on more than one volume, if necessary.

If you do not specify CONTIG, MXIG, or ALX, and the ROUND subparameter follows, you must code a comma to indicate their absence.

The ROUND subparameter is used only when you request space in terms of blocks. ROUND requests that the allocated space be equal to one or more whole cylinders. The system computes the number of tracks needed to hold the blocks and ensures that the space begins on the first track of a cylinder and ends on the last track of a cylinder. For example, if you code:

```
UNIT=2311,SPACE=(400,2000,,,ROUND)
```

the system will assign you 29 whole cylinders, rather than 28.6 cylinders (286 tracks).

All other subparameters (except TRK and CYL) can be coded with ROUND.

Requesting Specific Tracks: A data set can be placed in a specific position on a direct access volume by requesting space in terms of a quantity and track number. This absolute track allocation technique is recommended only for location-dependent data sets.

The format of the SPACE parameter for requesting specific tracks is:

```
SPACE=(ABSTR,(primary quantity,address[,directory]))
```

The ABSTR subparameter is required. Replace the term "primary quantity" with the number of tracks you desire, and "address" with the relative track address of the beginning track. Relative track numbers are based on the first track of the first cylinder being defined as 0. (Track 0 cannot be requested.) Count through the tracks of each cylinder until reach the track on which you want the data set to start. Table 10 will help you to perform this calculations. The system automatically converts the relative track number to an address based on the particular device assigned.

For example, if you want 50 tracks beginning with the third track of the second cylinder of a 2311 volume, code.

```
SPACE=(ABSTR,(50,12))
```

If you want 10 tracks beginning with the fourth track of the first cylinder of a 2314 volume, code

```
SPACE=(ABSTR,(10,3))
```

If the data set is a partitioned data set, replace "directory" with the number of directory blocks required.

If the tracks you request have already been allocated to another data set, or the data set exceeds the space available on the volume, the job

is terminated. A secondary quantity cannot be specified when you request specific tracks.

Note: Since a secondary quantity cannot be coded with the ABSTR subparameter, the absolute track allocation technique cannot be used to create a multivolume data set.

SPLIT: When a job step involves one or more sequential data sets having corresponding records, you can minimize access arm movement by defining split cylinders. In the split-cylinder mode each data set is given part of the tracks on every cylinder allocated. For example, if you want to allocate space to three data sets on a 2311 volume using the split-cylinder mode, you could allocate 15 cylinders to be shared by the three data sets. The first data set would occupy the first four tracks of each cylinder; the second data set would occupy the next five tracks; and the third data set would occupy the last track of each cylinder. (There are 10 tracks in each 2311 cylinder.)

The cylinders allocated to the data sets must be on one 2302, 2311, 2314 or 2321 volume. If there are not enough cylinders available on the volume to satisfy the request, the job is terminated. The space occupied by a data set residing on a cylinder that has been split is not available for reallocation until all data sets sharing the cylinder are deleted. (If the SPLIT parameter is used to allocate space for data sets that are to reside on a drum storage volume, space is allocated for the data sets, but the data sets are not stored using the split cylinder mode.)

To split cylinders among two or more data sets, you must arrange the associated DD statements in sequence in the input stream. The first DD statement in the sequence specifies the portion of the space required by the first data set and the total amount of space required for all data sets. Each succeeding DD statement requests a portion of the total space. You can specify a secondary quantity on the first DD statement in the sequence. If any of the data sets in the group exceeds its space, additional space is allocated in the amount of the secondary quantity. The additional space applies only to the data set that ran out of space, and is not split with the other data sets in the group. If a secondary quantity is not specified and one of the data sets in the group runs out of space, the job step ABENDs.

The SPLIT parameter lets you request space in units of cylinders or in units of blocks.

Units of Cylinders: The format of the SPLIT parameter for the first data set in the sequence is:

```
SPLIT=(n,CYL,(primary quantity [,secondary quantity] ))
```

Replace "n" with the number of tracks per cylinder you want allocated to the first data set. The CYL subparameter is required. Replace "primary quantity" with the total number of cylinders to be allocated for use by all the data sets in the group. Replace "secondary quantity" with the number of cylinders to be allocated to any data set in the group that runs out of space.

The format of the SPLIT parameter for each succeeding data set in the sequence is:

```
SPLIT=n
```

Replace "n" with the number of tracks per cylinder you want allocated to this data set. Be careful that the total number of "n" quantities specified for the data sets in the group is not greater than the number of tracks per cylinder. You can omit the VOLUME and UNIT parameters from all DD statements after the first of the group. If you code them, they are ignored.

In the following example three cataloged data sets named ABC.ONE, ABC.TWO and ABC.THREE are to split 50 cylinders on a private 2314 volume named VOL097. ABC.ONE is to have seven tracks on each cylinder; ABC.TWO is to have 9 tracks; and ABC.THREE is to have the remaining four tracks of each cylinder. Two more cylinders can be allocated to any of the three data sets that runs out of space.

```
//DD1 DD DSNAME=ABC.ONE,DISP=(,CATLG),UNIT=2314,
//       VOLUME=(PRIVATE,SER=VOL097),
//       SPLIT=(7,CYL,(50,2))
//DD2 DD DSNAME=ABC.TWO,DISP=(,CATLG),SPLIT=9
//DD3 DD DSNAME=ABC.THREE,DISP=(,CATLG),SPLIT=4
```

Units of Blocks: When you specify space in units of blocks, the system computes how many cylinders to allocate. Instead of specifying how many tracks each data set is to have, you specify which percentage of the cylinder each data set is to have.

The format of the SPLIT parameter for the first data set in the sequence is:

```
SPLIT=(%,block length,(primary quantity[,secondary quantity]))
```

Replace "%" with a number from 1 to 99. This number represents the percentage of tracks per cylinder you want allocated to the data set. Replace "block length" with the average block length of your data. Replace "primary quantity" with the total number of blocks to be allocated for use by all the data sets in the group. Replace "secondary quantity" with the number of blocks to be allocated to any data set in the group that runs out of space. (The system rounds up the secondary quantity to an integral-number of cylinders.)

The format of the SPLIT parameter for each succeeding data set in the sequence is:

```
SPLIT=%
```

Replace "%" with a number from 1 to 99, this number representing the percentage of tracks per cylinder you want allocated to this data set. The total number of percentages allocated to the data sets in the group must not be greater than 100. You can omit the VOLUME and UNIT parameters from all DD statements after the first of the group. If you code them they are ignored.

In the following example, four temporary data sets named &&ALPHA,&&BETA, &&DELTA, and &&GAMMA are to split 200 1024-byte blocks on a public 2321 volume. &&ALPHA is to have 10% of each cylinder; &&BETA is to have 25%; &&DELTA is to have 50%; and &&GAMMA is to have 15%.

```
//DDA DD DSNAME=&&ALPHA,DISP=(,PASS),UNIT=2321,
//       SPLIT=(10,1024,(200))
//DDB DD DSNAME=&&BETA,DISP=(,PASS),SPLIT=25
//DDC DD DSNAME=&&DELTA,DISP=(,PASS),SPLIT=50
//DDD DD DSNAME=&&GAMMA,DISP=(,PASS),SPLIT=15
```


SUBALLOC: Suballocation allows you to place a series of data sets in a certain sequence in a contiguous area of one direct access volume. This reserved area is defined with a DD statement using the SPACE parameter. The DD statements for the data sets that share that area must use the SUBALLOC parameter to request a portion of the area. All DD statements must be in the same job.

Defining the Reserved Area: The reserved area is a sequential data set. This data set must be used for suballocation purposes only; that is, you must not write any data in the data set. You must use the following SPACE parameter to request space for the area:

```
SPACE=({TRK       },primary quantity,,CONTIG)
       {CYL       }
       {block length}
```

You can request space in units of tracks, cylinders, or blocks. The "primary quantity" must be large enough to contain all the primary quantities of the suballocated data sets. (The suballocated data sets can request secondary quantities, but any secondary quantity will be allocated outside of the reserved area.) The CONTIG subparameter is required because the reserved area must be contiguous.

On this same DD statement, you can request more than one device (UNIT parameter) and volume (VOLUME parameter). If you do this, a suballocated data set that exceeds its primary quantity can have its secondary quantity allocated on another volume.

If any of the suballocated data sets is to be kept or cataloged, you should specify

```
DISP=(,KEEP)
```

for the reserved area. (It is not necessary to catalog it, but you may do so if you wish.) If all the suballocated data sets are temporary, you can specify

```
DISP=(,PASS)
```

for the reserved area.

Suballocated Data Sets: The DD statements for the suballocated data set must be in the same job as the DD statement for the reserved area. The DD statement for the reserved area must precede those of the suballocated data sets. Each data set is assigned the next portion of unused space from the reserved area.

You can omit the UNIT and VOLUME parameters from the DD statements for the suballocated data sets. If you code them they are ignored.

You must use the SUBALLOC parameter to request space for each suballocated data set.

```
SUBALLOC=( {TRK         } ,(primary quantity [,secondary quantity]
           {CYL         }                    [,                  ]
           {block length}
                                              {ddname                       }
                           [,directory]),     {stepname.ddname              })
                                              {stepname.procstepname.ddname }
```

You can request the suballocated space in units of tracks, cylinders, or blocks. (You can use a different unit for each suballocated data set; the system will perform the appropriate calculations.) Replace the term "primary quantity" with the amount of space you desire in the units you have chosen. If there is not enough space in the reserved area for the primary quantity, the job will ABEND. If you want additional space allocated to your data set when the primary quantity runs out, specify a "secondary quantity". The secondary quantity will not be allocated within the reserved area, but on available space elsewhere on the volume. If you requested more devices and volumes (with the UNIT and VOLUME parameters) in the DD statement for the reserved area, the secondary allocation could be made on another volume. (You must request more devices in addition to more volumes because the volume that contains the reserved area cannot be demounted until all requests in the job for suballocation have been satisfied.) If you are defining a partitioned data set, you must define a "directory quantity." Replace the term "directory quantity" with the number of 256-byte blocks you need for the directory. If you code a directory quantity but do not request a secondary quantity, indicate the absence of the secondary quantity with a comma.

The last positional parameter identifies the DD statement that defines the reserved area. Replace "ddname" with the name of the DD statement of the reserved area. Replace "stepname" with the name of the EXEC statement of the step that has the DD statement of the reserved area. If the earlier DD statement is in the same jobstep, omit the stepname.

When the DD statement that defines the reserved area is in a cataloged procedure, you must give both the name of the job step that invokes the procedure and the procedure step name, i.e., stepname.procstepname.ddname.

In the following example, the reserved area is defined with the DD statement named ONE in STEP1. The name of the reserved area is AREA, and it will occupy 150 cylinders of a 2314 volume named LOUISE. The DD statement TWO in STEP1 defines a cataloged data set named SUB.ONE. SUB.ONE requests the first 200 tracks of AREA. The DD statements named THREE and FOUR in STEP2 define nontemporary data sets named ALPHA and BETA. ALPHA is a partitioned data set that requests 110 cylinders of AREA. Seven directory blocks are needed for ALPHA. BETA is a sequential data set that needs 1000 1024-byte blocks. Should BETA need more space, a secondary allocation of 50 blocks can be made outside of AREA.

```
//STEP1 EXEC PGM=BLURB
//ONE DD DSNAME=AREA,DISP=(,KEEP),UNIT=2314,
//       VOLUME=SER=LOUISE,SPACE=(CYL,150,,CONTIG)
//TWO DD DSNAME=SUB.ONE,DISP=(,CATLG),
//       SUBALLOC=(TRK,(200),ONE)
//STEP2 EXEC PGM=BLA
//THREE DD DSNAME=ALPHA,DISP=(,KEEP),
//         SUBALLOC=(CYL,(110,,7),STEP1.ONE)
//FOUR DD DSNAME=BETA,DISP=(,KEEP),
//       SUBALLOC=(1024,(1000,50),STEP1.ONE)
```


Data Attributes

The DCB parameter allows you to specify data set attributes for your data set when your program is to be executed rather than when it is compiled. Any applicable attributes not specified in your program must be specified with the DCB parameter. However, in most cases, your compiler will provide a default value for an attribute if you do not specify it in the program or in the DD statement. Other attributes are always given a fixed value by the compiler and you do not have to specify them at all. For example, you can select a buffering technique with the assembler, but all other IBM compilers select one for you when it is needed.

DCB: You can use the DCB parameter to directly specify the attributes of your data set, or to copy those attributes specified for a cataloged data set or in a DD statement for another data set.

The format of the DCB parameter for specifying the attribute is:

```
DCB=(list of attributes)
```

The attributes in the list are coded in the form of keyword subparameters separated by commas; for example,

```
DCB=(BLKSIZE=300,LRECL=100)
```

The valid subparameters that can be used with each compiler for sequential, direct, and partitioned data sets are shown in Tables 12, 13, and 14 respectively. Underscored items are those default values selected if you omit the subparameter. Default values are not shown where the attribute can be specified either in your program or in the DD statement. If values for a given subparameter are not shown, it is either specified in your source program or given a fixed value by the compiler. A glossary of DCB parameters is given in Appendix B. Code only those parameters that apply to your compiler as shown in Tables 12, 13, and 14. Programs written in ALGOL or FORTRAN H cannot use direct data sets. Partitioned data sets can only be used by programs written in assembler language.

Table 12. DCB Subparameters for Creating a Sequential Data Set on Direct Access Devices

DCB Subparameter	ALGOL	Assembler	COBOL E	COBOL F	ANS COBOL [3]	FORTRAN E	FORTRAN G & H	PL/I F
BFALN=		F or D [1,2]						
BFTEK=		S, E, or A (QSAM only) [1,2]						
BLKSIZE=	number of bytes	number of bytes [1]		number of bytes [1]	number of bytes [1]	number of bytes	number of bytes	number of bytes [1]
BUFL=		number of bytes [1]						
BUFNO=		number of buffers [1]	number of buffers [1]	number of buffers [1]	number of buffers [1]	1 or 2	1 or 2	number of buffers [1]
DSORG=		PS or PSU [1]						
EROPT=		ABE [1]	ABE	ABE	ABE			
HIARCHY=		0 or 1 [1]						
KEYLEN=		number of bytes [1]						number of bytes [1]
LRECL=	number of bytes	number of bytes [1]				number of bytes	number of bytes	number of bytes [1]
NCP=		number of channel programs (BSAM only) [1]						number of channel programs (BSAM only) [1]
OPTCD=		[W] [C] [T] [1]	[W] [C]	[W] [C]	[W] [C]	[C]	[C]	[W] [C]
RECFM=	F[B/BS][A][T]	U[T][A/M] or V[B/S/T/BS/BT/ST/BST][A/M] or F[B/S/T/BS/BT/ST/BST][A/M]				Formatted: U[A][T] or V[B][A][T] or F[B][A/M][T] Unformatted: VS[B][A/M][T]	Formatted: U[A/M][T] or V[B][A/M][T] or F[B][A/M][T] Unformatted: VS[B][A/M][T]	[1] U[A/M] or V[B/S/BS][A/M] or F[B/S/BS][A/M]

[1] This function can be specified in your program rather than in the DD statement.

[2] For QSAM, you must specify both BFALN and BFTEK on the DD statement or omit both.

[3] American National Standard COBOL.

Table 13. DCB Subparameters for Creating a Direct Data Set

DCB Subparameter	Assembler	COBOL E	COBOL F	ANS COBOL [2]	FORTRAN E	FORTRAN G	PL/I F
BFALN=	F or D [1]						
BFTEK=	R [1]						
BLKSIZE=	number of bytes [1]						number of bytes [1]
BUFL=	number of bytes [1]						
BUFNO=	number of buffers [1]				1 or 2	1 or 2	
DSORG=	DA or DAU [1]		DA	DA	DA	DA	DA
HIARCHY=	0 or 1 [1]						
KEYLEN=	number of bytes [1]						number of bytes
LIMCT=	number of tracks or blocks [1]			number of tracks or blocks [1]			number of tracks or blocks
OPTCD=	[W] [E] [F] [R A] [1]	[W]	[W]	[W] [E]			[W]
RECFM=	U,V, or F[T] [1]						U,V, or F[T] [1]

[1] This function can be specified in your program rather than in the DD statement.

[2] American National Standard COBOL.

Table 14. DCB Subparameters for Creating a Partitioned Data Set

DCB Subparameter	Assembler
BFALN=	F or D
BLKSIZE=	number of bytes
BUFL=	number of bytes
BUFNO=	number of buffers
HIARCHY=	0 or 1
KEYLEN=	number of bytes
LRECL=	number of bytes
NCP=	number of channel programs
OPTCD=	[W] [C]
RECFM=	U[T][A or M] V[B][T][A or M] F[B][T][A or M]

[1] This function can be specified in your program rather than in the DD statement.

The format of the DCB parameter for copying the DCB parameter of a previous DD statement is:

```
                  ┌dsname                          ┐
DCB=(             │*.ddname                        │[,list of attributes])
                  │*.stepname.ddname               │
                  └*.stepname.procstepname.ddname  ┘
```

Replace "dsname" with the name of the cataloged data set whose attributes you want to copy. The volume that contains this data set must be mounted before the execution of the job step containing the copy request. A permanently resident volume is the most likely place from which to copy such information, in that it is always mounted. Unless you specify the volume sequence number and expiration date on the DD statement, these parameters are copied along with the DCB information. The following example shows how to code the DCB parameter to copy the attributes of a cataloged data set named SYS1.SVCLIB:

```
//NEWDD  DD...,DCB=SYS1.SVCLIB
```

Replace "ddname" with the name of the DD statement whose DCB parameter you want to copy. Replace "stepname" with the name of the EXEC statement of the step that contains that DD statement. If the DD statement you want to copy is contained in the same job step, omit the stepname, i.e.,

```
DCB=*.ddname
```

The following example shows how to code the DCB parameter to copy the DCB parameter of a DD statement in a previous step:

```
//STEP2 EXEC...
//DD1 DD...,DCB=(BLKSIZE=1600,LRECL=80)
.
.
.
//STEP4 EXEC...
//COPY DD...,DCB=*.STEP2.DD1
.
.
.
```

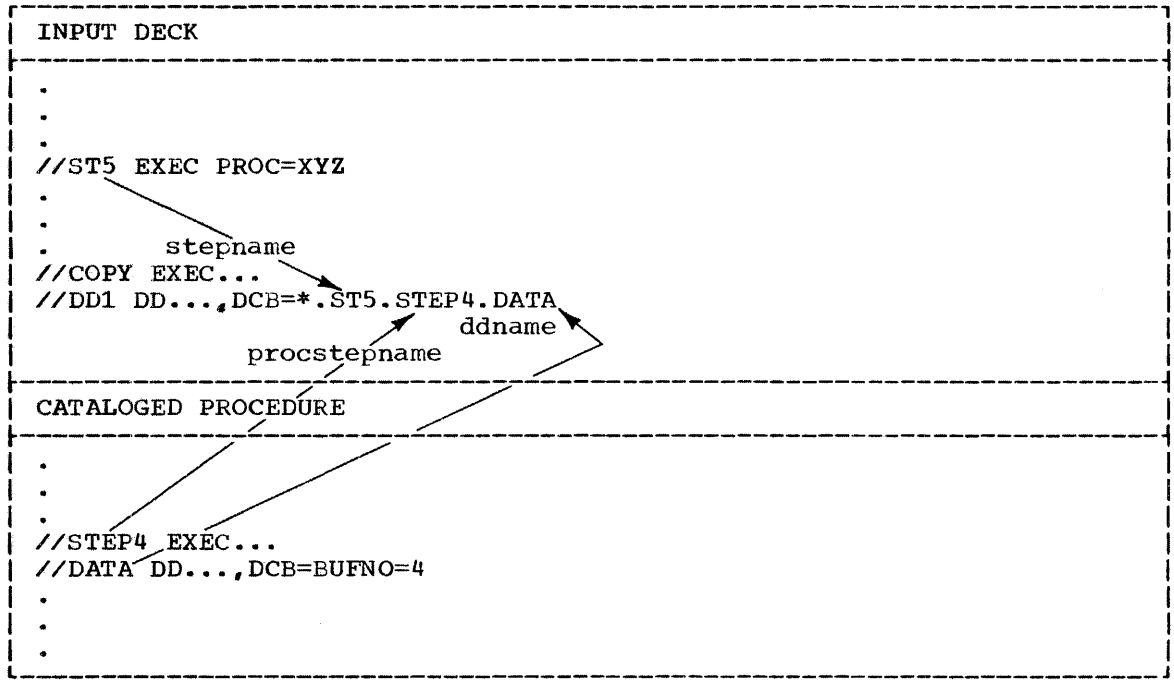
The following example shows how to code the DCB parameter of a previous DD statement in the same step:

```
.
.
.
//STEPD EXEC...
//DDA DD...,DCB=RECFM=F
.
.
.
//DDC DD...,DCB=*.DDA
.
.
.
```

If you want to copy the DCB parameter of a DD statement contained in a cataloged procedure step, you must give both the name of the job step that invokes the procedure and the precedure step name, i.e.,

```
DCB=*.stepname.procstepname.ddname
```

The first part of the following example shows your input deck; the second part shows a cataloged procedure called by the ST5 EXEC statement in your deck.



If you want to modify the DCB subparameters you are copying add the new subparameters to the reference. The subparameters you specify will override the corresponding copied subparameters. For example,

```
.
.
.
//OUT EXEC...
//DASDA DD...,DCB=(BLKSIZE=160,LRECL=80)
//DASDB DD...,DCB=(*.PUNCH1,BLKSIZE=1600)
.
.
.
```

Special Processing Options

There are two special processing options for data sets on direct access devices:

1. You can request channel separation from other data sets in the same job step using either the SEP or the AFF parameter.

2. You can suppress I/O operations on a sequential data set using the DUMMY parameter.

SEP: When two or more data sets are to be used in a job step, processing time may be shortened by requesting that the system transmit data sets over separate channels. For example, it would be faster to have your input data set and your output data set on separate channels than to have them on the same channel.

You can request channel separation for data sets in each job step using the SEP parameter. If possible, the system will honor this request. It may not always be possible to honor the request for separation because given devices may not be available for allocation when the job step is executed or because there may not be enough devices of a given type on different channels for all the data sets that request separation. If channel separation is not requested, the system will assign any available channel that has the device specified with the UNIT parameter.

The format of the SEP parameter is:

```
SEP=(ddname,...)
```

Replace the terms "ddname" with the names of up to eight earlier DD statements in the same job step. The earlier DD statements can define any type of data set: new or existing, on magnetic tape or direct access, temporary or nontemporary, etc.

In the following example, you request that the data set defined by the FOUR DD statements be assigned to a channel other than the ones assigned to the data sets defined by the ONE and TWO DD statement. The data sets defined by ONE and TWO may or may not be on the same channel. The data set defined by THREE may or may not be on the same channel as any of the other three data sets.

```
//PROG1 EXEC PGM=PROCESS
//ONE DD UNIT=2400,VOLUME=PRIVATE,LABLE=(,NL)
//TWO DD DSNAME=SYSTEM.USER.ABC,DISP=OLD
//THREE DD DSNAME=HELP,DISP=(,KEEP),UNIT=2314,
//         VOLUME=SER=AFL,LABEL=RETPD=31,SEP=TWO
//FOUR DD DSNAME=END,DISP=OLD,UNIT=2400-2,
//         VOLUME=SER=TAPE75,SEP=(ONE,TWO)
```

Using the preceding example, if you want to request that the data set defined by the THREE DD statement be on a channel other than the one assigned to the data set defined by the TWO DD statement code:

```
//PROG1 EXEC PGM=PROCESS
//ONE DD UNIT=2400,VOLUME=PRIVATE,LABEL=(,NL)
//TWO DD DSNAME=SYSTEM.USER.ABC,DISP=OLD
//THREE DD DSNAME=HELP,DISP=(,KEEP),UNIT=2314,
//         VOLUME=SER=AFL,LABEL=RETPD=31,SEP=TWO
//FOUR DD DSNAME=END,DISP=OLD,UNIT=2400-2,
//         VOLUME=SER=TAPE75,SEP=(ONE,TWO)
```

As a result of this example, the data sets defined by ONE and TWO may or may not be on the same channel. The data set defined by THREE and FOUR may or may not be on the same channel, but neither one will be on the same channel as the data set defined by TWO. Also, the data set defined by FOUR will not be on the same channel assigned by ONE.

Remember that the request for channel separation made with the SEP parameter is only a suggestion to the system for more efficient operation, and not a requirement. If your data sets must be on separate channels specify unit addresses with the UNIT parameter. You can obtain a configuration chart with all the unit addresses from your manager or supervisor.

AFF: The AFF parameter provides a shortcut method of requesting channel separation. When two or more data sets in the same job step have the same channel separation requirements, you can code the SEP parameter for the first data set, and then request the same separation using the AFF parameter in the later data sets. The AFF parameter tells the system that you want the data set defined on this DD statement to have the same channel separation as the data set defined in the named DD statement.

The format of the AFF parameter is:

```
AFF=ddname
```

Replace "ddname" with the name of the earlier DD statement that requests the desired channel separation. This data set and the earlier data set may or may not be on the same channel.

In the following example, the data sets defined by DD1 and DD2 may or may not be on the same channel. The data sets defined by DD3 and DD5 may or may not be on the same channel, but neither one will be on the same channel as the data sets defined by DD1 and DD2. The data set defined by DD4 will not be on the same channel as the data set defined by DD3, but it may be on the same channel as the data sets defined by DD1, DD2, or DD5.

```
//STEP  EXEC  PGM=NINE
//DD1   DD    ...
//DD2   DD    ...
//DD3   DD    SEP=(DD1,DD2),...
//DD4   DD    SEP=DD3,...
//DD5   DD    AFF=DD3,...
```

The requests for channel separation and affinity are only suggestions to the system. They will be honored if the appropriate devices are available.

DUMMY: The DUMMY parameter allows you to bypass I/O operations and data set disposition. When your processing programs ask to write the dummy data set, the write request is recognized, but no data is transmitted. This facility is particularly useful when you are debugging your program because it saves you processing time. For example, you can suppress the writing of a data set until you are sure your program is going to produce meaningful output. The DUMMY parameter is a positional parameter. That means that it must be coded first in the operand field. For example:

```
//OUTPUT DD DUMMY,UNIT=2311,VOLUME=SER=AAA999,SPACE=(TRK,(20,1))
//DASD2 DD DUMMY,DSNAME=A.B.C.,DISP=(CATLG),UNIT=2301,
//         VOLUME=(PRIVATE,RETAIN),SPACE=(2000,(500,,20),
//         DCB=(BUFNO=2,BLKSIZE=2500,LRECL=50)
```

If you wish, you can omit all other parameters required to define your data set except DCB. (If you were not going to write the DCB parameter you do not have to write it.) For example, the two DD statements shown above could be coded as follows:

```
//OUTPUT DD DUMMY
//DASD2 DD DUMMY,DCB=(BUFNO=2,BLKSIZE=2500,LRECL=50)
```

When you are ready to perform I/O operations on your data set, simply omit the DUMMY parameter and code all parameters required to write the data set. For example:

```
//OUTPUT DD UNIT=2311,VOLUME=SER=AAA999,SPACE=(TRK,(20,1))
//DASD2 DD DSNAME=A.B.C,DISP=(CATLG),UNIT=2301,
//         VOLUME=(PRIVATE,RETAIN),SPACE=2000,(500,,20),
//         DCB=(BUFNO=2,BLKSIZE=2500,LRECL=50)
```

Table 15. Parameters for Retrieving a Data Set

Data Set	Parameter Type	Parameter	Comments
Unit Record Devices	Location of the data set	UNIT	Required.
	Data attributes	DCB	Optional.
	Special Processing Option	DUMMY	Optional.
Input Stream	Location of the data set	*	You must specify one of these parameters.
		DATA	
	Data attributes	DCB	Systems with MFT or MVT only. Optional.
Passed Data Set	Data set information	DSNAME	Required.
		DISP	Required.
	Location of the data set	UNIT	Required only if you want more units.
		LABEL	Required only if the data set does not have standard labels.
	Data attributes	DCB	Optional.
	Special Processing Option	DUMMY	Optional.
Cataloged Data Set	Data Set Information	DSNAME	Required.
		DISP	Required.
	Location of the data set	UNIT	Optional.
		VOLUME	May be required if you want to begin processing with a volume after the first.
		LABEL	Required only if the data set does not have standard labels.
	Data attributes	DCB	Optional.
	Special Processing Options	SEP	Either parameter can be used.
		AFF	
		DUMMY	Optional.
Kept Data Set	Data set information	DSNAME	Required.
		DISP	Required.
	Location of the data set	UNIT	Required.
		VOLUME	Required.
		LABEL	Required only if the data set does not have standard labels.
	Data attributes	DCB	Optional.
	Special Processing Options	SEP	Either parameter can be used.
		AFF	
		DUMMY	Optional.

Retrieving an Existing Data Set

You can retrieve an existing data set located in unit record devices (card reader and paper tape reader), the input stream, magnetic tape, or direct access devices. Data sets on magnetic tape and direct access devices are retrieved in similar ways depending on whether they were passed, cataloged or kept when you created them.

This chapter describes how to retrieve:

- Unit record data sets (card reader and paper tape reader).
- Input stream data sets.
- Passed data sets.
- Cataloged data sets.
- Kept data sets.

Table 15 shows the parameters required to retrieve a data set. Please fold out Table 15 while reading this chapter.

Unit Record Devices

You can retrieve an input data set on either a card reader or a paper tape reader. As shown in Table 15, only the UNIT parameter is required to indicate the device you want for the data set. The DSNAME and DISP parameters are not used, because data sets on unit record devices are always temporary and cannot be retrieved by another DD statement in your job.

This section is summarized in Table 67 of Appendix D.

Location of the Data Set

The location of the data set is given by the UNIT parameter.

UNIT: The format of the UNIT parameter is:

```
      ┌unit address┐
UNIT=<│device type │>
      └group name  ┘
```

where:

unit address
: is the actual machine address of the device. For example, UNIT=00D. You should not specify the address unless you are sure you want that specific device.

device type
: corresponds to the model number of the I/O device. Coding a device type provides you with a certain degree of device independence in that your data set may be placed in any of a number of devices of the same type. For example, if you code

```
UNIT=1442
```

your data set can be placed by the operator in any 1442 Card Reader in the system. The following device types can be specified:

Device Type	Description
1442	1442 Card Read Punch
2520	2520 Card Read Punch
2540	2540 Card Read Punch (Read feed)
2671	2671 Paper Tape Reader

group name
is the name of a collection of devices, selected by your installation during system generation. For example, your installation might select the name CARD for all card readers in the configuration. If you do not care which card reader is used for your data set, you would then code

```
UNIT=CARD
```

Your manager or supervisor should tell you which group names were generated for your installation.

Data Attributes

The DCB parameter allows you to specify attributes for your data set when your program is to be executed rather than when it is compiled. Any applicable attributes not specified in your program must be specified with the DCB parameter. However, in most cases, your compiler will provide a default value for an attribute if you do not specify it in the program or in the DD statement. Other attributes are always given a fixed value by the compiler and you do not have to specify them at all. For example, you can select a buffering technique with the assembler, but all other IBM compilers select one for your use when it is needed.

DCB: You can use the DCB parameter to directly specify the DCB attributes of your data set or to copy those DCB attributes specified in a DD statement for another data set. The format of the DCB parameter for specifying the attributes is:

```
DCB=(list of attributes)
```

The attributes in the list are coded in the form of keyword subparameters separated by commas; for example,

```
DCB=(BLKSIZE=300,LRECL=100)
```

Table 16. DCB Subparameters for Card Reader

DCB Subparameter	ALGOL	Assembler	COBOL E	COBOL F	ANS COBOL [3]	FORTRAN E	FORTRAN G & H	PL/I F
BFALN=		F or D [1,2]						
BFTEK=		S or E (QSAM only) [1,2]						
BLKSIZE=	number of bytes	number of bytes [1]		number of bytes [1]	number of bytes [1]	number of bytes	number of bytes	number of bytes [1]
BUFL=		number of bytes [1]						
BUFNO=		number of buffers [1]	number of buffers [1]	number of buffers	number of buffers	1 or 2	1 or 2	number of buffers [1]
EROPT=		ABE [1]	ABE	ABE	ABE			
HIARCHY=		0 or 1 [1]						
LRECL=	number of bytes	number of bytes [1]				number of bytes	number of bytes	number of bytes [1]
MODE=		C or E [1]	C or E	C or E	C or E	C or E	C or E	C or E [1]
NCP=		number of channel programs (BSAM only) [1]						number of channel programs (BSAM only) [1]
OPTCD=		[C] [1]	[C]	[C]	[C]	[C]	[C]	[C]
RECFM=	F[B][A]	U[A/M] or V[B/S/BS][A/M] or F[B/S/BS][A/M] [1]				U[A] or F[B][A/M]	U[A/M] or F[B][A/M]	U[A/M] or V[B/S/BS][A/M] or F[B/S/BS][A/M] [1]
STACK=		1 or 2 [1]	1 or 2	1 or 2	1 or 2	1 or 2	1 or 2	1 or 2

[1] This function can be specified in your program rather than in the DD statement.

[2] For QSAM, you must specify both BFALN and BFTEK on the DD statement, or omit both.

[3] American National Standard COBOL.

Table 17. DCB Subparameters for Paper Tape Reader

DCB Subparameter	ALGOL	Assembler	COBOL E	COBOL F	ANS COBOL [3]	FORTRAN E	FORTRAN G & H	PL/I F
BFALN=		F or D [1,2]						
BFTEK=		S or E (QSAM only) [1,2]						
BLKSIZE=	number of bytes	number of bytes [1]		number of bytes [1]	number of bytes [1]	number of bytes	number of bytes	number of bytes [1]
BUFL=		number of bytes [1]						
BUFNO=		number of buffers [1]	number of buffers [1]	number of buffers	number of buffers	1 or 2	1 or 2	number of buffers [1]
CODE=		I,F,B,C,A,T, or N [1]						I,F,B,C,A,T or N [1]
EROPT=		ABE [1]	ABE	ABE	ABE			
HIARCHY=		0 or 1 [1]						
LRECL=	number of bytes	number of bytes [1]				number of bytes	number of bytes	number of bytes [1]
NCP=		number of channel programs (BSAM only) [1]						
OPTCD=		[C] [1]	[C]	[C]	[C]	[C]	[C]	[C]
RECFM=	E[B][A]	U[A/M] or V[B/S/BS][A/M] or F[B/S/BS][A/M] [1]				U[A] or V[B][A] or F[B][A/M]	U[A/M] or V[B][A/M] or F[B][A/M]	U or F [1]

[1] This function can be specified in your program rather than in the DD statement.

[2] For QSAM, you must specify both BFALN and BFTEK on the DD statement, or omit both.

[3] American National Standard COBOL.

The valid subparameters that can be used with each compiler for the card reader and paper tape reader are shown in Tables 16 and 17, respectively. Underscored items are those default values selected if you omit the subparameter. Default values are not shown where the attribute can either be specified in your program, or in the DD statement. If values for a given subparameter are not shown, it is either specified in your source program or given a fixed value by the compiler. A glossary of DCB subparameters is given in Appendix B. Code only those parameters that apply to your compiler as shown in Tables 16 and 17. The format of the DCB parameter for copying the DCB parameter of a previous DD statement in your job is:

```
DCB=({*.ddname                          }[,list of attributes])
     {*.stepname.ddname                 }
     {*.stepname.procstepname.ddname    }
```

Replace "ddname" with the name of the DD statement whose DCB parameter you want to copy. Replace "stepname" with the name of the EXEC statement of the step that contains that DD statement. If the DD statement you want to copy is contained in the same job step, omit the stepname, i.e.,

```
DCB=*.ddname
```

The following example shows how to code the DCB parameter to copy the DCB parameter of a DD statement in a previous step:

```
//STEP2 EXEC...
//DD1 DD...,DCB=(BLKSIZE=1600,LRECL=80)
.
.
.
//STEP4 EXEC...
//READ DD...,DCB=*.STEP2.DD1
.
.
.
```

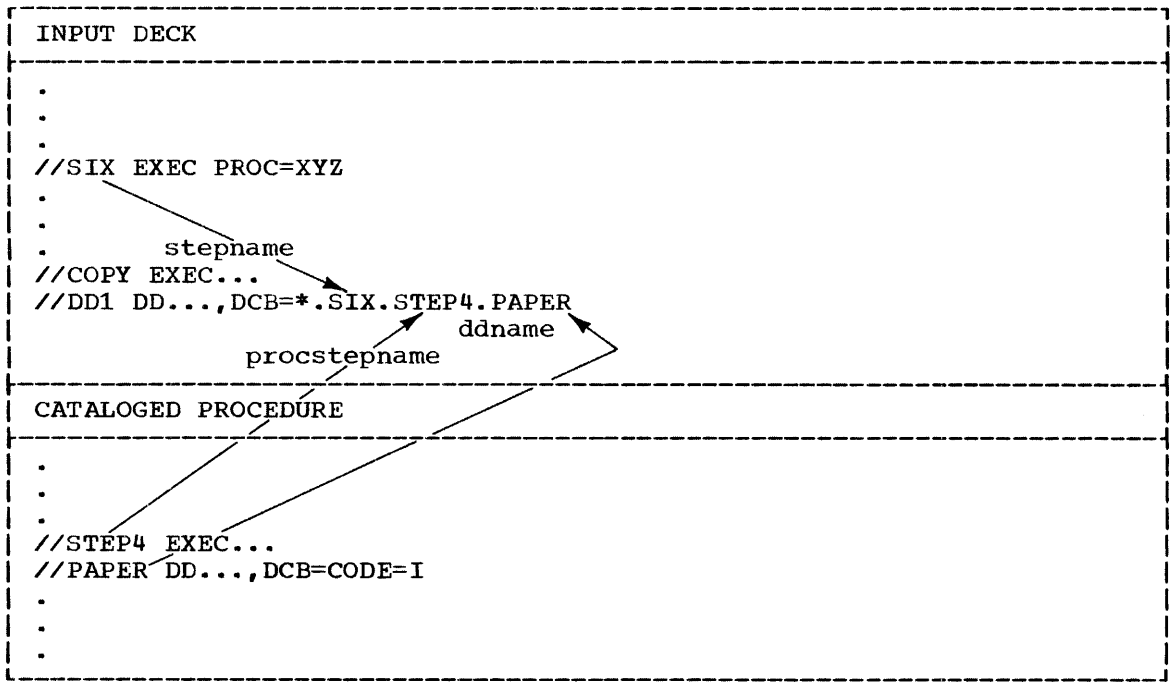
The following example shows how to code the DCB parameter of a previous DD statement in the same step:

```
.
.
.
//STEPD EXEC...
//DDA DD...,DCB=RECFM=F
.
.
.
//DDC DD...,DCB=*.DDA
.
.
.
```

If you want to copy the DCB parameter of a DD statement contained in a cataloged procedure step, you must give both the name of the job step that invokes the procedure and the procedure step name, i.e.,

```
DCB=*.stepname.procstepname.ddname
```

The first part of the following example shows your input deck; the second part shows a cataloged procedure called by the SIX EXEC statement in your deck.



If you want to modify the DCB subparameters you are copying add the new subparameters to the reference. The subparameters you specify will override the corresponding copied subparameters. For example:

```
.
.
.
//OUT EXEC...
//READ1 DD...,DCB=(BLKSIZE=160,LRECL=80,OPTCD=C,MODE=E,RECFM=F)
//READ2 DD...,DCB=(*.READ1,MODE=C,STACK=2)
```

Special Processing Option

You can use the DUMMY parameter to bypass I/O operations on your data set.

DUMMY: The DUMMY parameter allows you to bypass I/O operations and data set disposition. When your processing program asks to read the dummy data set, the read request is recognized, but no data is transmitted. This facility is particularly useful when you are debugging your program because it saves you processing time. For example, you can suppress the reading of a data set until you are sure your program is going to process it correctly.

The DUMMY parameter is a positional parameter. That means that it must be coded first in the operand field. For example:

```
//INPUT DD DUMMY,UNIT=2671,DCB=CODE=I
//R20 DD DUMMY,UNIT=1442
```

If you wish, you can omit all other parameters required to define your data set except DCB. (If you were not going to write the DCB parameter you do not have to write it.) For example, the two DD statements shown above could be coded as follows:

```
//INPUT DD DUMMY,DCB=CODE=I
//R20 DD DUMMY
```

When you are ready to perform I/O operations on your data set, simply omit the DUMMY parameter and code all parameters required to retrieve the data set. For example,

```
//INPUT DD UNIT=2671,DCB=CODE=I
//R20 DD UNIT=1442
```


Input Stream

You can place your input data set in the input stream following the other JCL statements in the job step. If the EXEC statement for the job step requests a program, you can include the input data for that program in the input stream. If the EXEC statement requests a cataloged procedure, you can include the data for each procedure step in the input stream.

The data must be preceded by a DD statement and followed by a delimited (/*) statement. As shown in Table 15, the DD statement must contain an * or DATA in the operand field. The * parameter means that data follows. The DATA parameter must be used if the data contains JCL statements.

This section is summarized in Table 68 of Appendix D.

The following considerations apply to data in the input stream in PCP and in MFT and MVT.

Input Stream Data in PCP:

1. The input stream must be on a card reader or magnetic tape.
2. You can include only one input stream data set for each job step or procedure step.
3. The DD* or DD DATA statement for the data set must be the last DD statement for the job step or procedure step.
4. Do not code any other parameter in the DD statement.
5. The data must consist of unblocked, 80-character records.
6. The characters in the records must be coded in BCD or EBCDIC.
7. The data must be followed by a delimiter (/*) statement.

Input Stream Data in MFT and MVT:

1. The input stream must be on a card reader, magnetic tape or direct access device. (The device must be supported by QSAM.)

2. You can include more than one input stream data set for each job step or procedure step.

3. If the data set contains JCL statements you must precede it with a DD DATA statement and follow it with a delimiter statement.

4. If the data set does not contain JCL statements, you can either precede it with a DD* statement or follow it with a /* statement, or both. If you omit the DD* statement the system provides one. Although it is not necessary, it is good practice to both precede the data with a DD* statement and to follow it with a /* statement.

5. The characters in the records must be coded in BCD or EBCDIC.

6. You may code the DCB subparameters BLKSIZE and BUFNO in the DD statement. (See "Data Attributes.")

Location of the Data Set

To indicate that the data set follows in the input stream, you must code the * or the DATA parameter.

THE * PARAMETER: To specify that data follows in the input stream, code an asterisk (*) in the operand field. For example:

```
//SYSIN DD *
```

The data must not contain JCL statements.

The following example shows a two step job named ELLEN. Each of the two steps in the job contains a data set in the input stream. The system can have PCP, MFT or MVT.

```
//ELLEN JOB
//STEP1 EXEC PGM=ONE
.
.
.
//INPUT DD *
     data records
/*
//STEP2 EXEC PGM=TWO
.
.
.
//SYSIN DD *
     data records
/*
```

Using the preceding example, assume a system with MFT or MVT. You can include more than one input stream data set for each job step. For example if you have two data sets for STEP1, code:

```
//ELLEN JOB
//STEP1 EXEC PGM=ONE
.
.
.
//INPUT DD *
     data records
/*
//READ DD *
     data records
/*
//STEP2 EXEC PGM=TWO
.
.
.
//SYSIN DD *
     data records
/*
```


In the following example, your EXEC statement requests the TOP cataloged procedure. TOP has three procedure steps named AA, BB, and CC. You are including an input stream data set for each procedure step. You are also including a DD statement named WORK for procedure step BB. The system can have PCP, MFT, or MVT.

```
//STEPY EXEC TOP
//AA.INPUT DD *
     data records
/*
//BB.WORK DD UNIT=2311,VOLUME=PRIVATE,SPACE=(TRK,200)
//BB.IN DD *
     data records
/*
//CC.SYSIN DD *
     data records
/*
```

Using the preceding example, assume a system with MFT or MVT. You can include more than one input stream data set for each procedure job step. For example, if you want to include two data sets for AA, three for BB, and one for CC, code:

```
//STEPY EXEC TOP
//AA.INPUT1 DD *
     data records
/*
//AA.INPUT2 DD *
     data records
/*
//BB.WORK DD UNIT=2311,VOLUME=PRIVATE,SPACE=(TRK,(200))
//BB.IN1 DD *
     data records
/*
//BB.IN2 DD *
     data records
/*
//BB.IN3 DD *
     data records
/*
//CC.SYSIN DD *
     data records
/*
```

Note: For information on how to modify cataloged procedures, refer to "Part III: Cataloged and In-stream Procedures."

DATA: To specify that data follows in the input stream, code DATA in the operand field. For example:

```
//SYSIN DD DATA
```

The data can contain any JCL statements except the delimiter (/*) statement, because the delimiter statement marks the end of the input data set.

For example, the following DD statement indicates that the data records that follow contain JCL statements.

```
//IN DD DATA
     data records
/*
```

In the following example an EXEC statement requests a cataloged procedure named ROSE. ROSE has two steps named PINK and TEA. You are including an input stream data set that has JCL statements for PINK and an input stream data set without JCL statements for TEA. You are also including a DD statement named BLANK for procedure step PINK. The system can have PCP, MFT, or MVT.

```
//STEPZ EXEC PROC=ROSE
//PINK.BLANK DD UNIT=2400,LABEL=(,BLP)
//PINK.IN DD DATA
     data records
/*
//TEA.INTAKE DD *
     data records
/*
```

Using the preceding example, assume a system with MFT or MVT. You can include more than one input stream data set for each procedure job step.

For example, if you want to include a data set without JCL statements for PINK and a data set with JCL statements for TEA, code:

```
//STEPZ EXEC PROC=ROSE
//PINK.BLANK DD UNIT=2400,LABEL=(,BLP)
//PINK.IN DD DATA
     data records
/*
//PINK.IN2 DD *
     data records
/*
//TEA.INTAKE DD *
     data records
/*
//TEA.GULP DD DATA
     data records
/*
```


Data Attributes

In MFT and MVT input stream data sets are written by the system onto direct access devices so that the data can be retrieved rapidly when required by the processing program. As the data is written onto the direct access device the data is blocked. The block size and number of buffers used for blocking the data are default values selected by your installation. Your manager or supervisor can tell you what those values are. If you want to specify shorter blocks or less buffers you can do so with the BLKSIZE and BUFNO subparameters of the DCB parameter. You cannot request larger blocks or more buffers.

DCB: Only the BLKSIZE and BUFNO subparameters can be coded in a DD* or DD DATA statement for MFT or MVT. You must find out what your installation's default values are and then you can specify smaller values, if applicable to your data set.

For example, assume that your installation's default values are BLKSIZE=2400 and BUFNO=4. If you want to specify a block size of 1600 for your data set, code

```
//SYSIN DD DATA,DCB=BLKSIZE=1600
```

If you want to specify two buffers for your data set, code:

```
//IN DD *,DCB=BUFNO=2
```

If you want to specify a blocksize of 1200 and three buffers, code:

```
//INPUT DD *,DCB=(BUFNO=3,BLKSIZE=1200)
```

Passed Data Set

A passed data set is one you defined with the DISP=(,PASS) parameter. The most common type of passed data set is a "temporary-for the duration of the job" data set, although you can also have nontemporary passed data sets.

A passed data set can only be retrieved in the same job it was created. However, if you have a nontemporary passed data set, you can change its disposition to KEEP or CATLG when you retrieve it so that it can be retrieved in other jobs.

The parameters shown in Table 15 are used to retrieve passed data sets.

This section is summarized in Table 69 of Appendix D.

Data Set Information

The DSNAME and DISP parameters are required. DSNAME identifies the data set. DISP specifies if you are retrieving or extending the data set.

DSNAME: You can identify the passed data set in one of two ways:

1. Giving its name exactly as it appeared in the DD statement where you defined it, or
2. Making a backward reference to the DD statement that defined the data set.

Both ways are equally valid, but the second is more useful because if you decide to change the data set's name you only have to change it in the original DD statement.

1. The format of the DSNAME parameter for identifying the passed data set (or member of a partitioned data set) by name is:

```
          ⎧dsname                  ⎫
          ⎪dsname(membername)      ⎪
DSNAME=   ⎪&&name                  ⎪
          ⎪&&name(membername)      ⎪
          ⎪&&name                  ⎪
          ⎩&name(membername)       ⎭
```

   Code the DSNAME parameter exactly as you coded it in the original DD statement. For example, if you coded

   DSNAME=&&AREA

   you must again code

   DSNAME=&&AREA

2. The format of the DSNAME parameter for identifying the passed data set with a backward reference is:

```
          ⎧*.ddname                    ⎫
DSNAME=   ⎪*.stepname.ddname           ⎪
          ⎩*.stepname.procstepname.ddname⎭
```

Replace "ddname" with the name of the DD statement, where the passed data set is defined. Replace "stepname" with the name of the EXEC statement of the step that has the earlier DD statement. If the earlier DD statement is contained in the same step, omit the stepname, i.e.,

```
DSNAME=*.ddname
```

When the earlier DD statement is contained in a cataloged procedure step, you must give both the name of the job step that invokes the procedure and the procedure step name, i.e.,

```
DSNAME=*.stepname.procstepname.ddname
```

The following example shows how to code the DSNAME parameter to identify a member of a passed data set defined in a previous step.

```
//STEP2 EXEC...
//TEMP DD DSNAME=&&ALPHA(FIRST),DISP=(,PASS),...
 .
 .
 .
//STEP4 EXEC...
//USE DD DSNAME=*.STEP2.TEMP,...
```

The following example shows how to code the DSNAME parameter to identify a passed data set defined in a cataloged procedure step. The first part of the example shows your input deck; the second part shows a cataloged procedure called by the STEP3 EXEC statement in your deck.

```
INPUT DECK
 .
 .
 .
//STEP3 EXEC PROC=PROC7
 .
 .      stepname
 .
//STEP5 EXEC
//DDA DD DSNAME=*.STEP3.NOVA.FILE
                        ddname
CATALOGED PROCEDURE
 .      procstepname
 .
 .
//NOVA EXEC PGM=N
//FILE DD DSNAME=LEFT,DISP=(,PASS),...
```

DISP: You can code the DISP parameter in one of two ways depending on whether other steps in the job will also use the data set or whether this is the last time you are using the data set.

1. The format of the DISP parameter for passing the data set to a subsequent job step is:

```
DISP=(OLD,PASS[,PASS  ])
              [,DELETE]
              [,CATLG ]
              [,KEEP  ]
```

You must code OLD. The PASS subparameter indicates that the data set can be used by later steps in the job. The third subparameter tells the system what is to be done with the data set if the step ABENDs. You can specify PASS or DELETE for both temporary and nontemporary data sets. PASS is assumed if you omit the third subparameter. If the data set is nontemporary you can specify KEEP or CATLG.

Note: When you are retrieving members of a partitioned data set, the disposition (PASS, DELETE, CATLG, KEEP) you specify applies to the entire data set and not to the one member. If you want to delete a particular member of a nontemporary partitioned data set you must use the IEHPROGM utility program. The entire temporary partitioned data set is deleted at the end of the job.

2. The format of the DISP parameter for the last time you use a passed temporary data set in a job is:

```
DISP=(OLD,DELETE,DELETE)
```

You must code OLD. A passed temporary data set will always be deleted at the end of the job so you do not have to code the DELETE subparameter.

The format of the DISP parameter for the last time you use a passed nontemporary data set in a job is:

```
DISP=(OLD[,KEEP  ][,KEEP  ])
         [,DELETE][,DELETE]
         [,CATLG ][,CATLG ]
         [,      ]
```

You must code OLD. At the end of the step, the data set will also be kept unless you specify DELETE or CATLG. If the step ABENDs, the data set will also be kept unless you specify DELETE or CATLG.

In the following example, two data sets named &&DATA(MEM1) and LIST are defined in the first step of a job. In the second step, &&DATA(MEM1) is retrieved and passed again. In the third step, LIST is retrieved and passed again. In the fourth step, &&DATA(MEM1) is retrieved and all of &&DATA is deleted, and LIST is retrieved and cataloged.

```
//MYJOB JOB
//STEP1 EXEC PGM=UNO
//DS1   DD   DSNAME=&&DATA(MEM1),DISP=(,PASS),UNIT=2311,
//           VOLUME=PRIVATE,SPACE=(TRK,(25,,3))
//DS2   DD   DSNAME=LIST,DISP=(,PASS),UNIT=2400
.
.
.
//STEP2 EXEC PGM=DOS
//DS3   DD   DSNAME=*.STEP1.DS1,DISP=(OLD,PASS)
.
.
.
//STEP3 EXEC PGM=TRES
//DS4   DD   DSNAME=LIST,DISP=(OLD,PASS)
.
.
.
//STEP4 EXEC PGM=CUATRO
//DS5   DD   DSNAME=*.STEP1.DS1,DISP=OLD
//DS6   DD   DSNAME=*.STEP1.DS2,DISP=(OLD,CATLG)
```


Location of the Data Set

The system obtains unit and volume information from the original DD statement. The UNIT and LABEL parameters are only needed in the following cases:

1. The UNIT parameter is used if you want more devices allocated to the data set than were requested the last time the data set was passed.

2. The LABEL parameter is used if the data set does not have standard labels.

UNIT: The UNIT parameter should be used only if the passed data set is a multivolume data set. The system assigns the same number of devices to the data set as it did in a previous step. If you want more devices assigned to the data set, code in the unit count subparameter the total number of devices needed.

The format of the UNIT parameter for passed data sets is:

```
UNIT=(,unit count)
```

Replace "unit count" with the number of devices needed for the data set. Remember that you should not request more devices than there are volumes.

In the following example a data set named &&JUNK is defined in the first step. A maximum of five volumes are required for &&JUNK and they are to be mounted in succession on two units. In the second step, you retrieve &&JUNK, but request that four units be used.

```
//SAM   JOB
//STEPA EXEC PGM=X
//DATA  DD   DSNAME=&&JUNK,DISP=(,PASS),UNIT=(2314,2)
//            VOLUME=(PRIVATE,,,5),SPACE=(CYL,(20,15))
.
.
.
//STEPB EXEC PGM=Y
//GET   DD   DSNAME=*.STEPA.DATA,DISP=(OLD,PASS),UNIT=(,4)
.
.
.
```

LABEL: The LABEL parameter is used only if the passed data set does not have standard labels. You must code the same label type you specified when you defined the data set. No other subparameters are allowed. The format of the LABEL parameter for passed data sets is:

Code:

SUL
: if the data set has standard and user labels.

NL
: if the data set has no labels.

NSL
: if the data set has nonstandard labels.

BLP
: to bypass label processing.

In the following example, a data set named BANK is defined in the first step. BANK has nonstandard labels. BANK is retrieved and passed in the second step.

```
//MAC   JOB
//STEPA EXEC PGM=ZYZ
//DDAA  DD   DSNAME=BANK,DISP=(,PASS),UNIT=2400,LABEL=(,NSL)
.
.
.
//STEPB EXEC PGM=WZW
//DDAB  DD   DSNAME=*.STEPA.DDAA,DISP=(OLD,PASS),LABEL=(,NSL))
```

Data Attributes

The DCB parameter is used only if the data set does not have standard labels and the DD statement that defined the passed data set contains the DCB parameter.

DCB: You must specify the DCB parameter in one of two ways:

1. Copy the DCB parameter exactly as it appeared in the DD statement where you defined the passed data set, or

2. Make a backward reference to the DD statement that defined the data set.

Making a backward reference reduces the possibliity of error in copying the parameter.

The format of the DCB parameter for copying the attributes of the passed data set is:

```
DCB=(list of attributes)
```

Code exactly the same DCB subparameters you coded in the DD statement where the passed data set is defined. You may not add, change, or delete subparameters. For example, if you coded

```
DCB=(BLKSIZE=800,LRECL=80)
```

you must again code

```
DCB=(BLKSIZE=800,LRECL=80)
```

The format of the DCB parameter for making a backward reference is:

```
DCB=({*.ddname                          }[,list of attributes])
     {*.stepname.ddname                 }
     {*.stepname.procstepname.ddname    }
```

Replace "ddname" with the name of the DD statement whose DCB parameter you want to copy. Replace "stepname" with the name of the EXEC statement of the step that contains that DD statement. If the DD statement you want to copy is contained in the same job step, omit the stepname, i.e.,

```
DCB=*.ddname
```

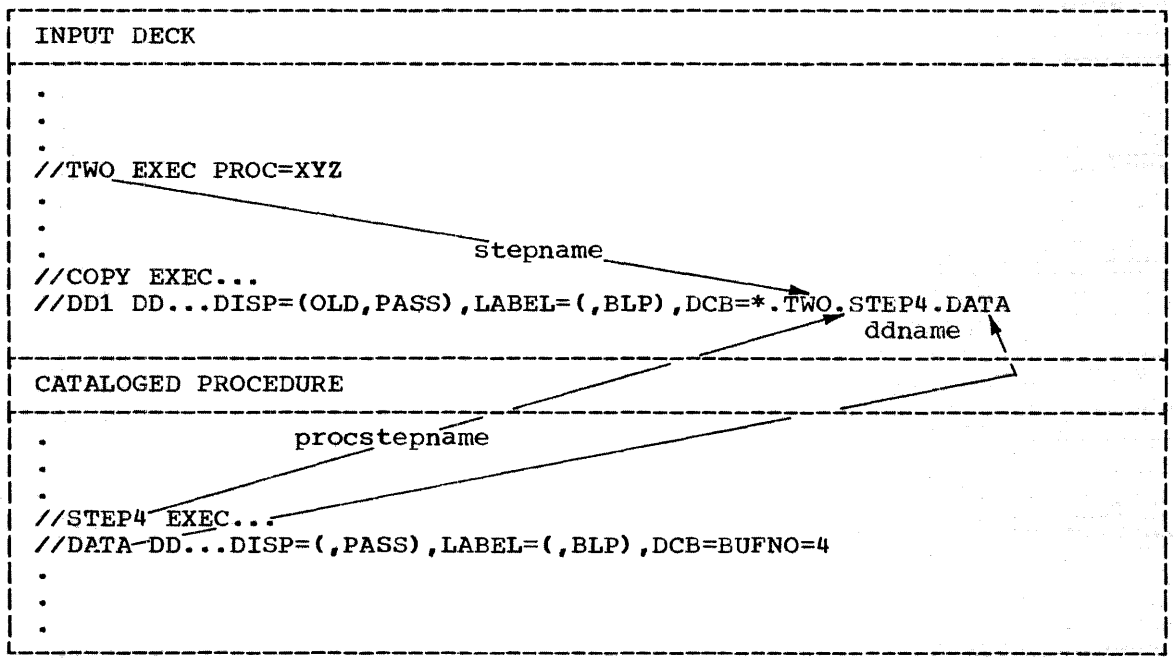
If you want to copy the DCB parameter of a DD statement contained in a cataloged procedure step, you must give both the name of the job step that invokes the procedure and the procedure step, i.e.,

```
DCB=*.stepname.procstepname.ddname
```

The following example shows how to code the DCB parameter to copy the DCB parameter of a DD statement in a previous step:

```
//STEP2 EXEC...
//DD1   DD...DISP=(,PASS),LABEL=(,NL),DCB=BLKSIZE=1600
.
.
.
//STEP4 EXEC...
//COPY  DD...DISP=(OLD,PASS),LABEL=(,NL),DCB=*.STEP2.DD1
.
.
.
```

The following example shows how to code the DCB parameter to copy the attributes of a passed data set defined in a cataloged procedure step. The first part of the example shows your input deck, the second part shows a cataloged procedure called by the TWO EXEC statement in your deck.



Special Processing Option

You can use the DUMMY parameter to bypass operations and data set disposition on a sequential data set (magnetic tape or direct access).

DUMMY: The DUMMY parameter allows you to bypass I/O operations and data set disposition. When your processing programs ask to read the dummy data sets, the read request is recognized, but no data is transmitted. This facility is particularly useful when you are debugging your program because it saves you processing time. For example, you can suppress the reading of a data set until you are sure your program is going to process it correctly.

The DUMMY parameter is a positional parameter. That means that it must be coded first in the operand field. For example:

```
//INPUT DD DUMMY,DSNAME=*.STEP1.DDA,DISP=(OLD,PASS),LABEL=(,NL),
//         DCB=BLKSIZE=500
//READ DD DUMMY,DSNAME=*.STEP4.ABC,DISP=(OLD,PASS)
```

If you wish, you can omit all other parameters required to define your data set except DCB. (If you were not going to write the DCB parameter you do not have to write it.) For example, the two DD statements shown above could be coded as follows:

```
//INPUT DD DUMMY,DCB=BLKSIZE=500
//READ DD DUMMY
```

When you are ready to perform I/O operations on your data set, simply omit the DUMMY parameter and code all parameters required to retrieve the data set. For example,

```
//INPUT DD DSNAME=*.STEP1.DDA,DISP=(OLD,PASS),LABEL=(,NL)
//READ DD DSNAME=*.STEP4.ABC,DISP=(OLD,PASS)
```


Cataloged Data Set

A cataloged data set is a nontemporary data set defined with the DISP=(,CATLG) parameter or cataloged by the IEHPROGM utility program. The parameters shown in Table 15 are used to retrieve a cataloged data set.

This section is summarized in Table 70 of Appendix D.

Data Set Information

The DSNAME and DISP parameters are required.

DSNAME: The format of the DSNAME parameter is:

```
DSNAME={dsname                 }
       {dsname(membername)     }
       {groupname(number)      }
```

Replace the term "dsname" with name of the cataloged data set. If the name is a fully qualified name you must give all levels of qualification. For example, if you created the data set as

```
DSNAME=PLUS
```

you must again code

```
DSNAME=PLUS
```

If you coded

```
DSNAME=SYST.GLOBE
```

you must again code

```
DSNAME=SYST.GLOBE
```

If you are retrieving a member of a partitioned data set, you must code the member name in parentheses after the data set name. For example,

```
DSNAME=ORB(MOON)
```

or

```
DSNAME=SYST.GLOBE(SUN)
```

If you are retrieving a member of a generation data group you must use the form

```
DSNAME=groupname(number)
```

Replace "groupname" with the name of the generation data group. Replace "number" with the number of the generation you are retrieving. Remember that the most recent generation is 0, the previous generation is -1, and so on. For example, if you want to retrieve the most recent generation of the ABC.FILE generation data group, code

```
DSNAME=ABC.FILE(0)
```

DISP: The format of the DISP parameter is:

```
                 ┌,UNCATLG┐ ┌,UNCATLG┐
DISP=({OLD }[    ├,DELETE ┤ ├,DELETE ┤])
       {SHR }    └,       ┘ └,       ┘
```

You must code either OLD or SHR. OLD indicates that you are only retrieving the data set. SHR indicates that you are operating under MFT or MVT and that concurrent jobs can also retrieve this data set at the same time.

The second positional parameter tells the system what is to be done with the data set at the end of the job step. If you omit this subparameter, the data set remains cataloged. If you code UNCATLG the data set will be uncataloged but not deleted. (The next time you retrieve an uncataloged data set you must follow the directions given for "Kept Data Sets.") If you code DELETE the data set will be deleted (and uncataloged).

The third positional subparameter tells the system what is to be done with the data set if the step ABENDs. You can specify UNCATLG or DELETE.

Note: When you are retrieving a member of a partitioned data set, the disposition (UNCATLG, or DELETE), you specify applies to the entire data set and not to the one member. If you want to delete a particular member, you must use the IEHPROGM utility program. In the following example, you are retrieving a cataloged data set named HULA.HOOP.

```
//DD9 DD DSNAME=HULA.HOOP,DISP=OLD
```

In the following example you are retrieving a member of a cataloged data set named SYS1.FORTLIB. You are operating under MFT or MVT and other jobs can also retrieve SYS1.FLORLIB at the same time.

```
//READ DD DSNAME=SYS1.FORTLIB(USERRTN),DISP=SHR
```

In the following example, you are retrieving a cataloged data set named AREA. After using it you want to uncatalog the data set.

```
//DDZ DD DSNAME=AREA,DISP=(OLD,UNCATLG)
```


Location of the Data Set

The system obtains unit and volume information from the catalog. The UNIT, VOLUME and LABEL parameters are only needed in the following cases:

1. If you want more than one unit assigned to your data set, use the UNIT parameter.
2. If you want to defer mounting of the volumes, use the UNIT parameter.
3. If you want your data set assigned to the same devices assigned to a data set defined earlier in the job step, request unit affinity with the UNIT parameter.
4. If you want the volume to be private, use the VOLUME parameter.
5. If the cataloged data set does not have standard labels you must use the LABEL parameter to specify the LABEL type.

UNIT: The UNIT parameter should be used only if you want to request more than one unit for your data set or if you want unit affinity.

Requesting Units: The system assigns one device to your data set. If you need more devices use the unit count subparameter or the P subparameter.

```
UNIT=(,[unit count][,DEFER])
      [P          ]
```

unit count
indicates how many units you want assigned to the data set. You can specify a maximum of 59 units per DD statement. Make sure that your system has at least the number of units you specify. Otherwise an error will result. If you specify fewer units than the number of volumes in your data set, only the same number of volumes as there are units can be mounted at the same time. (If you request less units than volumes code the PRIVATE subparameter in the VOLUME parameter.) If you specify the same number of units as there are volumes, all volumes can be mounted at the same time.

P
specifies parallel mounting. When you request parallel mounting the system assigns to the data set as many units as there are serial numbers in the catalog entry.

DEFER requests the system to assign the required tape drives or direct access devices to the data set and to defer the mounting of the volume or volumes until the processing program attempts to open the data set. If you are running under MFT or MVT, operating efficiency may decrease if you specify DEFER for a tape data set. This is because if you code DEFER, the system issues a mount message to the operator when the processing program attempts to use the data set and then waits until the volume is mounted. If you do not code DEFER, the mount message is issued when the device is assigned and there is no waiting for the operator. By the time your program tries to use the data set, it is very likely that the operator has already mounted the volume and that the system will not have to wait for him.

Unit Affinity: To conserve the number of units used in a job step, you can request that the cataloged data set be assigned to the same device or devices as assigned to an earlier data set in the same step. This technique, known as unit affinity, indicates that you want the volumes on which the cataloged data set resides and the volumes used by the earlier data set to be mounted on the same devices in sequential order. Unit affinity implies deferred mounting for one of the volumes since both volumes cannot be mounted at the same time.

Before you request unit affinity, make sure that both data sets use the same type of removable volumes; for example, both data sets could be on 2314 volumes, or both on 2400 volumes. It does not make sense to request unit affinity where both data sets are on different types of volumes. The format of the UNIT parameter for requesting unit affinity is:

```
UNIT=AFF=ddname
```

Replace the term "ddname" with the name of an earlier DD statement in the job step. Whenever it is required, the volume of the earlier data set will be demounted and the cataloged data set will be mounted on the same unit. If the ddname refers to a DD statement that defines a dummy data set, the data set defined on the DD statement requesting unit affinity is assigned a dummy status.

When unit affinity is requested for two data sets that reside on different 2321 volumes, the data sets are assigned the same device but can be assigned different bins.

In the following example, the DD statement named DD2 requests that the system assign the same number of units to this data set as it assigns to the data set defined on the statement named DD1. Since DD1 requests two devices, these two devices are assigned to the data set defined by DD2.

```
//STEPC EXEC PGM=OVER
//DD1   DD   UNIT=(2400,2),VOLUME=(,,,2)
//DD2   DD   DSNAME=ALPHA.OMEGA,DISP=OLD,UNIT=AFF=DD1
```

VOLUME: The VOLUME parameter is needed only when one or more of the following conditions are met:

1. You want the volume on which the cataloged data set resides to be private. A private volume is one that cannot be allocated to a new temporaty data set that makes a nonspecific request for a public volume.

2. You want the private volume to remain mounted at the end of the job step.

3. You want to begin processing with a certain volume when you are retrieving or extending a multivolume cataloged data set.

The format of the VOLUME parameter for cataloged data sets is:

```
VOLUME=([PRIVATE][,RETAIN] [,sequence])
                 [,     ]
```

The PRIVATE subparameter means that you want the volume(s) on which the cataloged data set resides to be private. The RETAIN subparameter means that the volume will not be demounted at the end of the step. PRIVATE must be coded when you code RETAIN. (RETAIN has the same effect on volume mounting as coding PASS in the DISP parameter.) Private volumes are demounted at the end of the step unless it is an unremovable volume, such as a 2301, or unless it has been designated by your installation to be a permanently resident volume, for example, the system residence volume. It is recommended that you use the RETAIN subparameter whenever you want the volume to remain mounted at the end of the step.

For example, if you want the volume that contains the JOE.DOE.SMITH data set to be private, code,

```
//OLD DD DSNAME=JOE.DOE.SMITH,DISP=OLD,VOLUME=PRIVATE
```

Using the same example, if you want the volume to remain mounted at the end of the step, code:

```
//OLD DD DSNAME=JOE.DOE.SMITH,DISP=OLD,VOLUME=(PRIVATE,RETAIN)
```

Using the same example, if you want to defer mounting of the volumes, code:

```
//OLD DD DSNAME=JOE.DOE.SMITH,DISP=OLD,VOLUME=(PRIVATE,RETAIN),
//       UNIT=(,,DEFER)
```

When you are retrieving a multivolume cataloged data set, you can begin processing with other than the first volume of the data set by coding the sequence number subparameter. Replace "sequence" with a number from 1 to 255 that indicates the sequence number of the volume you want. For example, if you want the third volume of the cataloged data set,code

```
VOLUME=(,,3)
```

The sequence number is a positional subparameter and must follow the PRIVATE and RETAIN subparameters. If you omit the sequence number subparameter and the volume count subparameter follows, you must indicate its absence by a comma.

In the following example, you are retrieving a cataloged data set named SYSTEM.TWO. SYSTEM.TWO now resides on four volumes and you want to begin processing with second volume.

```
//IN DD DSNAME=SYSTEM.TWO,DISP=(OLD,CATLG),VOLUME=(,,2)
```

Using the same example, if you want the volumes to be private, code:

```
//IN DD DSNAME=SYSTEM.TWO,DISP=(OLD,CATLG),VOLUME=(PRIVATE,,2)
```

LABEL: The LABEL parameter is needed only if the cataloged data set does not have standard labels. You must code the same label type you specified when you defined the data set. No other subparameters are allowed.

The format of the LABEL parameter for cataloged data sets is:

code,

SUL
: if the data set has standard and user labels.

NL
: if the data set has no labels.

NSL
: if the data set has nonstandard labels.

BLP
: to bypass label processing.

In the following example, a cataloged data set named A.B.D is retrieved. A.B.D has nonstandard labels.

```
//INPUT DD DSNAME=A.B.D,DISP=OLD,LABEL=(,NSL)
```

Data Attributes

The DCB parameter is needed only if the cataloged data set is on magnetic tape and has nonstandard labels or no labels. Follow the instructions given for "Magnetic Tape" in the section "Creating a New Data Set." If the data set has standard labels, you can specify the subparameters shown in Tables 18, 19, 20, and 21.

Table 18. DCB Subparameters for Retrieving a Data Set on Magnetic Tape

DCB Subparameter	ALGOL	Assembler	COBOL E	COBOL F	ANS COBOL [3]	FORTRAN E	FORTRAN G & H	PL/I F
BFALN=		F or D [1,2]						
BFTEK=		S, E, or A (QSAM only) [1,2]						
BUFL=		number of bytes [1]						
BUFNO=		number of buffers [1]	number of buffers [1]	number of buffers [1]	number of buffers [1]	1 or 2	1 or 2	number of buffers [1]
EROPT=		ABE,ACC, or SKP [1]	ABE,ACC, or SKP	ABE,ACC, or SKP	ABE,ACC, or SKP			
HIARCHY=		0 or 1 [1]						
NCP=		number of channel programs (BSAM only) [1]						number of channel programs (BSAM only) [1]
OPTCD=		[C] [Z] [B] [1]	[C]	[C]	[C]	[C]	[C]	[C] or [Z]

[1] This function can be specified in your program rather than in the DD statement.

[2] For QSAM, you must specify both BFALN, and BFTEK on the DD statement, or omit both.

[3] American National Standard COBOL.

Table 19. DCB Subparameters for Retrieving a Sequential Data Set on Direct Access Device

DCB Subparameter	ALGOL	Assembler	COBOL E	COBOL F	ANS COBOL [3]	FORTRAN E	FORTRAN G & H	PL/I F
BFALN=		F or D [1,2]						
BFTEK=		S, E, or A (QSAM only) [1,2]						
BUFL=		number of bytes [1]						
BUFNO=		number of buffers [1]	number of buffers [1]	number of buffers [1]	number of buffers [1]	1 or 2	1 or 2	number of buffers [1]
EROPT=		ABE,ACC, or SKP [1]	ABE,ACC, or SKP	ABE,ACC, or SKP				
HIARCHY=		0 or 1 [1]						
NCP=		number of channel programs (BSAM only) [1]						

[1] This parameter can be specified in your program rather than in the DD statement.

[2] For QSAM, you must specify both BFALN and BFTEK on the DD statement, or omit both.

[3] American National Standard COBOL.

Table 20. DCB Subparameters for Retrieving a Direct Data Set

DCB Subparameter	Assembler	COBOL E	COBOL F	ANS COBOL [2]	FORTRAN E	FORTRAN G	PL/1 F
BFALN=	F or D [1]						
BFTEK=	R [1]						
BUFL=	number of bytes [1]						
BUFNO=	number of buffers [1]				1 or 2	1 or 2	number of buffers
HIARCHY=	0 or 1 [1]						
LIMCT=	number of tracks or blocks [1]						number of tracks or blocks
OPTCD=	[E][F][R/A] [1]						

[1] This function can be specified in your program rather than in the DD statement.

[2] American National Standard COBOL.

Table 21. DCB Subparameters for Retrieving a Partitioned Data Set

DCB Subparameter	Assembler
BFALN=	F or D [1]
BUFL=	number of bytes [1]
BUFNO=	number of buffers [1]
HIARCHY=	0 or 1 [1]
NCP=	number of channel programs [1]
OPTCD=	[C] [1]

[1] This parameter can be specified in your program rather than in the DD statement.

Special Processing Options

There are two special processing options for cataloged data sets:

1. You can request channel separation from other data sets in the same job step using either the SEP or the AFF parameter.

2. You can suppress I/O operations on a sequential data set (magnetic tape or direct access) using the DUMMY parameter.

SEP: When two or more data sets are to be used in a job step, processing time may be shortened by requesting that the system transmit data sets over separate channels. For example, it would be faster to have your input data set and your output data set on separate channels than to have them on the same channel.

You can request channel separation for data sets in each job step using the SEP parameter. If possible, the system will honor this request. It may not always be possible to honor the request for separation because given devices may not be available for allocation when the job step is executed or because there may not be enough devices of a given type on different channels for all the data sets that request separation. If channel separation is not requested, the system will assign any available channel that has the device specified with the UNIT parameter.

The format of the SEP parameter is:

```
SEP=(ddname,...)
```

Replace the terms "ddname" with the names of up to eight earlier DD statements in the same job step. This earlier DD statements can define any type of data set: new or existing, on magnetic tape or direct access, temporary or nontemporary, etc.

In the following example, you request that the data set defined by the FOUR DD statements be assigned to a channel other than the ones assigned to the data sets defined by the ONE and TWO DD statement. The data sets defined by ONE and TWO may or may not be on the same channel. The data set defined by THREE may or may not be on the same channel as any of the other three data sets.

```
//PROG1 EXEC PGM=PROCESS
//ONE   DD   UNIT=2400,VOLUME=PRIVATE,LABEL=(,NL)
//TWO   DD   SANAME=SYSTEM.USER.ABC,DISP=OLD
//THREE DD   DSNAME=HELP,DISP=(,KEEP),UNIT=2314,
//           VOLUME=SER=AFL,LABEL=RETPD=31
//FOUR  DD   DSNAME=END,DISP=OLD,UNIT=2400-2,
//           VOLUME=SER=TAPE75,SEP=(ONE,TWO)
```

Using the preceding example, if you want to request that the data set defined by the THREE DD statement be on a channel other than the one assigned to the data set defined by the TWO DD statement code:

```
//PROG1 EXEC PGM=PROCESS
//ONE   DD   UNIT=2400,VOLUME=PRIVATE,LABEL=(,NL)
//TWO   DD   DSNAME=SYSTEM.USER.ABC,DISP=OLD
//THREE DD   DSNAME=HELP,DISP=(,KEEP),UNIT=2314,
//            VOLUME=SER=AFL,LABEL=RETPD=31,SEP=TWO
//FOUR  DD   DSNAME=END,DISP=OLD,UNIT=2400-2,
//            VOLUME=SER=TAPE75,SEP=(ONE,TWO)
```

As a result of this example, the data sets defined by ONE and TWO may or may not be on the same channel. The data sets defined by THREE and FOUR may or may not be on the same channel, but neither one will be on the same channel as the data set defined by TWO. Also, the data set defined by FOUR will not be on the same channel as the data set defined by ONE.

Remember that the request for channel separation made with the SEP parameter is only a suggestion to the system for more efficient operation, and not a requirement. If your data set must be on separate channels specify unit addresses with the UNIT parameter. You can obtain a configuration chart with all the unit addresses from your manager or supervisor.

AFF: The AFF parameter provides a shortcut method of requesting channel separation. When two or more data sets in the same job step have the same channel separation requirements, you can code the SEP parameter for the first data set, and then request the same separation using the AFF parameter in the later data sets. The AFF parameter tells the system that you want the data set defined on this DD statement to have the same channel separation as the data set defined in the named DD statement.

The format of the AFF parameter is:

```
AFF=ddname
```

Replace "ddname" with the name of the earlier DD statement that requests the desired channel separation. This data set and the earlier data set may or may not be on the same channel.

In the following example, the data sets defined by DD1 and DD2 may or may not be on the same channel. The data sets defined by DD3 and DD5 may or may not be on the same channel, but neither one will be on the same channel as the data set defined by DD1 and DD2. The data set defined by DD4 will not be on the same channel as the data set defined by DD3, but it may be on the same channel as the data sets defined by DD1, DD2, or DD5.

```
//STEP EXEC PGM=METHOD2
//DD1  DD   ...
//DD2  DD   ...
//DD3  DD   SEP=(DD1,DD2),...
//DD4  DD   SEP=DD3,...
//DD5  DD   AFF=DD3,...
```

The requests for channel separation and affinity are only suggestions to the system. They will be honored if the appropriate devices are available.

DUMMY: The DUMMY parameter allows you to bypass I/O operations and data set disposition. When your processing programs ask to read the dummy data set, the read request is recognized, but no data is transmitted. This facility is particularly useful when you are debugging your program because it saves you processing time. For example, you can suppress the reading of a data set until you are sure your program is going to process it correctly.

The DUMMY parameter is a positional parameter. That means that it must be coded first in the operand field. For example:

```
//IN DD DUMMY,DSNAME=ABLE,DISP=OLD,LABEL=(,NL),DCB=BLKSIZE=100
//DDT DD DUMMY,DSNAME=BAKER,DISP=OLD
```

If you wish, you can omit all other parameters required to define your data set except DCB. (If you were not going to write the DCB parameter you do not have to write it.) For example, the two DD statements shown above could be coded as follows:

```
//IN  DD DUMMY,DCB=BLKSIZE=100
//DDT DD DUMMY
```

When you are ready to perform I/O operations on your data set, simply omit the DUMMY parameter and code all parameters required to retrieve the data set. For example:

```
//IN DD DSNAME=ABLE,DISP=OLD,LABEL=(,NL),DCB=BLKSIZE=100
//DDT DD DSNAME=BAKER,DISP=OLD
```


Kept Data Set

A kept data set is a nontemporary data set defined with the DISP=(,KEEP) parameter. The parameters shown in Table 15 are used to retrieve a kept data set.

This section is summarized in Table 71 of Appendix D.

Data Set Information

The DSNAME and DISP parameters are required.

DSNAME: The format of the DSNAME parameter is:

```
          {dsname              }
DSNAME=   {dsname(membername)  }
```

Replace the term "dsname" with the name of the kept data set. For example, if you created the data set as DSNAME=NONO you must again code

```
DSNAME=NONO
```

If you are retrieving a member of a partitioned data set, you must code the member name in parentheses after the data set name. For example,

```
DSNAME=COLOR(BLUE)
```

DISP: The format of the DISP parameter is:

```
DISP=( {OLD} [,KEEP  ] [,KEEP  ] )
       {SHR} [,CATLG ] [,CATLG ]
             [,DELETE] [,DELETE]
             [,      ]
```

You must code either OLD or SHR. OLD indicates that you are retrieving the data set. SHR indicates that you are operating under MFT or MVT and that concurrent jobs can also retrieve this data set at the same time.

The second positional subparameter tells the system what is to be done with the data set at the end of the job step. If you omit this subparameter the data set remains kept. If you code CATLG the data set will be cataloged. If you code DELETE the data set is deleted.

The third positional subparameter tells the system what is to be done with the data set if the step ABENDs. If you omit this subparameter the data set remains kept. You can specify instead CATLG or DELETE.

Note: When you are retrieving a member of a partitioned data set, the disposition (KEEP, CATLG, or DELETE) you specify applies to the entire data set. If you want to delete a particular member, you must use the IEHPROGM utility program.

In the following example you are retrieving kept data set named FRANK. FRANK resides on a 2314 volume named ABCXYZ.

```
//#75 DD DSNAME=FRANK,DISP=SHR,UNIT=2314,VOLUME=SER=ABCXYZ
```

In the following example, you are retrieving a member of a kept data set named FILE. FILE resides on a 2311 volume named PACK70. You want to catalog the entire partitioned data set after you use the member.

```
//AB DD DSNAME=FILE(MEM12),DISP=(OLD,CATLG),UNIT=2311,
//         VOLUME=SER=PACK70
```

Location of the Data Set

The UNIT and VOLUME parameters are required. The LABEL parameter is required only if the data set does not have standard labels or if it is not the first data set on a tape volume.

UNIT: You can use the UNIT parameter to directly specify units for your data set or to request the same units used for an earlier data set in the job step (unit affinity).

Requesting Units: The format of the UNIT parameter for directly specifying units for your data set is:

```
UNIT=({unit address}[,unitcount][,DEFER])
      {device type }[,P        ]
      {group name  }[,         ]
```

The first positional subparameter identifies the tape drive or direct access you want to use for your data set by its address, or unit name, or group name.

unit address
is the actual machine address of the direct access device. For example,

```
UNIT=190
```

You should not specify the address unless you are sure you want it. Do not specify an address if you are going to need more than one unit for the data set. (Multiple units are requested with the second parameter of the UNIT parameter.)

device type
corresponds to the type of device. Coding a device type provides you with a certain degree of device independence in that your data set may be placed in any number of devices of the same type. For example, if you code

```
UNIT=2311
```

your data set uses any 2311 Disk Storage Drive. The following device types can be specified.

Device Type	Description
2301	2301 Drum Storage Unit
2302	2302 Disk Storage Drive
2303	Any 2303 Drum Storage Unit
2311	2311 Disk Storage Drive
2314	2314 Storage Facility
2321	Any bin mounted on a 2321 data cell drive
2400	2400 series Nine-Track Magnetic Tape Drive that can be allocated to a data set written or to be written in 800 bpi density when the dual density feature is not installed on the drive, or in 1600 bpi when the dual-density feature is installed in the drive.
2400-1	2400 series Magnetic Tape Drive with Seven-Track Compatibility and without Data Conversion
2400-2	2400 series Magnetic Tape Drive with Seven-Track Compatibility and Data Conversion

2400-3 2400 series Nine-Track Magnetic Tape Drive that can be allocated to a data set written or to be written in 1600 bpi density.

2400-4 2400 series Nine-Track Magnetic Tape Drive having an 800 and 1600 bpi capability.

group name
is the name of a collection of devices, selected by your installation during system generation. For example, your installation might select the name TAPE for all tape drives in the configuration. If you do not care which tape drive is used for your data set, you would then code

```
UNIT=TAPE
```

Your manager or supervisor should tell you which group names were generated for your installation.

The second positional subparameter is used only if your data set has more than one volume and if you want more than one of these volumes to be mounted at the same time. This subparameter indicates how many drives are to be used for mounting your data set's volumes.

unit count
indicates how many units you want assigned to the data set. You can specify a maximum of 59 units per DD statement. Make sure that your system has at least the number of units you specify. Otherwise an error will result. If you specify fewer units than the number of volumes in your data set, only the same number of volumes as there are units can be mounted at the same time. (If you request fewer units than volumes code the PRIVATE subparameter in the VOLUME parameter.) If you specify the same number of units as there are volumes, all volumes can be mounted at the same time.

P
specifies parallel mounting. When you request parallel mounting the system counts the number of serial numbers specified with the VOLUME parameter and assigns to the data set as many units as there are serial numbers.

DEFER is the third positional subparameter. DEFER requests the system to assign the required tape drives or direct access devices to the data set and to defer the mounting of the volume or volumes until the processing program attempts to open the data set. If you are running under MFT or MVT, operating efficiency may decrease if you specify DEFER for a tape data set. This is because if you code DEFER, the system issues a mount message to the operator when the processing program attempts to use the data set and then waits until the volume is mounted. If you do not code DEFER, the mount message is issued when the device is assigned and there is no waiting for the operator. By the time your program tries to use the data set, it is very likely that the operator has already mounted the volume and that the system will not have to wait for him.

Unit Affinity: To conserve the number of units in a job step, you can request that the kept data set be assigned to the same device or devices as assigned to an earlier data set in the same step. This technique, known as unit affinity, indicates that you want the volumes on which the kept data set resides and the volumes used by the earlier data set to be mounted on the same devices in sequential order. Unit affinity implies deferred mounting for one of the volumes since both volumes cannot be mounted at the same time. Before you request unit affinity, make sure that both sets use the same type of removable volumes; for example, both

data sets could be on 2314 volumes, or both on 2400 volumes. It does not make sense to request unit affinity when both data sets are on different types of volume.

The format of the UNIT parameter for requesting unit affinity is:

```
UNIT=AFF=ddname
```

Replace the term "ddname" with the name of an earlier DD statement in the job step. Whenever it is required, the volume of the earlier data set will be demounted and the cataloged data set will be mounted on the same unit. If the ddname refers to a DD statement that defines a dummy data set, the data set defined on the DD statement requesting unit affinity is assigned a dummy status.

When unit affinity is requested for two data sets that reside on different 2321 volumes, the data sets are assigned the same device but can be assigned different bins.

In the following example, the DD statement named DD2 requests that the system assign the same number of units to this data set as it assigns to the data set defined on the statement named DD1. Since DD1 requests two devices, these two device are assigned to the data set defined by DD2.

```
//STEPC EXEC PGM=OVER
//DD1   DD   UNIT=(2400,2),VOLUME=(,,,2)
//DD2   DD   DSNAME=ALPHA.DISP=OLD,
//           VOLUME=SER=(TAPE22,TAPE23),UNIT=AFF=DD1
```

VOLUME: The format of the VOLUME parameter for kept data sets is:

```
VOLUME=([PRIVATE][,RETAIN,]SER=(serial,...)
                [,      ]
```

The PRIVATE subparameter means that you want the volume(s) on which the kept data set resides to be private. (A private volume is one that cannot be allocated to a new temporary data set that makes a nonspecific request for a public volume.) The RETAIN subparameter means that the volume will not be demounted at the end of the step. PRIVATE must be coded when you code RETAIN. (RETAIN has the same effect on volume mounting as coding PASS in the DISP parameter.) Private volumes are demounted at the end of the step unless it is an unremovable volume, such as a 2301, or unless it has been designated by your installation to be a permanently resident volume, for example, the system residence volume. It is recommended that you use the RETAIN subparameter whenever you want the volume to remain mounted at the end of the step. If there are more volumes than devices, only the last volumes mounted simultaneously are retained.

The SER subparameter identifies the serial numbers of the volumes on which your data set resides. Replace the term "serial" with the 1-to-6 character serial number associated with the volume. The volume serial number can contain any alphameric and national characters and the hyphen. If it is necessary to include special characters, other than a hyphen, in the volume serial number, enclose it in apostrophes, for example,

```
VOLUME=SER=(54AB2,'6/10/8').
```

If only one volume is involved you need not code the parentheses, for example,

```
VOL=SER=ABCDEF.
```

In the following example you retrieve a kept data set named A99 that resides on volumes AA22 and AB45. Only one tape drive named TAPE will be used and the last volume used is to remain mounted at the end of the step.

```
//DD9 DD DSNAME=A99,DISP=OLD,UNIT=TAPE,
//       VOLUME=(PRIVATE,RETAIN,SER=(AA22,AB45))
```

In the following example, you retrieve the same data set, A99, but request that both volumes be mounted in parallel.

```
//DD99 DD DSNAME=A99,DISP=OLD,UNIT=(TAPE,P),
//        VOLUME=(PRIVATE,RETAIN,SER=(AA22,AB45))
```

In the case of multivolume data sets, give only the serial numbers of the volumes you want to process in the order you want to process them. For example, you want to retrieve a kept data set named PLUS. PLUS resides on volumes A24, A25, A26, and A27, but you only want to process volumes A25, and A27, in that order. Only one tape drive named TAPE2 will be used.

```
//DDT  DD DSNAME=PLUS,DISP=OLD,
//        UNIT=TAPE2,VOLUME=SER=(A25,A27)
```

LABEL: The LABEL parameter is used only if the kept data set does not have standard labels, or if the data set is not the first data set on a tape volume. The format of the LABEL parameter for kept data sets is:

```
LABEL=([sequence#][,SUL]
                  [,NL ]
                  [,NSL]
                  [,BLP])
```

The first positional subparameter indicates the order of your data set on the tape volume. You only have to specify a sequence number if your data set is not the first data set on the volume. If you do not specify a sequence number and there are other data sets on the volume, your data set cannot be retrieved and the step will ABEND. Replace the term "sequence#" with the sequence number of your data set. For example, if your data set is the third data set on the volume, code

```
LABEL=3
```

Note: If you request the system to bypass label processing (specify BLP in the second positional subparameter of the LABEL parameter), the system treats anything within tape marks as a data set. Therefore, if you want your data set to be written in the proper sequence, you must include all header and trailer labels and data sets that precede your data set in the sequence number subparameter.

The second positional subparameter indicates the label type of your data set. Code:

SUL
if the data set has standard and user labels.

NL
if the data set has no labels.

NSL
if the data set has nonstandard labels.

BLP
to bypass label processing.

In the following example a kept data set named BLUE is retrieved. BLUE is the fourth data set on a tape volume named TAPE2. The data set has standard labels.

```
//T77 DD DSNAME=BLUE,DISP=OLD,UNIT=2400,VOLUME=SER=TAPE2,LABEL=4
```

Using the same example, if the data set has no labels, code:

```
//T77 DD DSNAME=BLUE,DISP=OLD,UNIT=2400,VOLUME=SER=TAPE2,
//    LABEL=(4,NL)
```


Data Attributes

The DCB parameter is needed only if the kept data set is on magnetic tape and has no labels or nonstandard labels. Follow the instructions given for "Magnetic Tape" in the section "Creating a New Data Set". If the data set has standard labels, you can specify the subparameters shown for "Cataloged Data Sets" in Tables 18, 19, 20, and 21.

Special Processing Options

There are two special processing options for kept data sets:

1. You can request channel separation from other data sets on the same job step using either the SEP or AFF parameter.
2. You can suppress I/O operations on a sequential data set (magnetic tape or direct access) using the DUMMY parameter.

SEP: When two or more data sets are to be used in a job step, processing time may be shortened by requesting that the system transmit data sets over separate channels. For example, it would be faster to have your input data set and your output data set on separate channels than to have them on the same channel.

You can request channel separation for data sets in each job step using the SEP parameter. If possible, the system will honor this request. It may not always be possible to honor the request for separation because given devices may not be available for allocation when the job step is executed or because there may not be enough devices of a given type on different channels for all the data sets that request separation. If channel separation is not requested, the system will assign any available channel that has the device specified with the UNIT parameter.

The format of the SEP parameter is:

```
SEP=(ddname,...)
```

Replace the terms "ddname" with the names of up to eight earlier DD statements in the same job step. The earlier DD statements can define any type of data set: new or existing, on magnetic tape or direct access, temporary or nontemporary, etc.

In the following example, you request that the data set defined by the FOUR DD statement be assigned to a channel other than the ones assigned to the data sets defined by the ONE and TWO DD statement. The data sets defined by ONE and TWO may or may not be on the same channel. The data set defined by THREE may or may not be on the same channel as any of the other three data sets.

```
//PROG1 EXEC PGM=PROCESS
//ONE   DD   UNIT=2400,VOLUME=PRIVATE,LABEL=(,NL)
//TWO   DD   DSNAME=SYSTEM.USER.ABC,DISP=OLD
//THREE DD   DSNAME=HELP,DISP=(,KEEP),UNIT=2314,
//            VOLUME=SER=AFL,LABEL=RETPD=31
//FOUR  DD   DSNAME=END,DISP=OLD,UNIT=2400-2,
//            VOLUME=SER=TAPE75,SEP=(ONE,TWO)
```

Using the preceding example, if you want to request that the data set defined by the THREE DD statement may be on a channel other than the one assigned to the data set defined by the TWO DD statement code:

```
//PROG1 EXEC PGM=PROCESS
//ONE   DD   UNIT=2400,VOLUME=PRIVATE,LABEL=(,NL)
//TWO   DD   DSNAME=SYSTEM.USER.ABC,DISP=OLD
//THREE DD   DSNAME=HELP,DISP=(,KEEP),UNIT=2314,
//            VOLUME=SER=AFL,LABEL=RETPD=31,SEP=TWO
//FOUR  DD   DSNAME=END,DISP=OLD,UNIT=2400-2,
//            VOLUME=SER=TAPE75,SEP=(ONE,TWO)
```

As a result of this example, the data sets defined by ONE and TWO may or may not be on the same channel. The data set defined by THREE and FOUR may or may not be on the same channel, but neither one will be on the same channel as the data set defined by TWO. Also, the data set defined by FOUR will not be on the same channel as the data set defined by ONE.

Remember that the request for channel separation made with the SEP parameter is only a suggestion to the system for more efficient operation, and not a requirement. If your data set must be on separate channels specify unit addresses with the UNIT parameter. You can obtain a configuration chart with all the unit addresses from your manager or supervisor.

AFF: The AFF parameter provides a shortcut method of requesting channel separation. When two or more data sets in the same job step have the same channel separation requirements, you can code the SEP parameter for the first data set, and then request the same separation using the AFF parameter in the later data sets. The AFF parameter tells the system that you want the data set defined on this DD statement to have the same channel separation as the data set defined in the named DD statement.

The format of the AFF parameter is:

```
AFF=ddname
```

Replace "ddname" with the name of the earlier DD statement that requests the desired channel separation. This data set and the earlier data set may or may not be on the same channel.

In the following example, the data sets defined by DD1 and DD2 may or may not be on the same channel. The data sets defined by DD3 and DD5 may or may not be on the same channel, but neither one will be on the same channel as the data sets defined by DD1 or DD2. The data set defined by DD4 will not be on the same channel as the data set defined by DD3, but it may be on the same channel as the data sets defined by DD1, DD2, or DD5.

```
//STEP EXEC PGM=METHOD2
//DD1  DD   ...
//DD2  DD   ...
//DD3  DD   SEP=(DD1,DD2),...
//DD4  DD   SEP=DD3,...
//DD5  DD   AFF=DD3,...
```

The requests for channel separation and affinity are only suggestions to the system. They will be honored if the appropriate devices are available.

DUMMY: The DUMMY parameter allows you to bypass I/O operations and data set disposition. When your processing program asks to read the dummy data set, the read request is recognized, but no data is transmitted. This facility is particularly useful when you are debugging your program because it saves you processing time. For example, you can suppress the reading of a data set until you are sure your program is going to process it correctly.

The DUMMY parameter is a positional parameter. That means that it must be coded first in the operand field. For example:

```
//INTAPE DD DUMMY,DSNAME=BC74,DISP=OLD,UNIT=2400,
//          VOLUME=SER=7942,LABEL=(,NSL),DCB=DEN=2
//P79 DD DUMMY,DSNAME=TOM,DISP=OLD,UNIT=2311,VOLUME=SER=A27941
```

If you wish, you can omit all other parameters required to define your data set except DCB. (If you were not going to write the DCB parameter you do not have to write it.) For example, the two DD statements shown above could be coded as follows:

```
//INTAPE DD DUMMY,DCB=DEN=2
//P79 DD DUMMY
```

When you are ready to perform I/O operations on your data set, simply omit the DUMMY parameter and code all parameters required to retrieve the data set. For example:

```
//INTAPE DD DSNAME=BC74,DISP=OLD,UNIT=2400,VOLUME=SER=7942,
//          LABEL=(,NSL),DCB=DEN=2
//P79 DD DSNAME=TOM,DISP=OLD,UNIT=2311,VOLUME=SER=A27941
```

Table 22. Parameters for Extending a Data Set

Data Set	Parameter Type	Parameter	Comments
Passed Data Set	Data set information	DSNAME	Required.
		DISP	Required.
	Location of the data set	UNIT	Required only if you want more units.
		VOLUME	Required only if you need more volumes.
		LABEL	Required only if the data set does not have standard labels.
	Size of the data set	SPACE	Required only if you want to override the secondary quantity.
	Data Attributes	DCB	May be required if data set does not have standard labels.
	Special Processing Option	DUMMY	Optional.
Cataloged Data Set	Data set information	DSNAME	Required.
		DISP	Required.
	Location of the data set	UNIT	Optional.
		VOLUME	Required only if you need more volumes.
		LABEL	Required only if the data set does not have standard labels.
	Size of the data set	SPACE	Required only if you want to override the secondary quantity.
	Data Attributes	DCB	Required only if the data set does not have standard labels.
	Special Processing Options	SEP	Either parameter can be used.
		AFF	
		DUMMY	Optional.
Kept Data Set	Data set information	DSNAME	Required.
		DISP	Required.
	Location of the data set	UNIT	Required.
		VOLUME	Required.
		LABEL	Required only if data set does not have standard labels.
	Size of the data set	SPACE	Required only if you want to override the secondary quantity.
	Data Attributes	DCB	Required only if the data set does not have standard labels.
	Special Processing Options	SEP	Either parameter can be used.
		AFF	
		DUMMY	Optional.

Extending an Existing Data Set

You can only extend with additional output data sets on magnetic tape or sequential data sets on direct access devices. You cannot extend a direct data set or a member of a partitioned data set using JCL facilities. You can, however, add more members to a partitioned data set using JCL facilities. (The publication IBM System/360 Operating System: Supervisor and Data Management Services explains how to extend a direct data set or a member of a partitioned data set.)

Data sets on magnetic tape or direct access devices are extended in similar ways depending on whether they were passed, cataloged, or kept when you created them. This chapter describes how to extend or add more members to:

- Passed data sets.
- Cataloged data sets.
- Kept data sets.

Table 22 shows the parameters required to extend a data set. Please fold out Table 22 while reading this chapter.

Notes:

- When you extend a data set, the read/write mechanism is automatically positioned after the last record in the data set.
- If you extend a data set that has fixed block standard (FBS) records and the last block was a truncated one, an end-of-data set condition occurs when the truncated block is encountered. If an attempt is made to read the data set backward on magnetic tape, processing is terminated immediately (with an end-of-data set condition) upon reading the truncated block.
- When you are extending or adding members to a data set on a direct access device, make sure that there is enough room left in the primary quantity for the additional output or that you requested a secondary quantity when you defined the data set.

Passed Data Set

A passed data set is one you defined with the DISP=(,PASS) parameter. The most common type of passed data set is a "temporary for the duration of the job" data set, although you can also have nontemporary passed data sets.

A passed data set can only be extended in the same job it was created. However, if you have a nontemporary passed data set, you can change its disposition to KEEP or CATLG when you extend it so that it can be retrieved or extended in other jobs.

The parameters shown in Table 22 are used to extend passed data sets. Remember that you can extend any sequential passed data set, but you can only add members to a partitioned data set.

This section is summarized in Table 72 of Appendix D.

Data Set Information

The DSNAME and DISP parameters are required.

DSNAME: You can identify the passed data set in one of two ways:

1. Giving its name exactly as it appeared in the DD statement where you defined it, or

2. Making a backward reference to the DD statement that defined the data set.

Both ways are equally valid, but the second is more useful because if you decide to change the data set's name you only have to change it in the original DD statement. You must identify the partitioned data set by name when you are adding a member.

1. The format of the DSNAME parameter for identifying the passed data set by name is:

```
DSNAME={dsname                    }
       {dsname(membername)        }
       {&&name                    }
       {&&name(membername)        }
       {&name                     }
       {&name(membername)         }
```

Replace "dsname" with the name of the passed data set you are extending. If you are adding a new member replace "membername" with the name of the new member. The membername is enclosed in parentheses and consists of one to eight alphameric or national (@,$,#) characters. The first character must be a letter or national character. For example, if you are adding a member named #2 to a partitioned data set named GEORGE, code:

```
DSNAME=GEORGE(#2)
```

2. The format of the DSNAME parameter for identifying the passed data set with a backward reference is:

```
DSNAME={*.ddname                  }
       {*.stepname.ddname         }
       {*.stepname.procstepname.ddname}
```

Replace "ddname" with the name of the DD statement where the passed data set is defined. Replace "stepname" with the name of the EXEC statement of the step that has the earlier DD statement. If the earlier DD statement is contained in the same step, omit the stepname, i.e.,

```
DSNAME=*.ddname
```

When the earlier DD statement is contained in a cataloged procedure step, you must give both the name of the job step that invokes the procedure and the procedure stepname, i.e.,

```
DSNAME=*.stepname.procstepname.ddname
```

The following example shows how to code the DSNAME parameter to identify a passed data set defined in a previous step.

```
//STEP2 EXEC ...
//TEMP  DD   DISP=(,PASS),...
.
.
.
//STEP4 EXEC ...
//USE   DD   DSNAME=*.STEP2.TEMP,...
```

The following example shows how to code the DSNAME parameter to identify a passed data set defined in a cataloged procedure step. The first part of the example shows your input deck; the second part shows a cataloged procedure called by the STEP3 EXEC statement in your deck.

```
INPUT DECK
.
.
.
//STEP3 EXEC PROC=PROC7
.
.
.       stepname
//STEP5 EXEC
//DDA DD DSNAME=*.STEP3.NOVA.FILE
                        ddname
CATALOGED PROCEDURE
.
. procstepname
.
//NOVA EXEC PGM=N
//FILE DD DSNAME=LIST,DISP=(,PASS),...
```

DISP: The DISP parameter indicates that you are extending the passed data set. You can code the DISP parameter in one of two ways depending on whether other steps in the job will also use the data set or whether this is the last time you are using the data set.

1. The format of the DISP parameter for passing the data set to a subsequent job step is:

```
DISP=(MOD,PASS[,PASS   ])
              [,DELETE ]
              [,CATLG  ]
              [,KEEP   ]
```

You must code MOD. The PASS subparameter is required and it indicates that the data set can be used by later steps in the job. The third subparameter tells the system what is to be done with the data set if the step abnormally terminates. You can specify PASS or DELETE for both temporary and nontemporary data sets. PASS is

assumed if you omit the third subparameter. If the data set is nontemporary you can specify KEEP or CATLG.

Note: When you are adding a member to a partitioned data set, the disposition (PASS, DELETE, CATLG, or KEEP) you specify applies to the entire data set and not to the one member. If you want to delete a particular number of a nontemporary partitioned data set you must use the IEHPROGM utility program. The entire temporary partitioned data set is deleted at the end of the job.

2. The format of the DISP parameter for the last time you use a passed temporary data set in a job is:

```
DISP=(MOD,DELETE,DELETE)
```

You must code MOD. A passed temporary data set will always be deleted at the end of the job so you do not have to code the DELETE subparameters.

The format of the DISP parameter for the last time you use a passed non-temporary data set in a job is:

```
                    [,KEEP  ]  [,KEEP  ]
DISP=(MOD[,DELETE] [,DELETE])
                    [,CATLG ]  [,CATLG ]
                    [,      ]
```

You must code MOD. A passed nontemporary data set will be kept at the end of the job unless you specify DELETE or CATLG. If the step abnormally terminates, the data set will also be kept unless you specify DELETE or CATLG.

In the following example, two data sets named &&DATA (with a member named MEM1) and LIST are defined in the first step of a job. In the second step, a new member named MEM2 is added to &&DATA and passed again. In the third step, LIST is extended with additional output and passed again. In the fourth step, &&DATA is retrieved. The entire &&DATA data set is deleted at the end of the job. LIST is extended again and cataloged.

```
//MYJOB JOB
//STEP1 EXEC PGM=UNO
//DS1   DD   DSNAME=&&DATA(MEM1),DISP=(,PASS),UNIT=2311,
//           VOLUME=PRIVATE,SPACE=(TRK,(25,,3))
//DS2   DD   DSNAME=LIST,DISP=(,PASS),UNIT=2400
.
.
.
//STEP2 EXEC PGM=DOS
//DD3   DD   DSNAME=&&DATA(MEM2),DISP=(MOD,PASS)
.
.
.
//STEP3 EXEC PGM=TRES
//DS4   DD   DSNAME=LIST,DISP=(MOD,PASS)
.
.
.
//STEP4 EXEC PGM=CUATRO
//DS5   DD   DSNAME=*.STEP1.DS1,DISP=OLD
//DS6   DD   DSNAME=*.STEP1.DS2,DISP=(MOD,CATLG)
```


Location of the Data Set

The system obtains unit and volume information from the original DD statement. The UNIT, VOLUME and LABEL parameters are only needed in the following cases:

1. Use the UNIT parameter if you want more devices allocated to the data set than were requested the last time the data set was passed.

2. Use the LABEL parameter if the data set does not have standard labels.

3. If you are extending a data set and the number of volumes required to extend the data set may exceed the number of volumes you requested, when you defined the data set, either specify a new volume count with the VOLUME parameter or deferred mounting with the UNIT parameter. If you specify deferred mounting the system will give you a new volume each time you need it.

UNIT: The UNIT parameter should be used only if the passed data set is a multivolume data set. The system assigns the same number of devices to the data set as it did in a previous step. If you want more devices assigned to the data set, code in the unit count subparameter the total number of devices needed. If you need an indefinite number of new volumes to extend the data set, code the DEFER subparameter.

The format of the UNIT parameter for passed data sets is:

```
UNIT=([,unit count][,DEFER])
      [,
```

Replace "unit count" with the number of devices needed for the data set. Remember that you should not request more devices than there are volumes. Code DEFER if you need an indefinite number of new volumes to extend the data set.

In the following example, a data named &&JUNK is defined in the first step. A maximum of five volumes are required for &&JUNK and they are to be mounted in succession on two units. In the second step, you extend &&JUNK, but request that four units be used. In the third step, you again extend &&JUNK with additional output and request that more volumes be assigned to it if necessary. Four units will be used. The data set can be used by subsequent job steps.

```
//SAM   JOB
//STEPA EXEC PGM=X
//DATA  DD   DSNAME=&&JUNK,DISP=(,PASS),UNIT=(2314,2)
//           VOLUME=(PRIVATE,,,5),SPACE=(CYL,(20,15))
//STEPB EXEC PGM=Y
//GET   DD   DSNAME=*.STEPA.DATA,DISP=(MOD,PASS),UNIT=(,4)
.
.
//STEPC EXEC PGM=2
//EXT   DD   DSNAME=*.STEPA.DATA,DISP=(MOD,PASS),UNIT=(,,DEFER)
.
.
.
```

VOLUME: The VOLUME parameter is used only when you want to request additional volumes for the data set. You must indicate the new maximum number of volumes that can be used. You cannot make specific requests for more volumes. (You do not have to request more units just because you request more volumes.) The format of the VOLUME parameter for passed data sets is:

```
VOLUME=(,,,volcount)
```

Replace "volcount" with the maximum number of volumes that can be used by the data set.

In the following example a data set named &&PART is defined in the first step. &&PART will reside on volume BC44 and can use two more volumes. In the second step, you extend &&PART with additional output and specify that a total of five volumes and three units can be used.

```
//JOE JOB
//ONE EXEC PGM=W
//DD1 DD   DSNAME=&PART,DISP=(,PASS),UNIT=2400,
//         VOLUME=(,,,3,SER=BC44)
.
.
.
//TWO EXEC PGM=Y
//DD5 DD   DSNAME=*.ONE.DD1,DISP=(MOD,PASS),UNIT=(,3),
//         VOLUME=(,,,5)
.
.
.
```

LABEL: The LABEL parameter is needed only if the passed data set does not have standard labels. You must code the same label type you specified when you defined the data set. No other subparameters are allowed. The format of the LABEL parameter for passed data sets is:

```
LABEL=( ,SUL  )
      ( ,NL   )
      ( ,NSL  )
      ( ,BLP  )
```

Code:

SUL
if the data set has standard and user labels.

NL
if the data set has no labels.

NSL
if the data set has nonstandard labels.

BLP
to bypass label processing.

In the following example, a data set named BANK is defined in the first step. BANK has nonstandard labels. BANK is retrieved and passed in the second step.

```
//MAC   JOB
//STEPA EXEC PGM=ZYZ
//DDAA  DD   DSNAME=BANK,DISP=(,PASS),UNIT=2400,LABEL=(,NSL)
.
.
.
//STEPB EXEC PGM=WZW
//DDAB  DD   DSNAME=*.STEPA.DDAA,DISP=(MOD,PASS),LABEL=(,NSL)
```

Size of the Data Set

You can use the SPACE parameter to override any secondary quantity specified when you created the data. This new secondary quantity is in effect only for the duration of the job step. Note that if you did not specify a secondary quantity when you created the data set you cannot do so now.

SPACE: The format of the SPACE parameter is:

```
SPACE=( TRK          (1,secondary quantity))
      ( CYL          )
      ( block length )
```

The first positional parameter indicates whether you are requesting space in units of tracks (TRK), cyclinders (CYL), or blocks. If you are using blocks, replace "block length" with the average block length of your data. You do not have to use the same type of unit of space you used when you created the data set.

Any primary quantity you code is ignored. However, for reasons of syntax you must code some number, such as 1, as the primary quantity.

Replace the term "secondary quantity" with the amount of space you desire in the units specified.

In the following example, you are extending a passed data set named &&DRIP that you defined in the first step of your job. &&DRIP requested a secondary allocation of two cylinders. When you extend &&DRIP in the second step you want to use a secondary allocation of 300 tracks. This allocation is in effect only during the second step.

```
//HISJOB JOB
//STEP1  EXEC PGM=EAST
//DSET1  DD   DSNAME=&&DRIP,DISP=(,PASS),UNIT=2314,
//            VOLUME=(PRIVATE,,,2,SER=794321),SPACE=(CYL,(25,1))
.
.
.
//STEP2  EXEC PGM=WEST
//DSET11 DD   DSNAME=*.STEP1.DSET1,DISP=(MOD,PASS),
//            SPACE=(TRK,(1,300))
.
.
.
```

Data Attributes

The DCB parameter is used only if the data set does not have standard labels and the DD statement that defined the passed data set contains the DCB parameter.

DCB: You must specify the DCB parameter in one of two ways:

1. Copy the DCB parameter exactly as it appeared in the DD statement where you defined the passed data set, or
2. Make a backward reference to the DD statement that defined the data set.

Making a backward reference reduces the possibility of error in copying the parameter.

The format of the DCB parameter for copying the attributes of the passed data set is:

```
DCB=(list of attributes)
```

Code exactly the same DCB subparameters you coded in the DD statement where the passed data set is defined. You may not add, change, or delete subparameters. For example, if you coded

```
DCB=(BLKSIZE=800,LRECL=80)
```

you must again code

```
DCB=(BLKSIZE=800,LRECL=80)
```

The format of the DCB parameter for making a backward reference is:

```
      ⎧*.ddname                          ⎫
DCB=  ⎨*.stepname.ddname                 ⎬
      ⎩*.stepname.procstepname.ddname    ⎭
```

Replace "ddname" with the name of the DD statement whose DCB parameter you want to copy. Replace "stepname" with the name of the EXEC statement of the step that has the earlier DD statement. If the earlier DD statement is contained in the same job step, omit the stepname, i.e.,

```
DCB=*.ddname
```

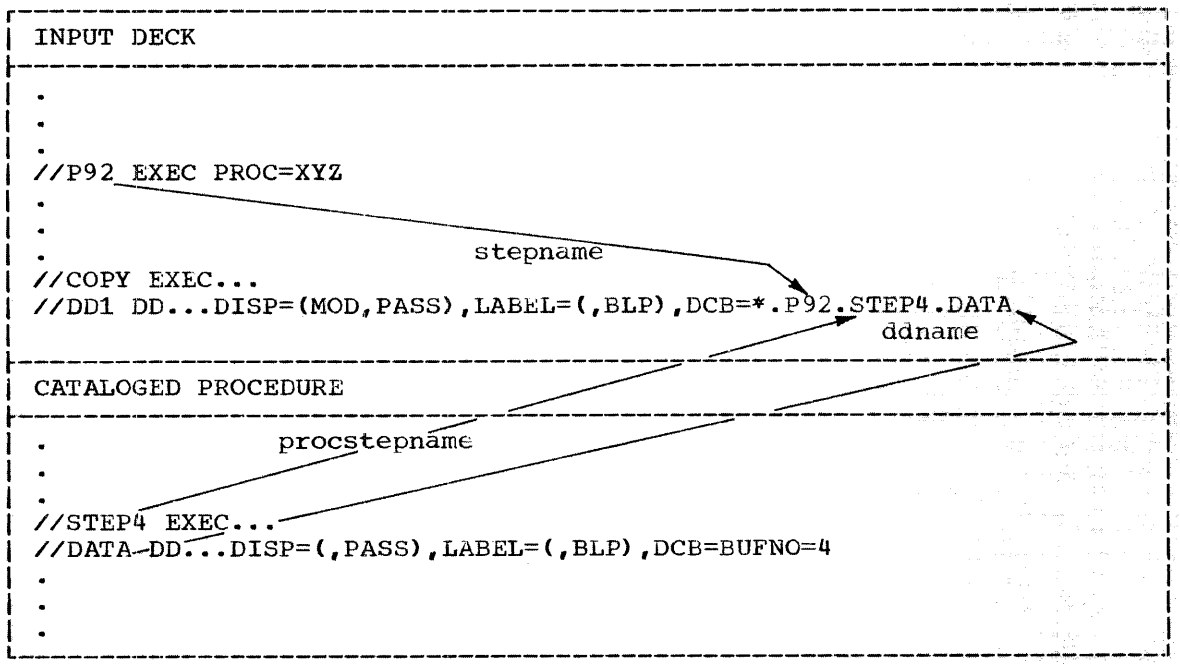
When the earlier DD statement is contained in a cataloged procedure step, you must give both the name of the job step that invokes the procedure and the procedure step name, i.e.,

```
DCB=*.stepname.procstepname.ddname
```

The following example shows how to code the DCB parameter to copy the DCB parameter of a DD statement in a previous step:

```
//STEP2 EXEC...
//DD1   DD...DISP=(,PASS),LABEL=(,NL),DCB=(BLKSIZE=1600,LRECL=80)
.
.
.
//STEP4 EXEC...
//COPY  DD...DISP=(MOD,PASS),LABEL=(,NL),DCB=*.STEP2.DD1
.
.
.
```

The following example shows how to code the DCB parameter to copy the attributes of a passed data set defined in a cataloged procedure step. The first part of the example shows your input deck, the second part shows your cataloged procedure called by the P92 EXEC statement in your deck.



Special Processing Option

You can use the DUMMY parameter to bypass operations and data set disposition on a sequential data set (magnetic tape or direct access).

DUMMY: The DUMMY parameter allows you to bypass I/O operations and data set disposition. When your processing program asks to extend the dummy data set, the write request is recognized, but no data is transmitted. This facility is particulary useful when you are debugging your program because it saves you processing time. For example, you can suppress the extending of a data set until you are sure your program is going to produce meaningful output. The DUMMY parameter is a positional parameter. That means that it must be coded first in the operand field. For example:

```
//INPUT DD DUMMY,DSNAME=*.STEP1.DDA,DISP=(MOD,PASS),LABEL=(,NL),
// DCB=BLKSIZE=500
//READ DD DUMMY,DSNAME=*.STEP4.ABC,DISP=(MOD,PASS)
```

If you wish, you can omit all other parameters required to define your data set except DCB. (If you were not going to write the DCB parameter you do not have to write it.) For example, the two DD statements shown above could be coded as follows:

```
//INPUT DD DUMMY,DCB=BLKSIZE=500
//READ DD DUMMY
```

When you are ready to perform I/O operations on your data set, simply omit the DUMMY parameter and code all parameters required to extend the data set. For example,

```
//INPUT DD DSNAME=*.STEP1.DDA,DISP=(MOD,PASS),LABEL=(,NL),
// DCB=BLKSIZE=500
//READ DD DSNAME=*.STEP4.ABC,DISP=(MOD,PASS)
```

Cataloged Data Set

A cataloged data set is a nontemporary data set defined with the DISP=(,CATLG) parameter or cataloged by the IEHPROGM utility program. The parameters shown in Table 22 are used to extend a cataloged data set. Remember that you can extend any sequential cataloged data set, but you can only add members to a partitioned data set.

This section is summarized in Table 73 of Appendix D.

Data Set Information

The DSNAME and DISP parameters are required. DSNAME identifies the data set. DISP specifies if you are retrieving or extending the data set.

DSNAME: The format of the DSNAME parameter is:

```
DSNAME= {dsname              }
        {dsname(membername)  }
```

Replace the term "dsname" with name of the cataloged data set. If the name is a fully qualfied name you must give all levels of qualification. For example if you created the data set as

```
   DSNAME=PLUS
```

you must again code

```
   DSNAME=PLUS
```

If you coded

```
   DSNAME=SYST.GLOBE
```

you must again code

```
   DSNAME=SYST.GLOBE
```

If you are adding a member to a partitioned data set, you must code the new member name in parentheses after the data set name. The member name consists of one to eight alphameric or national (@,#,$) characters. The first character must be a letter or national character. For example,

```
   DSNAME=ORB(MOON)
```

or

```
   DSNAME=SYST.GLOBE(SUN)
```

DISP: The format of the DISP parameter is:

```
                  [,UNCATLG] [,UNCATLG]
DISP=(MOD         [,DELETE ] [,DELETE ])
                  [,CATLG  ] [,CATLG  ]
                  [,       ] [        ]
```

You must code MOD. The second positional parameter tells the system what is to be done with the data set at the end of the job step. If you omit this subparameter the data set remains cataloged. If you code UNCATLG the data set will be uncataloged but not deleted. The next time you retrieve an uncataloged data set you must follow the directions given for "Kept Data Set." If you code DELETE the data set will be deleted. The only time you need to code CATLG is when you are extending the data set and the data set may require additional volumes. In this case, when you code

```
DISP=(MOD,CATLG)
```

the system updates the catalog entry to include the new volume serial numbers.

The third positional subparameter tells the system what is to be done with the data set if the step ABENDs. You can specify UNCATLG or DELETE and, when you are extending the data set, you can specify CATLG.

Note: When you are adding a member to a partitioned data set, the disposition (UNCATLG, DELETE, or CATLG) you specify applies to the entire data set and not to the one member. If you want to delete a particular member, you must use the IEHPROGM utility program.

In the following example, you are extending a cataloged data set named HULA.HOOP.

```
//DD9 DD DSNAME=HULA.HOOP,DISP=MOD
```

In the following example, you are adding a member to a cataloged data set named SYS1.FORTLIB.

```
//READ DD DSNAME=SYS1.FORTLIB(USERRTN),DISP=MOD
```

In the following example you are extending a cataloged data set named AREA. After extending it you want to uncatalog the data set.

```
//DDZ DD DSNAME=AREA,DISP=(MOD,UNCATLG)
```

Location of the Data Set

The system obtains unit and volume information from the catalog. The UNIT, VOLUME, and LABEL parameters are only needed in the following cases:

1. If you need more than one unit for your data set, use the UNIT parameter.
2. If you need additional volumes, either specify a new volume count with the VOLUME parameter or deferred mounting with the UNIT parameter. If you specify deferred mounting the system will give you a new volume each time you need it.
3. If you want your data set assigned to the same devices assigned to an earlier data set in the jobstep, request unit affinity with the UNIT parameter.
4. If you want the volume to be private you should use the VOLUME parameter.
5. If the cataloged data set does not have standard labels, you must use the LABEL parameter to specify the label type.

UNIT: The UNIT parameter should be used only if you need more than one unit for your data set or if you want unit affinity.

Requesting Units: The system assigns one device to your data set. If you want more devices assigned, code in the unit count subparameter the total number of devices needed. If you need an indefinite number of new volumes code the DEFER subparameter. The format of the UNIT parameter for cataloged data sets is:

```
UNIT=(,[unit count][,DEFER])
      [P         ]
```

Replace "unit count" with the number of devices needed for the data set. Code P if you want the same number of devices as there are volume serial numbers in the catalog entry.

Code DEFER if you need an indefinite number of new volumes to extend the data set. If you code DEFER, code the PRIVATE subparameter in the VOLUME parameter. If you do not code DEFER you should code the VOLUME parameter to indicate the total number of volumes you need to extend the data set.

In the following example, you are extending a cataloged data set named ALPHA.BETA that resides on three volumes. You think two more volumes may be needed and indicate this fact with the VOLUME parameter. You request one extra device with the UNIT parameter.

```
//EXT DD DSNAME=ALPHA.BETA,DISP=MOD,UNIT=(,4),VOLUME=(,,,5)
```

Using the same example, if you do not know how many extra volumes you need, but want the system to mount as many volumes as you need using the same maximum of four units, code:

```
//EXT DD DSNAME=ALPHA.BETA,DISP=MOD,UNIT=(,4,DEFER),VOLUME=PRIVATE
```

Using the same example, if you want the new volumes cataloged, code:

```
//EXT DD DSNAME=ALPHA.BETA,DISP=(MOD,CATLG),UNIT=(,4,DEFER),
//       VOLUME=PRIVATE
```

Unit Affinity: To conserve the number of units used in a job step, you can request that the cataloged data set be assigned to the same device or devices as assigned to an earlier data set in the same step. This technique, known as unit affinity, indicates that you want the volumes on which the cataloged data set resides and the volumes used by the earlier data set to be mounted on the same devices in sequential order. Unit affinity implies deferred mounting for one of the volumes since both volumes cannot be mounted at the same time. Before you request unit affinity, make sure that both data sets use the same type of removable volumes, for example, both data sets could be on 2314 volumes, or both on 2400 volumes. It does not make sense to request unit affinity where both data sets are on different types of volumes.

The format of the UNIT parameter for requesting unit affinity is:

```
UNIT=AFF=ddname
```

Replace the term "ddname" with the name of an earlier DD statement in the job step. Whenever it is required, the volume of the earlier data set will be demounted and the cataloged data set will be mounted on the same unit. If the ddname refers to a DD statement that defines a dummy data set, the data set defined on the DD statement requesting unit affinity is assigned a dummy status.

When unit affinity is requested for two data sets that reside on different 2321 volumes, the data sets are assigned the same device but can be assigned different bins.

In the following example, the DD statement named DD2 requests that the system assign the same number of units to this data set as it assigns to the data set defined on the statement named DD1. Since CC1 requests two devices, these two devices are assigned to the data set defined by DD2.

```
//STEPC EXEC PGM=OVER
//DD1   DD   UNIT=(2400,2),VOLUME=(,,,2)
//DD2   DD   DSNAME=ALPHA.OMEGA,DISP=MOD,UNIT=AFF=DD1
```

VOLUME: The VOLUME parameter is needed only when one or more of the following conditions are met:

1. You want the volume on which the cataloged data set resides to be private. A private volume is one that cannot be allocated to a new temporary data set that makes a nonspecific request for a public volume.

2. You want the private volume to remain mounted at the end of the job step.

3. You want to begin processing with a certain volume when you are retrieving or extending a multivolume cataloged data set.

4. You need additional volumes to extend the data set. (You cannot request additional volumes for a partitioned data set.)

The format of the VOLUME parameter for cataloged data sets is:

```
VOLUME=([PRIVATE][,RETAIN][,sequence] [,volcount])
                 [,    ][,        ]
```

The PRIVATE subparameter means that you want the volume(s) on which the cataloged data set resides to be private. The RETAIN subparameter means that the volume will not be demounted at the end of the step. PRIVATE must be coded when you code RETAIN. (RETAIN has the same effect on volume mounting as coding PASS in the DISP parameter.) Private volumes are demounted at the end of the step unless it is an unremovable volume, such as a 2301, or unless it has been designated by your installation to be a permanently resident volume, for example, the system residence volume. It is recommended that you use the RETAIN subparameter whenever you want the volume to remain mounted at the end of the step.

For example if you want the volume that contains the SMITH.JOHN data set to be private, code:

```
//OLD DD DSNAME=SMITH.JOHN,DISP=MOD,VOLUME=PRIVATE
```

Using the same example, if you want the volume to remain mounted at the end of the step, code:

```
//OLD DD DSNAME=SMITH.JOHN,DISP=MOD,VOLUME=(PRIVATE,RETAIN)
```

When you are extending a multivolume cataloged data set, you can begin processing with other than the first volume of the data set by coding the sequence number subparameter. Replace "sequence#" with a number from 1 to 255 that indicates the sequence number of the volume you want. For example, if you want the third volume of the cataloged data set, code

```
VOLUME=(,,3)
```

The sequence number is a positional subparameter and must follow the PRIVATE and RETAIN subparameters or the commas that indicate their absence. If you omit the sequence number subparameter and the volume count subparameter follows, you must indicate its absence by a comma.

The volume count subparameter is used only when you want to request additional volumes for the data set. Replace "volcount" with the maximum number of volumes that can be used for the data set. If you code "volcount" but do not request the same number of devices in the UNIT parameter, you must code the PRIVATE subparameter.

In the following example, you are extending a cataloged data set named SYSTEM.TWO. SYSTEM.TWO now resides on four volumes and you think you may need two more. Only four units are to be used. The new volumes are to be cataloged.

```
//IN DD DSNAME=SYSTEM.TWO,DISP=(MOD,CATLG),VOLUME=(PRIVATE,,,6)
```

Using the same example, you want to use one device for each unit. (In this case you do not have to code PRIVATE unless you want all six volumes to be private.)

```
//IN DD DSNAME=SYSTEM.TWO,DISP=(MOD,CATLG),UNIT=(,6),
//      VOLUME=(PRIVATE,,,6)
```

Using the same example, you want to use one device for each unit. (In this case you do not have to code PRIVATE unless you want all six volumes to be private.)

```
//IN DD DSNAME=SYSTEM.TWO,DISP=(MOD,CATLG),UNIT=(,6),
//      VOLUME=(,,,6)
```

Using the same example, if you want to begin processing with the second volume, code:

```
//IN DD DSNAME=SYSTEM.TWO,DISP=(MOD,CATLG),UNIT=(,6),
//      VOLUME=(,,2,6)
```

LABEL: The LABEL parameter is needed only if the cataloged data set does not have standard labels. You must code the same label type you specified when you defined the data set. No other subparameters are allowed. The format of the LABEL parameter for cataloged data sets is:

code:

SUL
if the data set has standard and user labels.

NL
if the data set has no labels.

NSL
if the data set has nonstandard labels.

BLP
to bypass label processing.

In the following example, a cataloged data set named A.B.D is extended. A.B.D has nonstandard labels.

```
//INPUT DD DSNAME=A.B.D,DISP=MOD,LABEL=(,NSL)
```

Size of the Data Set

You can use the SPACE parameter to override any secondary quantity specified when you created the data. This new secondary quantity is in effect only for the duration of the job step. Note that if you did not specify a secondary quantity when you created the data set you cannot do so now.

SPACE: The format of the SPACE parameter is:

```
SPACE=({TRK       },(1,secondary quantity))
       {CYL       }
       {block length}
```

The first positional parameter indicates whether you are requesting space in tracks (TRK), cylinders (CYL), or blocks. If you are using blocks, replace "blocklength" with the average block length of your data. You do not have to use the same type of unit of space you used when you created the data set.

Any primary quantity you code is ignored. However, for reasons of syntax you must code some number, such as 1, as the second positional subparameter.

Replace the term "secondary quantity" with the amount of space you desire in the units specified.

In the following example you are extending a cataloged data set named COLOR.GREEN.APPLE. COLOR.GREEN.APPLE has a secondary allocation of 10 tracks. You want to use a secondary allocation of 50 tracks for the duration of the job step.

```
//DD79 DD DSNAME=COLOR.GREEN.APPLE,DISP=MOD,SPACE=(TRK,(1,50))
```

Data Attributes

The DCB parameter is needed only if the cataloged data set is on magnetic tape and has nonstandard labels or no labels. Follow the instructions given for "Magnetic Tape" in the section "Creating A New Data Set." If the data set has standard labels, you can specify the subparameters shown in Tables 18, 19, and 21 for "Cataloged Data Sets" in the section "Retrieving an Existing Data Set."

Special Processing Options

There are two special processing options for cataloged data sets:

1. You can request channel separation from other data sets in the same job step using either the SEP or the AFF parameter.

2. You can suppress I/O operations on a sequential data set (magnetic tape or direct access) using the DUMMY parameter.

SEP: When two or more data sets are to be used in a job step, processing time may be shortened by requesting that the system transmit data sets over separate channels. For example, it would be faster to have your input data set and your output data set on separate channels than to have them on the same channel.

You can request channel separation for data sets in each job step using the SEP parameter. If possible, the system will honor this request. It may not always be possible to honor the request for separation because given devices may not be available for allocation when the job step is executed or because there may not be enough devices of a given type of different channels for all the data sets that request separation. If channel separation is not requested, the system will assign any available channel that has the device specified with the UNIT parameter.

The format of the SEP parameter is:

```
SEP=(ddname,...)
```

Replace the terms "ddname" with the names of up to eight earlier DD statements in the same job step. The earlier DD statements can define any type of data set: new or existing, on magnetic tape or direct access, temporary or nontemporary, etc.

In the following example, you request that the data set defined by the FOUR DD statements be assigned to a channel other than the ones assigned to the data sets defined by the ONE and TWO DD statement. The data sets defined by ONE and TWO may or may not be on the same channel. The data set defined by THREE may or may not be on the same channel as any of the other three data sets.

```
//PROG1 EXEC PGM=PROCESS
//ONE   DD   UNIT=2400,VOLUME=PRIVATE,LABEL=(,NL)
//TWO   DD   DSNAME=SYSTEM.USER.ABC,DISP=MOD
//THREE DD   DSNAME=HELP,DISP=(,KEEP),UNIT=2314,
//            VOLUME=SER=ALF,LABEL=RETPD=31
//FOUR  DD   DSNAME=END,DISP=OLD,UNIT=2400-2,
//            VOLUME=SER=TAPE75,SEP=(ONE,TWO)
```

Using the preceding example, if you want to request that the data set defined by the THREE DD statement be on a channel other than the one assigned to the data set defined by the TWO DD statement code:

```
//PROG1 EXEC PGM=PROCESS
//ONE   DD   UNIT=2400,VOLUME=PRIVATE,LABEL=(,NL)
//TWO   DD   DSNAME=SYSTEM.USER.ABC,DISP=MOD
//THREE DD   DSNAME=HELP,DISP=(,KEEP),UNIT=2314,
//            VOLUME=SER=AFL,LABEL=RETPD=31,SEP=TWO
//FOUR  DD   DSNAME=END,DISP=OLD,UNIT=2400-2,
//            VOLUME=SER=TAPE75,SEP=(ONE,TWO)
```

As a result of this example, the data sets defined by ONE and TWO may or may not be on the same channel. The data set defined by THREE and FOUR may or may not be on the same channel, but neither one will be on the same channel as the data set defined by TWO. Also, the data set defined by FOUR will not be on the same channel as the data set defined by ONE.

Remember that the request for channel separation made with the SEP parameter is only a suggestion to the system for more efficent operation, and not a requirement. If your data set must be on separate channels specify unit addresses with the UNIT parameter. You can obtain a configuration chart with all the unit addresses from your manager or supervisor.

AFF: The AFF parameter provides a shortcut method of requesting channel separation. When two or more data sets in the same job step have the same channel separation requirements, you can code the SEP parameter for the first data set, and then request the same operation using the AFF parameter in the later data sets. The AFF parameter tells the system that you want the data set defined on this DD statement to have the same channel separation as the data set defined in the named DD statement.

The format of the AFF parameter is:

```
AFF=ddname
```

Replace "ddname" with the name of the earlier DD statement that requests the desired channel separation. This data set and the earlier data set may or may not be on the same channel.

In the following example, the data sets defined by DD1 and DD2 may or may not be on the same channel. The data sets defined by DD3 and DD5 may or may not be on the same channel, but neither one will be on the same channel as the data sets defined by DD1, DD2, or DD5.

```
//STEP EXEC PGM=METHOD2
//DD1  DD   ...
//DD2  DD   ...
//DD3  DD   SEP=(DD1,DD2),...
//DD4  DD   SEP=DD3,...
//DD5  DD   AFF=DD3,...
```

The requests for channel separation and affinity are only suggestions to the system. They will be honored if the appropriate devices are available.

DUMMY: The DUMMY parameter allows you to bypass I/O operations and data set disposition. When your processing program asks to extend the dummy data set, the write request is recognized, but no data is transmitted. This facility is particularly useful when you are debugging your program because it saves you processing time. For example, you can suppress the extending of a data set until you are sure your program is going to produce meaningful output.

The DUMMY parameter is a positional parameter. That means that it must be coded first in the operand field. For example:

```
//IN DD DUMMY,DSNAME=ABLE,DISP=MOD,LABEL=(,NL),DCB=BLKSIZE=100
//DDT DD DUMMY,DSNAME=BAKER,DISP=MOD
```

If you wish, you can omit omit all other parameters required to define your data set except DCB. (If you were not going to write the DCB parameter you do not have to write it.) For example, the two DD statements shown above could be coded as follows:

```
//IN DD DUMMY,DCB=BLKSIZE=100
//DDT DD DUMMY
```

When you are ready to perform I/O operations on your data set, simply omit the DUMMY parameter and code all parameters required to retrieve or extend the data set. For example,

```
//IN DD DSNAME=ABLE,DISP=MOD,LABEL=(,NL),DCB=BLKSIZE=100
//DDT DD DSNAME=BAKER,DISP=MOD
```

Kept Data Set

A kept data set is a nontemporary data set defined with the DISP=(,KEEP) parameter. The parameters shown in Table 22 are used to retrieve or extend a kept data set. Remember that you can extend any sequential data set, but you can only add members to a partitioned data set.

This section is summarized in Table 74 of Appendix D.

Data Set Information

The DSNAME and DISP parameters are required.

DSNAME: The format of the DSNAME parameter is:

```
DSNAME={dsname              }
       {dsname(membername)  }
```

Replace the term "dsname" with the name of the kept data set. For example, if you created the data set as

```
DSNAME=NONO
```

you must again code

```
DSNAME=NONO
```

If you are adding a member to a partitioned data set, you must code the new membername in parentheses after the data set name. The member name consists of one to eight alphameric or national (@,$,#) characters. The first character must be a letter or national character. For example,

```
DSNAME=NOVA(STAR1)
```

DISP: The format of the DISP parameter is:

```
DISP=(MOD [,KEEP   ][KEEP   ])
          [,CATALG ][,CATLG  ]
          [,DELETE ][,DELETE ]
          [,       ][       ]
```

You must code MOD.

The second positional subparameter tells the system what is to be done with the data set at the end of the job step. If you omit this subparameter the data set remains kept. If you code CATLG the data set will be cataloged. If you code DELETE the data set is deleted.

The third positional subparameter tells the system what is to be done with the data set if the step ABENDs. If you omit this subparameter the data set remains kept. You can specify instead CATLG or DELETE.

Note: When you are adding members to a partitioned data set, the disposition (KEEP, CATLG, or DELETE) you specify applies to the entire data set. If you want to delete a particular member, you must use the IEHPROGM utility program.

In the following example you are extending a kept data set named FRANK. FRANK resides on a 2400 volume named ABCXYZ.

```
//#75 DD DSNAME=FRANK,DISP=MOD,UNIT=2400,VOLUME=SER=ABCXYZ
```

In the following example you are adding a member to a kept data set named FILE, FILE resides on a 2311 volume named PACK70. You want to catalog the entire partitioned data set after you add the member.

```
//AB DD DSNAME=FILE(MEM12),DISP=(MOD,CATLG),UNIT=2311,
//      VOLUME=SER=PACK70
```

Location of the Data Set

The UNIT and VOLUME parameters are required. The LABEL parameter is required only if the data set does not have standard labels or if it is not the first data set on a tape volume.

UNIT: You can use the UNIT parameter to directly specify units for your data set or to request the same units used for an earlier data set in the job step (unit affinity).

Requesting Units: The format of the UNIT parameter for directly specifying units for your data set is:

```
UNIT=({unit address}[,unit count]                    [,DEFER])
      {device type }[,P         ]
      {group name  }[,          ]
```

The first positional subparameter identifies the tape drive or direct access device you want to use for your data set by its address, or unit name, or group name.

unit address
: is the actual machine address of the direct access device. For example,

```
UNIT=190
```

You should not specify the address unless you are sure you want it. Do not specify an address if you are going to need more than one unit for the data set. (Multiple units are requested with the second subparameter of the UNIT parameter.)

device type
: Corresponds to the type of device. Coding a device type provides you with a certain degree of device independence in that your data set may be placed in any number of devices of the same type. For example, if you code

```
UNIT=2311
```

your data set uses any 2311 Disk Storage Drive. The following device types can be specified.

Device Type	Description
2301	2301 Drum Storage Unit
2302	2302 Disk Storage Drive
2303	2303 Drum Storage Unit
2311	Any 2311 Disk Storage Drive

2314	2314 Storage Facility
2321	Any bin mounted on a 2321 data cell device
2400	2400 series Nine-Track Magnetic Tape Drive that can be allocated to a data set written or to be written in 800 bpi density when the dual density feature is not installed on the drive, or in 1600 bpi when the dual-density feature is installed in the drive.
2400-1	2400 series Magnetic Tape Drive with Seven-Track Compatibility and without Data Conversion.
2400-2	2400 series Magnetic Tape Drive with Seven-Track Compatibility and Data Conversion.
2400-3	2400 series Nine-Track Magnetic Tape Drive that can be allocated to a data set written or to be written in 1600 bpi density.
2400-4	2400 series Nine-Track Magnetic Tape Drive having an 800 and 1600 bpi capability.

groupname
is the name of a collection of devices, selected by your installation during system generation. For example, your installation might select the name TAPE for all tape drives in the configuration. If you do not care which tape drive is used for your data set, you would then code

```
UNIT=TAPE
```

Your manager or supervisor should tell you which group names were generated for your installation.

The second positional subparameter is used only if your data set occupies more than one volume and if you want more than one of these volumes to be mounted at the same time. This subparameter indicates how many drives are to be used for mounting your data set's volumes.

unit count
indicates how many units you want assigned to the data set. You can specify a maximum of 59 units per DD statement. Make sure that your system has at least the number of units you specify. Otherwise an error will result. If you specify fewer units than there are volumes in your data set, only the same number of volumes as there are units can be mounted at the same time. (If you request fewer units than volumes code the PRIVATE subparameter in the VOLUME parameter.) If you specify the same number of units as there are volumes, all volumes can be mounted at the same time. (If you are extending your data set and you think your data set may use more volumes than you expect, you should indicate the maximum number of volumes that can be used with the volume count subparameter of the VOLUME parameter. Do not request any more units than the maximum number of volumes you specify.) For example, if you code

```
UNIT=(2301,2)
```

your data set will be assigned two 2301 drum storage units.

P
specifies parallel mounting. When you request parallel mounting the system counts the number of serial numbers specified with the VOLUME parameter, you also indicate a maximum number of volumes

that can be used by the data set, the number of units assigned is the number of serial volumes you specify and not the larger maximum number of volumes that can be used.

DEFER is the third positional subparameter. DEFER requests the system to assign the required tape drives or direct access devices to the data set and to defer the mounting of the volume or volumes until the processing program attempts to open the data set. If you are running under MFT or MVT, operating efficiency may decrease if you specify DEFER for a tape data set. This is because if you code DEFER, the system issues a mount message to the operator when the processing program attempts to use the data set and then waits until the volume is mounted. If you do not code DEFER, the mount message is issued when the device is assigned and there is no waiting for the operator. By the time your program tries to use the data set, it is very likely that the operator has already mounted the volume and that the system will not have to wait for him. DEFER has a special use if you think you may need additional volumes to extend your data set. If you specify DEFER the system will give you a new volume each time you need it. If you code DEFER, code the PRIVATE subparameter in the VOLUME parameter. If you do not code DEFER, you should code the VOLUME parameter to indicate the total number of volumes you need to extend the data set.

Unit Affinity: To conserve the number of units used in a job step, you can request that the kept data set be assigned to the same device or devices as assigned to an earlier data set in the same step. This technique, known as unit affinity, indicates that you want the volumes on which the kept data set resides and the volumes used by the earlier data set to be mounted on the same devices in sequential order. Unit affinity implies deferred mounting for one of the volumes since both volumes cannot be mounted at the same time. Before you request unit affinity, make sure that both data sets use the same type of removable volumes; for example, both data sets could be on 2314 volumes, or both on 2400 volumes. It does not make sense to request unit affinity where both data sets are on different types of volumes.

The format of the UNIT parameter for requesting unit affinity is:

```
UNIT=AFF=ddname
```

Replace the term "ddname" with the name of an earlier DD statement in the job step. Whenever it is required, the volume of the earlier data set will be mounted and the cataloged data set will be mounted on the same unit. If the ddname refers to a DD statement that defines a dummy data set, the data set defined on the DD statement requesting unit affinity is assigned a dummy status.

When unit affinity is requested for two data sets that reside on different 2321 volumes, the data sets are assigned the same device but can be assigned different bins.

In the following example, the DD statement named DD2 requests that the system assign the same number of units to this data set as it assigned to the data set defined on the statement named DD1. Since DD1 requests two devices, these two devices are assigned to the data set defined by DD2.

```
//STEPC EXEC PGM=OVER
//DD1   DD   UNIT=(2400,2),VOLUME=(,,,2)
//DD2   DD   DSNAME=ALPHA.DISP=MOD,
//           VOLUME=SER=(TAPE22,TAPE23),UNIT=AFF=DD1
```

VOLUME: The format of the VOLUME parameter for kept data sets is:

```
VOLUME=([PRIVATE][,RETAIN][,,volcount,]SER=(serial,...))
```

The PRIVATE subparameter means that you want the volume(s) on which the kept data set resides to be private. (A private volume is one that cannot be allocated to a new temporary data set that makes a nonspecific request for a public volume.) The RETAIN subparameter means that the volume will not de demounted at the end of the step. PRIVATE must be coded when you code RETAIN. (RETAIN has the same effect on volume mounting as coding PASS in the DISP parameter.) Private volumes are demounted at the end of the step unless it is an unrecoverable volume, such as a 2301, or unless it has been designated by your installation to be a permanently resident volume, for example, the system residence volume. It is recommended that you use the RETAIN subparameter whenever you want the volume to remain mounted at the end of the step. If there are more volumes than devices, only the last volumes mounted simultaneously are retained.

The volume count subparameter is used only when you want to request additional volumes for the data set. (You cannot request additional volumes for a partitioned data set.) Replace the term "volcount" with the maximum number of volumes the data set can use. You should only use this subparameter if you think your data set may use more volumes than those named in the SER subparameter. Note that "volcount" is the maximum number of volumes that can be used and not additional volumes. For example, if you name three volumes in the SER subparameter, but you think you may need two more volumes, replace "volcount" with 5. Ask the operator for the serial numbers of the volumes assigned, because you will have to know them the next time you retrieve or extend your data set.

The SER subparameter identifies the serial numbers of the volumes on which your data set resides. Replace the term "serial" with the 1-to-6 character serial number associated with the volume. The volume serial number can contain any alphameric and national characters and the hyphen. If it is necessary to include special characters, other than a hyphen, in the volume serial number, enclose it in apostrophes, for example:

```
VOLUME=SER=(54AB2,'6/10/8')
```

If only one volume is involved you need not code the parentheses, for example:

```
VOL=SER=ABCDEF
```

In the following example you extend a kept data set named A99 that resides on volumes AA22 and AB45. Only one tape drive named TAPE will be used and the last volume used is to remain mounted at the end of the step.

```
//DD9 DD DSNAME=A99,DISP=MOD,UNIT=TAPE,
//       VOLUME=(PRIVATE,RETAIN,SER=(AA22,AB45))
```

In the following example, you extend the same data set, A99, but request that both volumes be mounted in parallel.

```
//DD99 DD DSNAME=A99,DISP=MOD,UNIT=(TAPE,P),
//        VOLUME=(PRIVATE,RETAIN,SER=(AA22,AB45))
```

In the following example you are extending a kept data set named PARTS. PARTS resides on two 2311 volumes named 7945 and 7946. Only one device is used, so you have to code PRIVATE.

```
//P4 DD DSNAME=PARTS,DISP=MOD,UNIT=2311,
//      VOLUME=(PRIVATE,SER=(7945,7946)
```

Using the same example, if you think you may need one more volume to extend the data set, code:

```
//P4 DD DSNAME=PART,DISP=MOD,UNIT=2311,
//      VOLUME=(PRIVATE,,,3,SER=(7945,7946))
```

In the following example you are extending a kept data set named WHOLE. WHOLE resides on three 2314 volumes named 12345, 12346, and 12347. Three 2314 units are used.

```
//P5 DD DSNAME=WHOLE,DISP=MOD,UNIT=(2314,3),
//      VOLUME=SER=(12345,12346,12347)
```

Using the same example, if you need an indefinite number of volumes to extend the data set you should code DEFER. You must also code PRIVATE because you are not increasing the number of devices.

```
//P5 DD DSNAME=WHOLE,DISP=MOD,UNIT=(2314,3,DEFER),
//      VOLUME=(PRIVATE,SER=(12345,12346,12347))
```

In the case of multivolume data sets, give only the serial numbers of the volumes you want to process in the order you want to process them. For example, you want to extend a kept data set named LIST. LIST resides on two 2314 volumes named 231410 and 231411. You want to process 231411 only.

```
//D9 DD DSNAME=LIST,DISP=MOD,
//        UNIT=2314,VOLUME=SER=231411
```

Using the same example, you need one more volume to extend the data set and request volume 231412. Only one device is used, so you have to code PRIVATE.

```
//D9 DD   DSNAME=LIST,DISP=MOD,UNIT=2314,
//        VOLUME=(PRIVATE,SER=(231411,231412))
```

LABEL: The LABEL parameter is used only if the kept data set does not have standard labels, or if the data set is not the first data set on a tape volume.

The format of the LABEL parameter for kept data sets is:

```
LABEL=([sequence#][,SUL]
                  [,NL ]
                  [,NSL]
                  [,BLP])
```

The first positional subparameter indicates the order of your data set on the tape volume. You only have to specify a sequence number if your data set is not the first data set on the volume. If you do not specify a sequence number and there are other data sets on the volume, your data set cannot be retrieved and the step will abnormally terminate. Replace

the term "sequence#" with the sequence number of your data set. For example, if your data set is the third data set on the volume, code:

```
LABEL=3
```

Note: If you request the system to bypass label processing (specify BLP in the second positional subparameter of the LABEL parameter), the system treats anything within tape marks as a data set. Therefore, if you want your data set to be written in the proper sequence, you must include all header and trailer labels and data sets that precede your data set in the sequence number subparameter.

The second positional subparameter indicates the label type of your data set.

Code:

SUL
if the data set has standard and user labels.

NL
is the data set has no labels.

NSL
if the data set has nonstandard labels.

BLP
to bypass label processing.

In the following example, a kept data set named BLUE is extending BLUE as the fourth data set on a tape volume named TAPE2. The data set has standard labels.

```
//T77 DD DSNAME=BLUE,DISP=MOD,UNIT=2400,
//       VOLUME=SER=TAPE2,LABEL=4
```

Using the same example, if the data set has no labels, code:

```
//T77 DD DSNAME=BLUE,DISP=MOD,UNIT=2400,
//       VOLUME=SER=TAPE2,LABEL=(4,NL)
```

In the following example you extend a kept data set named RED that has nonstandard labels. RED resides on a tape volume named 47922A.

```
//INPUT DD DSNAME=RED,DISP=MOD,UNIT=2400,
//         VOLUME=SER=47922A,LABEL=(,NSL)
```

Size of the Data Set

You can use the SPACE parameter to override any secondary quantity specified when you created the data. This new secondary quantity is in effect only for the duration of the job step. Note that if you did not specify a secondary quantity when you created the data set you cannot do so now.

SPACE: The format of the SPACE parameter is:

```
SPACE=({TRK       },(1,secondary quantity))
       {CYL       }
       {block length}
```

The first positional parameter indicates whether you are requesting space in tracks (TRK), cylinders (CYL), or blocks. If you are using blocks, replace "blocklength" with the average block length of your data. You do not have to use the same type of unit of space you used when you created the data set.

Any primary quantity you code is ignored. However, for reasons of syntax you must code some number, such as 1, as the second positional parameter.

Replace the term "secondary quantity" with the amount of space you desire in the units specified.

In the following example you are extending a kept data set named BAG. BAG has a secondary allocation of 3 cylinders. You want to use a secondary allocation of 5 1024-byte blocks for the duration of the job step. BAG resides on a 2311 volume named MMM942.

```
//EXT DD DSNAME=BAG,DISP=MOD,UNIT=2311,VOLUME=SER=MMM942,
//       SPACE=(1024,(1,5))
```


Data Attributes

The DCB parameter is needed only if the kept set is on magnetic tape and has no labels or nonstandard labels. Follow the instructions given for "Magnetic Tape" in the section "Creating a New Data Set." If the data set has standard labels, you can specify the subparameters shown in Tables 18, 19, and 21 for "Cataloged Data Sets" in the section on "Retrieving an Existing Data Set."

Special Processing Options

There are two special processing options for kept data sets:

1. You can request channel separation from other data sets on the same job step using either the SEP or AFF parameter.
2. You can suppress I/O operations on a sequential data set (magnetic tape or direct access) using the DUMMY parameter.

SEP: When two or more data sets are to be used in a job step, processing time may be shortened by requesting that the system transmit data sets over separate channels. For example, it would be faster to have your input data set on separate channels than to have them on the same channel.

You can request channel separation for the data sets of each job step using the SEP parameter. If possible, the system will honor this request. It may not always be possible to honor the request for

separation because given devices may not be available for allocation when the job step is executed or because there may not be enough devices of a given type on different channels for all the data sets that request separation. If channel separation is not requested, the system will assign any available channel that has the device specified with the UNIT parameter.

The format of the SEP parameter is:

```
SEP=(ddname,...)
```

Replace the terms "ddname" with the names of up to eight earlier DD statements in the same job step. The earlier DD statements can define any type of data set: new or existing, on magnetic tape or direct access, temporary or nontemporary, etc.

In the following example, you request that the data set defined by the FOUR DD statements be assigned to a channel other than the ones assigned to the data sets defined by the ONE and TWO DD statement. The data sets defined by ONE and TWO may or may not be on the same channel. The data set defined by THREE may or may not be on the same channel as any of the other three data sets.

```
//PROG1 EXEC PGM=PROCESS
//ONE   DD   UNIT=2400,VOLUME=PRIVATE,LABEL=(,NL)
//TWO   DD   DSNAME=SYSTEM.USER.ABC,DISP=MOD
//THREE DD   DSNAME=HELP,DISP=(,KEEP),UNIT=2314,
//            VOLUME=SER=AFL,LABEL=RETPD=31
//FOUR  DD   DSNAME=END,DISP=OLD,UNIT=2400-2,
//            VOLUME=SER=TAPE75,SEP=(ONE,TWO)
```

Using the preceding example, if you want to request that the data set defined by the THREE DD statement be on a channel other than the one assigned to the data set defined by the TWO DD statement code:

```
//PROG1 EXEC PGM=PROCESS
//ONE   DD   UNIT=2400,VOLUME=PRIVATE,LABEL=(,NL)
//TWO   DD   DSNAME=SYSTEM.USER.ABC,DISP=MOD
//THREE DD   DSNAME=HELP,DISP=(,KEEP),UNIT=2314,
//            VOLUME=SER=AFL,LABEL=RETPD=31,SEP=TWO
//FOUR  DD   DSNAME=END,DISP=OLD,UNIT=2400-2,
//            VOLUME=SER=TAPE75,SEP=(ONE,TWO)
```

As a result of this example, the data sets defined by ONE and TWO may or may not be on the same channel. The data set defined by THREE and FOUR may or may not be on the same channel, but neither one will be on the same channel as the data set defined by TWO. Also, the data set defined by FOUR will not be on the same channel as the data set defined by ONE.

Remember that the request for channel separation made with the SEP parameter is only a suggestion for more efficient operation, and not a requirement. If your data set must be on separate channels specify unit addresses with the UNIT parameter. You can obtain a configuration chart with all the unit addresses from your manager or supervisor.

AFF: The AFF parameter provides a shortcut method of requesting channel separation. When two or more data sets in the same job step have the same channel separation requirements, you can code the SEP parameter for the first data set, and then request the same separation using the AFF parameter in the later data sets. The AFF parameter tells the system that you want the data set defined on this DD statement to have the same channel separation as the data set defined in the named DD statement.

The format of the AFF parameter is:

```
AFF=ddname
```

Replace "ddname" with the name of the earlier DD statement that requests the desired channel separation. This data set and the earlier data set may or may not be on the same channel.

In the following example, the data sets defined by DD1 and DD2 may or may not be on the same channel. The data sets defined by DD3 and DD5 may or may not be on the same channel, but neither one will be on the same channel as the data sets defined by DD1 and DD2. The data set defined by DD4 will not be on the same channel as the data set defined by DD3, but it may be on the same channel as the data sets defined by DD1, DD2, or DD5.

```
//STEP EXEC PGM=METHOD2
//DD1  DD   ...
//DD2  DD   ...
//DD3  DD   SEP=(DD1,DD2),...
//DD4  DD   SEP=DD3,...
//DD5  DD   AFF=DD3,...
```

The requests for channel separation and affinity are only suggestions to the system. They will be honored if the appropriate devices are available.

DUMMY: The DUMMY parameter allows you to bypass I/O operations and data set disposition. When your processing programs asks to extend the dummy data set, the write request is recognized, but no data is transmitted. This facility is particularly useful when you are debugging your program because it saves you processing time. For example, you can suppress the extending of a data set until you are sure your program is going to produce meaningful output.

The DUMMY parameter is a positional parameter. That means that it must be coded first in the operand field. For example:

```
//INTAPE DD DUMMY,DSNAME=BC74,DISP=MOD,UNIT=2400,
//          VOLUME=SER=7942,LABEL=(,NSL),DCB=DEN=2

//P79 DD    DUMMY,DSNAME=TOM,DISP=MOD,UNIT=2311,
//          VOLUME=SER=A27941
```

If you wish, you can omit all other parameters required to define your data set except DCB. (If you were not going to write the DCB parameter you do not have to write it.) For example, the two DD statements shown above could be coded as follows:

```
//INTAPE DD DUMMY,DCB=DEN=2

//P79 DD DUMMY
```

When you are ready to perform I/O operations on your data set, simply omit the DUMMY parameter and code all parameters required to retrieve or extend the data set. For example:

```
//INTAPE DD DSNAME=BC74,DISP=MOD,UNIT=2400,
//          VOLUME=SER=7942,LABEL=(,NSL),DCB=DEN=2

//P79 DD DSNAME=TOM,DISP=MOD,UNIT=2311,VOLUME=SER=A27941
```

Special DD Statements

You can use the DD statement facilities to define three types of special data sets. They are:

1. Private libraries

   - JOBLIB DD Statement
   - STEPLIB DD Statement

2. Data Sets for abnormal termination dumps

   - SYSABEND DD Statement
   - SYSUDUMP DD Statement

3. Checkpoint data set

   - SYSCHK DD Statement

Private Libraries

The most frequently used processing programs reside in the system library SYS1.LINKLIB. However, you may want to place a program in a private library for one of several reasons:

- It is used infrequently.
- It is not completely checked out.
- It is used only by a limited number of people.
- You wish to transport it from one location to another.

To retrieve a program from a private library, the library must be made available. You do this by defining a private library for the entire job or for a particular job step, using one of two special DD statements. The JOBLIB DD statement is used to define a private library for the entire job and the STEPLIB DD statement is used to define a library for a job step. When the operating system encounters one of these statements, it effectively concatenates the private library with the system library for the duration of the job or job step. When a request is made for a program the system searches for the program, first in the private library, and then in the system library.

Note: Private libraries are partitioned data sets. Each member of the library is a program. (See the description of the PGM parameter in the section "The EXEC Statement.")

JOBLIB DD Statement

The JOBLIB DD statement defines a private library for the duration of a job. The parameters you code on the DD statement depend on whether the library exists or whether you are creating it in the same job. In either case you must follow these rules:

1. The ddname must be JOBLIB. Never use the ddname JOBLIB except when you are defining a private library.

2. The JOBLIB DD statement must appear immediately after the JOB statement to which it pertains.

3. A JOBLIB DD statement cannot appear in a cataloged procedure.

The Library Exists: The private library can exist as a cataloged data set or as a kept data set. If it is a cataloged data set you must use the DSNAME and DISP parameters in the JOBLIB DD statement. If it is a kept data set, you must use the DSNAME, DISP, UNIT, and VOLUME parameters. The DSNAME, UNIT, and VOLUME parameters are coded as described for "Cataloged Data Sets" and "Kept Data Sets" in the section "Retrieving an Existing Data Set." The format of the DISP parameter is:

```
DISP=({OLD}[,PASS])
      {SHR}
```

You must code either OLD or SHR. OLD means that the library already exists. SHR means that the library already exists and that it may be used by other jobs that are executing concurrently (MFT or MVT). PASS means that the library is kept at the end of the job. You can omit PASS if you wish. No other subparameters are allowed.

If you wish to refer to the private library in a later DD statement, you can code

```
DSNAME=*.JOBLIB
```

and the DISP parameter,

```
DISP=({OLD}{,PASS }
      {SHR}{,KEEP }
           {,CATLG})
```

(Do not assign a disposition of DELETE, because the library would then be deleted at the end of the job step and be unavailable for use during the remainder of the job.) If a later DD statement defines a data set that is to be placed on the same volume as the private library, you can code

```
VOLUME=REF=*.JOBLIB
```

to obtain volume and unit information.

In the following example, the private library defined in the JOBLIB DD statement is cataloged. The CHECK DD statement refers to the private library defined in the JOBLIB DD statement.

```
//ATWL   JOB
//JOBLIB DD   DSNAME=SYSTEM.LIB4.DAILY,DISP=OLD
//FIRST  EXEC PGM=NUM7
.
.
.
//SECOND EXEC PGM=NUM9
//CHECK  DD   DSNAME=*.JOBLIB,DISP=(OLD,PASS)
.
.
.
```

In the following example, the private library defined in the JOBLIB DD statement is not cataloged. The UPDT DD statement refers to the private library and catalogs it. This means that the next time you use the private library in another job you only have to use the DSNAME and DISP parameters in its JOBLIB DD statement.

```
//ALL    JOB
//JOBLIB DD   DSNAME=PRIV.LIB12,DISP=SHR,UNIT=2301,
//           VOLUME=SER=AB2497
//STEP1  EXEC PGM=ONE
//UPDT   DD   DSNAME=*.JOBLIB,DISP=(SHR,CATLG),
//           VOLUME=REF=*.JOBLIB
.
.
.
//STEP2  DD   PGM=FOUR
.
.
.
```


You can concatenate other libraries to the one defined in the JOBLIB DD statement. That is, you can arrange a sequence of DD statements that define different libraries. The libraries are searched in the order in which the DD statements appear. If the system library is not defined on one of these DD statements, it is searched last.

To concatenate libraries, omit the ddname from all the DD statements defining the libraries except the first DD statement. The first DD statement must specify a ddname of JOBLIB, and the entire group must appear immediately after the JOB statement.

In the following example, several private libraries are concatenated. The system searches for each program in this order: LIBS.LETTERS.ABC,SERIES2.GROUP5,PROGRAMS, before searching SYS1.LINKLIB. PROGRAMS is kept, the other two are cataloged.

```
//CHECKUP JOB
//JOBLIB  DD  DSNAME=LIBS.LETTERS.ABC,DISP=OLD
//        DD  DSNAME=SERIES2.GROUP5,DISP=OLD
//        DD  DSNAME=PROGRAMS,DISP=OLD,UNIT=(2314,2),
//            VOLUME=SER=(231407,231408)
```

Creating the JOBLIB: You can create a private library and use it in the same job. To do this you must use two DD statements. The first is the JOBLIB DD statement. The second is a DD statement located in the job step that creates the library. You can have other DD statements that add more members to the library. Make sure that any programs executed in your job before you create the private library are in SYS1.LINKLIB.

The JOBLIB DD statement must have the DSNAME and DISP parameters. The format of the DISP parameter is:

```
DISP=( NEW[,PASS ] )
       [,CATLG]
```

NEW indicates that you will create the library later on in the job. PASS means that the library is to deleted at the end of the job. CATLG means that library is to be cataloged.

The DD statement that actually defines the library must use the same DSNAME and DISP parameters as the JOBLIB DD statement and all other parameters required to create a partitioned data set. (See "Direct Access Devices" in the section "Creating a New Data Set.")

The following example shows how to create and use a private library in the same job. The private library is named SET4.GENPROG.DEPT639 and is to be cataloged. The library is defined in the NEW DD statement in STEP1. The first member of the library, named DAY, is also created. When STEP2 is executed a second member, named NIGHT, is added to the library. The programs used in the job are found as follows: The HOUR program is executed in STEP1 and it is found in SYS1.LINKLIB. The DAY program is executed in STEP2 and it is found in the newly created SET4.GENPROG.DEPT639. The NIGHT program is executed in STEP3 and it is also found in the private library.

```
//MYJOB  JOB
//JOBLIB DD   DSNAME=SET4.GENPROG.DEPT639,DISP=(NEW,CATLG)
//STEP1  EXEC PGM=HOUR
//NEW    DD   DSNAME=SET4.GENPROG.DEPT639(DAY),DISP=(NEW,CATLG),
//            UNIT=2314,VOLUME=SER=2314AA,SPACE=(TRK,(20,2,2))
.
.
.
//STEP2  EXEC PGM=DAY
//MORE   DD   DSNAME=SET4.GENPROG.DEPT639(NIGHT),DISP=MOD
.
.
.
//STEP3  EXEC PGM=NIGHT
.
.
.
```

STEPLIB DD Statement

The STEPLIB DD statement defines a private library to be used in a job step. If you include a STEPLIB DD statement in a job step, the system first looks in the private library for the program the job step uses; if the program is not there, the system looks in SYS1.LINKLIB.

If both JOBLIB and STEPLIB DD statements appear in a job, the STEPLIB definition has precedence, i.e., the private library defined by the JOBLIB DD statement is not searched for the step that contains the STEPLIB definition. If you want the JOBLIB definition ignored but the step does not require use of another private library, define the system library on the STEPLIB DD statement:

```
//STEPLIB DD DSNAME=SYS1.LINKLIB,DISP=OLD
```

The private library must exist before the step is executed. The parameters you use in the STEPLIB DD statement depend on whether the library is cataloged, kept, or passed from a previous job step. In any case, you must follow these rules:

1. The ddname must be STEPLIB. Never use the ddname STEPLIB except when you are defining a private library.
2. A STEPLIB DD statement can appear in any position among the DD statements for the step.
3. The library defined on a STEPLIB DD statement can be referred to by or passed to later job steps in the same job.

4. A STEPLIB DD statement can appear in a cataloged procedure.

5. You may concatenate private libraries.

The Library is Cataloged: If the private library is cataloged you must code the DSNAME and DISP parameters as described for "Cataloged Data Sets" in the section "Retrieving An Existing Data Set."

- The DSNAME parameter specifies the name of the private library.
- The DISP parameter specifies the library's status, either OLD or SHR, and its disposition. The disposition would be KEEP, UNCATLG, DELETE, or PASS, depending on how you want the library treated after its use in the job step.

The following example shows a job with three steps. The UPDT program used in STEP1 could reside in the PRIV.LIB2 cataloged private library or in SYS1.LINKLIB. The CRTW program used in STEP2 resides in SYS1.LINKLIB. During STEP2, a private library named PRIV.LIB3 is created and cataloged. The first member, MEMB1, of this library is also created. The MEMB1 program used in STEP3 resides in PRIV.LIB3.


```
//AJOB    JOB
//STEP1   EXEC PGM=UPDT
//STEPLIB DD   DSNAME=PRIV.LIB2,DISP=SHR
.
.
.
//STEP2   EXEC PGM=CRTW
//NEW     DD   DSNAME=PRIV.LIB3(MEMB1),DISP=(,CATLG),
//             UNIT=2311,VOLUME=SER=AAAAAA,SPACE=(CYL,(5,1,2))
.
.
.
//STEP3   EXEC PGM=MEMB1
//STEPLIB DD   DSNAME=PRIV.LIB3,DISP=OLD
.
.
.
```

The Library is Kept: If the private library is kept, you must use the DSNAME, DISP, UNIT, and VOLUME parameter as described in "Kept Data Sets" in the section "Retrieving An Existing Data Set".

- The DSNAME parameter specifies the name of the private library.
- The DISP parameter specifies the library's status, either OLD or SHR, and its disposition. The disposition would be KEEP, CATLG, DELETE, or PASS depending on how you want the library treated after its use in the job step.
- The VOLUME parameter identifies the volume serial number.
- The UNIT parameter specifies the device to be allocated to the library.

The following example shows a job with five steps. The MULT program used in STEP1 can reside in the INST.SERIES2 library (defined with the JOBLIB DD statement) or in SYS1.LINKLIB. The DIV program used in STEP2 can reside in the USER.PLIB private library or in SYS1.LINKLIB. USER.PLIB is a kept data set but is cataloged in STEP2. The SUBT

program used in STEP3 resides in SYS1.LINKLIB since the JOBLIB definition is ignored and the library defined in the STEPLIB DD statement is SYS1.LINKLIB. The ADD program used in STEP4 can reside in the USER.PLIB or in SYS1.LINKLIB. USER.PLIB is now cataloged and only the DSNAME and DISP parameters are needed in the DD statement. The MATRIX program used in STEP5 can reside in the INST.SERIES2 library or in SYS1.LINKLIB.

```
//JONAS   JOB
//JOBLIB  DD   DSNAME=INST.SERIES2,DISP=SHR
//STEP1   EXEC PGM=MULT
.
.
.
//STEP2   EXEC PGM=DIV
//STEPLIB DD   DSNAME=USER.PLIB,DISP=(OLD,CATLG),UNIT=2314,
//            VOLUME=SER=LIBVOL
.
.
.
//STEP3   EXEC PGM=SUBT
//STEPLIB DD   DSNAME=SYS1.LINKLIB,DISP=SHR
.
.
.
//STEP4   EXEC PGM=ADD
//STEPLIB DD   DSNAME=USER.PLIB,DISP=OLD
.
.
.
//STEP5   EXEC PGM=MATRIX
```

The Library is Passed: A library can be passed in two ways:

1. You create the library with the DISP=(NEW,PASS) parameter in one job step and you then retrieve it in the STEPLIB DD statement of a later job step, or

2. You retrieve a cataloged or kept private library in the STEPLIB DD statement of a job step and use a DISP=(OLD,PASS) or DISP=(SHR,PASS) parameter. Any other steps that use that private library then retrieve it as a passed data set and not as a cataloged or kept data set. (If you use the PASS subparameter the first time you retrieve the library, the volume that contains the library is not demounted at the end of the step.

If the private library is passed you must use the DSNAME and DISP parameters as described for "Passed Data Sets" in the section "Retrieving an Existing Data Set."

- The DSNAME parameter specifies either the name of the private library or a backward reference of the form *.stepname.ddname. If the DD statement that assigned a disposition of PASS occurs in a cataloged procedure, the backward reference must include the procedure step name, i.e.,

  *.stepname.procstepname.ddname

- The DISP parameter specifies a status of OLD or SHR and a disposition. The disposition would be KEEP, CATLG, UNCATLG, DELETE, or PASS depending on how you want the library treated after its use in the job step.

The following example shows a job with four steps. The ABA program used in STEP1 resides in SYS1.LINKLIB. During STEP1, a private library named MYLIB is created and passed. The first member, BAPA, is also created. The ABO program used in STEP2 resides in a cataloged private library named PRIVATE.LIB. PRIVATE.LIB is passed to other steps. The BAPA program used in STEP3 resides in MYLIB. MYLIB is cataloged at the end of the job step. The CHI program used in the STEP4 resides in PRIVATE.LIB.

```
//RV      JOB
//STEP1   EXEC PGM=ABA
//BUILD   DD   DSNAME=MYLIB(BAPA),DISP=(,PASS),UNIT=2302,
//             VOLUME=SER=BVB111,SPACE=(TRK,(96,,10))
.
.
.
//STEP2   EXEC PGM=ABO
//STEPLIB DD   DSNAME=PRIVATE.LIB,DISP=(OLD,PASS)
.
.
.
//STEP3   EXEC PGM=BAPA
//STEPLIB DD   DSNAME=*.STEP1.BUILD,DISP=(OLD,CATLG)
.
.
.
//STEP4   EXEC PGM=CHI
//STEPLIB DD   DSNAME=*.STEP2.STEPLIB,DISP=OLD
```


Data Sets for Abnormal Termination Dumps

Job steps subject to abnormal termination can take advantage of the operating system abnormal termination dumping facilities. To avail a job step of these facilities, you must include a special DD statement defining a data set on which the dump can be written. This DD statement must be identified by one of the special ddnames SYSABEND or SYSUDUMP and must include appropriate parameters for a new data set on a unit record device (printer), a system output device, magnetic tape or a sequential data set on a direct access device as described in the section "Creating a New Data Set." The processing program must not make a reference to such a data set. If more than one SYSABEND or SYSUDUMP DD statement is included in a job step, all but the first SYSABEND or SYSUDUMP DD statement are ignored.

The dump provided when the SYSABEND DD statement is used includes the system nucleus, the problem program storage area, and a trace table, if the trace table option was requested at system generation (PCP and MFT only). The SYSUDUMP DD statement provides only a dump of the problem program storage area.

If you define a data set on printer or a system output device, the dump data set is produced immediately. If you define a data set on magnetic tape or direct access, you are actually storing the dump so you can write it at a later time. When you store the dump you must give a conditional disposition of KEEP or CATLG in the DISP parameter. This is because the conditional disposition is used only when the job abnormally terminates and the dump is produced only when the job abnormally terminates.

In the following example, the SYSABEND DD statement specifies that you want the dump routed through output class D.

```
//STEP4 EXEC PGM=N
//SYSABEND DD SYSOUT=D
.
.
.
```

Using the same example, assume that you are running under MFT or MVT. The dump is to be stored temporarily on 15 tracks of a 2314 volume. If you omit the UNIT and SPACE parameter the system assigns you a default unit and allocation.

```
//STEP4 EXEC PGM=N
//SYSABEND DD SYSOUT=D,UNIT=2314,SPACE=(TRK,15)
```

In the following example, the dumps produced by two steps are stored on the same data set. In both steps a conditional disposition of KEEP is specified. This allows storing of the dump if either step abnormally terminates. If both steps are successfully executed, the data set is deleted in the second step.

```
//STEP1 EXEC PGM=ONE
//SYSUDUMP DD DSNAME=DUMP,DISP=(,PASS,KEEP),UNIT=2400,
//            VOLUME=SER=17924
.
.
.
//STEP2 EXEC PGM=TWO
//SYSUDUMP DD DSNAME=*.STEP1.SYSUDUMP,DISP=(MOD,DELETE,KEEP)
.
.
.
```

In the following example, STEP1 specifies the dump is to be stored if the step abnormally terminates. Because COND=ONLY is specified in STEP2, the step is executed only if STEP1 abnormally terminates. STEP2 uses a program that prints the dump.

```
//STEP1 EXEC PGM=PLOP
//SYSUDUMP DD  DSNAME=DUMP,DISP=(,DELETE,KEEP),
//             UNIT=2314,VOLUME=SER=24935,SPACE=(TRK,(20,10))
.
.
.
//STEP2    EXEC PGM=PRINT,COND=ONLY
//INPUT    DD   DSNAME=DUMP,DISP=(OLD,DELETE),
//              VOLUME=REF=*.STEP1.SYSUDUMP
```

Note: For further information on abnormal termination dumps, refer to the publication, IBM System/360 Operating System: Programmer's Guide to Debugging, GC28-6670.

Checkpoint Data Set

If CHKPT macro instructions were executed during the original execution of your processing program, checkpoint entries were written on a checkpoint data set. If you resubmit your job for restart, and execution is to be restarted at a particular checkpoint, you must include a DD statement named SYSCHK. The SYSCHK DD statement defines the data set on which the checkpoint entry was written. The parameters you code on the SYSCHK DD statement depend on whether the data set was cataloged or kept. In either case, you must follow these rules:

1. The ddname must be SYSCHK. SYSCHK can be used as the ddname of DD statements in jobs.

2. The SYSCHK DD statement must immediately precede the first EXEC statement of the resubmitted job when restart is to begin at a checkpoint.

   (Do not use the SYSCHK DD statement when restart is to begin at a step.)

3. The SYSCHK DD statement must follow a DD statement named JOBLIB, if one is present.

4. The RESTART parameter must be coded on the JOB statement; otherwise, the SYSCHK DD statement has no effect.

5. The parameters you code on the SYSCHK DD statement are determined by whether the checkpoint data set is cataloged.

The Checkpoint Data Set is Cataloged

If the checkpoint data set is cataloged, you must always code the DSNAME and DISP parameters.

- The DSNAME parameter specifies the name of the checkpoint data set. If the checkpoint data set is partitioned, do not include a member name in the DSNAME parameter.

- The DISP parameter must specify or imply a status OLD and disposition of KEEP.

Other parameters you might code are VOLUME, UNIT, LABEL, and DCB.

- If the checkpoint entry exists on a tape volume other than the first volume of the checkpoint data set, you must indicate this by coding the volume serial number or volume sequence number in the VOLUME parameter (the serial number of the volume on which a checkpoint entry was written). If you code the volume serial number, you must also code the UNIT parameter, since the system will not look in the catalog for unit information.

- Code the LABEL parameter if the checkpoint data set does not have standard labels.

- Code DCB=TRTCH=C if the checkpoint data set is on 7-track magnetic tape with nonstandard labels or no labels.

In the following example the checkpoint data set is cataloged and follows the JOBLIB DD statement.

```
//C429    JOB  RESTART=(*,CK7)
//JOBLIB DD    DSNAME=PRIV.LIB7,DISP=SHR
//SYSCHK DD    DSNAME=CHECK,DISP=(OLD,KEEP)
//STEP1   EXEC ...
```

In the following example, the checkpoint entry resides on the third tape volume of the checkpoint data set. The volumes have nonstandard labels and use 7-track tape.

```
//RERUN   JOB  RESTART=(STEP3,POINT6)
//SYSCHK DD    DSNAME=CHK.PNT2,DISP=OLD,VOLUME=(,,3),
//             LABEL=(,NSL),DCB=TRTCH=C
//STEP1   EXEC ...
```

The Checkpoint Data Set is Kept

- The DSNAME parameter specifies the name of the checkpoint data set. If the checkpoint data set is partitioned, do not include a member name in the DSNAME parameter.

- The DISP parameter must specify or imply a status of OLD and disposition of KEEP.

- The VOLUME parameter specifies the volume serial number of the volume on which the checkpoint entry resides. (The serial number of the volume on which a checkpoint entry was written is contained in the console message printed after the checkpoint entry is written.)

- The UNIT parameter specifies the device to be allocated to the data set.

Other parameters you might code are LABEL and DCB.

- Code the LABEL parameter if the checkpoint data set does not have standard labels.

- Code DCB=TRTCH=C if the checkpoint data set is on 7-track magnetic tape with nonstandard or no labels.

In the following example the checkpoint data set is kept and has no labels.

```
//TRY     JOB  RESTART=(STEP4,STOP2)
//SYSCHK DD    DSNAME=CHK,DISP=OLD,UNIT=2400,
//             VOLUME=SER=AB927,LABEL=(,NL)
//STEP1   EXEC ...
```

Postponing Definition of a Data Set

The DDNAME parameter allows you to postpone defining a data set until later in the same job step. In the case of cataloged procedures, this parameter allows you to postpone defining a data set in the procedure until the procedure is called by a job step.

The DDNAME parameter is most often used in cataloged procedures and in job steps that call procedures. It is used in cataloged procedures to postpone defining data in the input stream until a job step calls the procedure. (Procedures cannot contain DD statements that define data in the input stream, i.e., DD * or DD DATA statements.) It is used in job steps that call procedures to postpone defining data in the input stream on an overriding DD statement until the last overriding DD statement for a procedure step. (Overriding DD statements must appear in the same order as the corresponding DD statements in the procedure, but in PCP a statement that defines data in the input stream must be the last overriding DD statement for a procedure step.)

The format of the DDNAME parameter is

```
DDNAME=ddname
```

Replace the term "ddname" with the name of a following DD statement in t defines this data set.

The following rules apply to the DDNAME parameter:

1. If you want to make a backward reference (i.e., *.ddname), you must refer to the statement that contains the DDNAME parameter, not to the statement that has the matching ddname. If the backward reference occurs before a statement with the matching ddname is encountered, it refers to a dummy data set.

2. If you want to concatenate data sets, the unnamed DD statements must follow the statement that contains the DDNAME parameter; they must not follow the statement that has the matching ddname.

3. You can use the DDNAME parameter up to five times in a job step and up to five times in a cataloged procedure step; however, each use must refer to a different ddname.

4. You cannot use the DDNAME parameter in a JOBLIB DD statement.

Two DCB subparameters can be coded with the DDNAME parameter -- BLKSIZE and BUFNO. This allows you to assign these DCB characteristics to the data set defined in the referenced DD statement. When the DCB subparameters BLKSIZE and BUFNO are coded both on the DD statement that contains the DDNAME parameter and on the referenced DD statement, the subparameters coded on the former are ignored.

For PCP, if the BLKSIZE or BUFNO subparameter is coded and the DDNAME parameter refers to a DD statement that defines data in the input stream (DD * or DD DATA statement), the subparameter is ignored.

For MFT and MVT, these subparameters would most often be coded with the DDNAME parameter when the referenced DD statement defines data in the input stream. Data in the input stream for MFT and MVT is written onto a direct access device, and the records are blocked as they are written. The input reader procedure normally assigns a block size and number of buffers for blocking. Coding the BLKSIZE subparameter allows you to specify that you want shorter blocks. Coding the BUFNO

subparameter allows you to specify that you want fewer buffers. You cannot specify that you want larger blocks or more buffers than would be assigned by the input reader procedure. (When a job is submitted via remote job entry and the BUFNO subparameter is coded, the BUFNO subparameter is ignored.)

The first part of the following example is a procedure step named ABC in a cataloged procedure named BETTY. The second part of the example is your input deck. In your input deck you define DD1 as a data set in the input stream.

```
CATALOGED PROCEDURE STEP
//ABC        EXEC PGM=PROGB
//DD1        DD   DDNAME=INPUT
//DD2        DD   UNIT=2311,SPACE=(TRK,10)
INPUT DECK
//STEP3      EXEC PROC=BETTY
//ABC.INPUT DD *
.
.       input data
.
/*
```

In the following example, the NOW DD statement defines the data set for the LATER DD statement. The COPY DD statement in STEP2 makes a backward reference to this data set.

```
//STEP1 EXEC PGM=HELEN
//LATER DD   DDNAME=NOW
//OTHER DD   UNIT=1442
//ELSE  DD   DSNAME=MY.DATA,DISP=OLD
//NOW   DD   DSNAME=&&FILE,DISP=(,PASS),UNIT=2311,
//           SPACE=(CYL,10)
//STEP2 EXEC PGM=PARIS
//COPY  DD   DSNAME=*.STEP1.LATER,DISP=(OLD,DELETE)
.
.
.
```

In the following example, the STREAM DD statement defines the data set for the DEF DD statement. If the job is run under MFT or MVT, the DCB parameter coded with the DDNAME parameter are used to block the input data. If the job is run under PCP, the DCB parameter is ignored. The two data sets named B.B.B and DATA are concatenated with the data set in the input stream.

```
//ST25    EXEC PGM=JASON
//DEF     DD   DDNAME=STREAM,DCB=(BLKSIZE=1600,BUFNO=2)
//        DD   DSNAME=B.B.B,DISP=SHR
//        DD   DSNAME=DATA,DISP=OLD,UNIT=2400,VOLUME=SER=17942
//STREAM DD    *
.
.  input data
.
/*
```

The first part of the following example shows a procedure step named SETUP in a cataloged procedure named ARGOS. The second part of the example is your input deck. In your input deck you use the DDNAME parameter to override the DD1 DD statement and make it refer to a data set in the input stream. You also override the DD2 DD statement.

```
CATALOGED PROCEDURE STEP
//SETUP        EXEC PGM=CLOUD
//DD1          DD   DSNAME=LON,DISP=OLD
//DD2          DD   UNIT=2400-2
INPUT DECK
//REFER        EXEC PROC=ARGOS
//SETUP.DD1    DD   DDNAME=INPUT
//SETUP.DD2    DD   UNIT=2400
//SETUP.INPUT DD    *
.
.   input data
.
/*
```

The Delimiter Statement

When you submit data through an input stream, you must indicate to the system the beginning of the data and the end of the data. The beginning of the data is indicated by a DD * or DD DATA statement. In PCP, the end of the data is indicated by a delimiter statement. In MFT and MVT, the delimiter statement is not required if the data is preceded with a DD * statement. For further information on the use of the delimiter statement refer to "Input Stream" in the section "Retrieving an Existing Data Set."

The delimiter statement consists of the characters /* in columns 1 and 2 and the comments field.

The format of the delimiter statement is:

```
/* comments
```

The comments cannot be continued onto another statement.

The following example shows an input deck with an input stream data set.

```
//AJOB  JOB
//STEP1 EXEC PGM=ONE
//DD1   DD   UNIT=2400
//DD2   DD   DSNAME=JOHN.SMITH,DISP=MOD
//DD3   DD   *
.
.  input data
.
/*
```

The Null Statement

The null statement can be placed at the end of a job's JCL statements and data or at the end of all the statements in an input stream. The null statement tells the system that the job just read should be placed on the queue of jobs ready for processing. If there are any JCL statements or data between a null statement and the next JOB statement, these are flushed by the system.

If you do not follow your job's JCL statements and data with a null statement, the system places your job on the queue when it encounters another JOB statement in the input stream. If your job is the last job in the input stream and a null statement does not follow it, the system recognizes that this is the last job in the input stream and it places your job on the queue.

Although it is not technically necessary to use a null statement at the end of your job it is good practice to do so because it ensures against other people's mistakes. For example, if the programmer who wrote the job that is to run after yours forgot to include his JOB statement, all of his JCL statements are considered to be part of your job. If you include the null statement at the end of your job, your job will end where it is supposed to and you will not have to pay for the other programmer's mistake.

The null statement consists only of the characters // in columns 1 and 2. The remainder of the statement must be blank.

The format of the null statement is:

```
//
```

The following example illustrates the use of the null statement.

```
//MYJB  JOB  ,'C BROWN',MSGLEVEL=(1,1)
//STEP1 EXEC PROC=FIELD
//DD1   DD   UNIT=2400
//DD2   DD   *
.
.
.
.  input data
.
.
.
/*
//
```

The Comment Statement

The comment statement can be used to contain information that may be helpful to yourself or another person that may be running your job or reviewing your output listing.

The comment statement may appear anywhere except before the JOB statement. A comment statement cannot be continued, but if you have many comments to write you can use several comment statements.

The comment statement consists of the characters //* in columns 1, 2, and 3, and the comments field.

The format of the comment statement is:

```
//* comments
```

In the MSGLEVEL parameter, you can request an output listing of all the JCL statements processed in your job. If you do, you can identify comment statements by the appearance of *** in columns 1, 2, and 3.

The following example illustrates the use of the comments statement in your input deck.

```
//DEPT58 JOB
//*FOR INFORMATION ON THIS JOB, CONSULT PROGRAM
//*FILE NUMBER 73492 (OCTOBER 30, 1969)
//STEP1 EXEC PGM=ONE
//*STEP1-WRITTEN BY JOHN DOE
.
.
.
//STEP2 EXEC PGM=TWO
//*STEP2-WRITTEN BY HELEN JONES AND GUS SMITH
.
.
.
//*END OF JOB-ERRORS SHOULD BE REPORTED TO CHARLIE
```

The PROC Statement

The PROC statement is the first control statement in an in-stream procedure. Optionally, the PROC statement can also be the first control statement in a cataloged procedure. If a PROC statement is included in a cataloged procedure, it is used to assign default values for symbolic parameters in the procedure. In an in-stream procedure, the PROC statement is used to mark the beginning of the procedure and can be used to assign default values to symbolic parameters in the procedure. A default value appearing on a PROC statement can be overridden by assigning a value to the same symbolic parameter on the EXEC statement that calls the procedure.

The name field of the PROC statement is optional for cataloged procedures only. The PROC statement must contain the term PROC in its operation field. For cataloged procedures, the operand field must contain symbolic parameters and their default values. For in-stream procedures, the operand field is optional.

The PROC statement and symbolic parameters are explained in detail in Part III of this publication.

Naming the PROC Statement

The name field is optional for cataloged procedures only. The name field is required for in-stream procedures. When a name is coded, it must be from one to eight characters in length and can contain any alphameric or national (@, $, #,) character. However, the first character of the name must be a letter or national character and must begin in column 3.

The name that appears on the PROC statement of an in-stream procedure is the name of the procedure. It is this name that is used on an EXEC statement to call the procedure. For cataloged procedures, the name of the PROC statement does not have to be the name of the cataloged procedure although the same name may be used. The name of the cataloged procedure is the name of the member of SYS1.PROCLIB that contains the cataloged procedure.

Symbolic Parameters

To assign a value on a PROC statement to a symbolic parameter, code:

```
symbolic parameter=value
```

Omit the ampersand that precedes the symbolic parameter in the procedure.

You can also nullify a symbolic parameter on the PROC statement. Code:

```
symbolic parameter=
```

Omit the ampersand that precedes the symbolic parameter and do not follow the equal sign with a value.

There are some rules you should keep in mind as you assign values to symbolic parameters:

1. The value you assign can be any length, but it cannot be continued onto another statement.

2. If the value contains special character, enclose the value in apostrophes (the enclosing apostrophes are not considered part of the value). If the special characters include apostrophes, each must be shown as two consecutive apostrophes.

3. The value you assign cannot contain an ampersand.

4. If you assign more than one value to a symbolic parameter on the PROC statement, the first value encountered is assigned.

5. If the symbolic parameter is concatenated with some other information (e.g., &JOBNO.321), this information and the value you assign to the symbolic parameter cannot exceed a combined total of 120 characters.

In the following example, you are writing a cataloged procedure that has four symbolic parameters: &SET, &TAPE, &SERIAL, and &CLASS. Values are assigned to the symbolic parameters on the PROC statement. These values are used when the procedure is called and the values are not assigned to the symbolic parameters by the programmer.

```
//EXTEND PROC SET=DAILY,TAPE=2400,
//            SERIAL=CCC249,CLASS=A
//LOOKUP EXEC PGM=SEARCH
//DD1    DD   DSNAME=MASTER.&SET,DISP=OLD
//DD2    DD   DSNAME=TABLE,UNIT=&TAPE,VOLUME=SER=&SERIAL,
//            DISP=(,PASS)
//PRINT  EXEC PGM=POST
//DD1    DD   DSNAME=*.LOOKUP.DD2,DISP=(OLD,DELETE)
//DD2    DD   SYSOUT=&CLASS
```

The PEND Statement

A PEND (procedure end) statement is required for every in-stream procedure you include in the input stream. It is used to mark the end of the procedure. The name of the PEND statement is optional.

The PEND statement consists of the characters // in columns 1 and 2, an three fields -- the name field, the operation (PEND) field, and the comments field. If comments are coded, one or more blanks must separate the operation field from the comment field. The PEND statement cannot be continued.

The format of the PEND statement is:

```
//name PEND comments
```

The following examples show the PEND statement.

```
//ENDPROC PEND THIS STATEMENT IS REQUIRED
```

This PEND statement contains a comment; therefore, a blank is required to separate the operation field (PEND) from the comments field.

```
// PEND
```

A PEND statement can contain only the coded operation field preceded by // and one or more blanks.

The Command Statement

Commands are issued to communicate with and control the system. All commands may be issued to the system via the operator's console; some commands may be also issued via a command statement in the input stream. In most cases, the operator issues the command. If you include a command statement as part of your JCL statements, the command is usually executed as soon as it is read. Since this is so, in MFT and MVT it is not likely that the command will be synchronized with the execution of the job step to which it pertains. Therefore, you should tell the operator which commands you want issued and when they should be issued, and let him issue them.

A command statement may appear immediately before a JOB statement, an EXEC statement, a null statement, another command statement, or between DD statements.

The command statement consists of characters // in columns 1 and 2, and three fields -- the operation (command), operand, and comments field. The format of the command statement is:

```
┌──────────────────────────────────────────────────────────────────────┐
│  // command operand comments                                         │
└──────────────────────────────────────────────────────────────────────┘
```

The commands that can be entered through the input stream in PCP, MFT, or MVT are listed below, with a brief explanation of what each command requests the system to do. Most command statements consist of an operation (command) field and an operand field, which includes options associated with the command. The operand field is not described here; a complete discussion of the commands and operands is presented in the publication IBM System/360 Operating System: Operator's Guide.

PCP

In PCP, the following commands can be entered through the input stream.

DISPLAY: The DISPLAY command causes a console display of certain system status information.

MOUNT: The MOUNT command tells the system to assign a device so a particular volume can be mounted on it. This device can then be assigned by the system to any job step that requires that volume.

SET: The SET command is used to establish the values of certain variables, such as the time of day and the date.

START: The START command tells the system to start either an input reader or an output writer.

STOP: The STOP command tells the system to stop the console display effected by the DISPLAY command, or to stop an output writer.

UNLOAD: The UNLOAD command tells the system to remove the volume previously mounted after a MOUNT command was issued.

VARY: The VARY command tells the system to place an I/O device into an online or offline status.

MFT

In MFT, the following commands can be entered through the input stream.

CANCEL: The CANCEL command tells the system to immediately terminate the scheduling or execution of a job, to cancel a job on the queue, or to stop the writing of an output data set currently being processed by an output writer.

DISPLAY: The DISPLAY command causes a console display of certain system status information.

HOLD: The HOLD command causes the system to temporarily prevent one job or all jobs from being selected for processing.

LOG: The LOG command is used to enter information into the system log.

MODIFY: The MODIFY command tells the system to change the characteristics of a functioning output writer.

MOUNT: The MOUNT command tells the system to assign a device so a particular volume can be mounted on it. This device can then be assigned by the system to any job step that requires that volume.

RELEASE: The RELEASE command tells the system to resume job selection, which had been suspended by the HOLD command or TYPRUN=HOLD on the JOB statement.

REPLY: The REPLY command is used to reply to messages from the system or from a processing program that requests information.

RESET: The RESET command tells the system to change the class or priority, or both, of a job in an input, hold, or system output queue.

SET: The SET command is used to establish the values of certain variables, such as the time of day and the date.

START: The START command tells the system to start a particular system process, e.g., an input reader, graphic job processor, initiator, etc.

STOP: The STOP command tells the system to stop a system process that has been previously started by a START command, or to stop the console display affected by the DISPLAY command.

SWAP: The SWAP command allows Dynamic Device Reconfiguration of two volumes.

UNLOAD: The UNLOAD command tells the system to remove the volume previously mounted in response to a MOUNT command.

VARY: The VARY command tells the system to place an I/O device or path into an online or offline status.

WRITELOG: The WRITELOG command tells the system to have the system output writer write out the contents of the system log.

MVT

In MVT, the following commands can be entered through the input stream.

CANCEL: The CANCEL command tells the system to immediately terminate the scheduling or execution of a job, to cancel a job on the queue, or to stop the writing of an output data set currently being processed by an output writer.

DISPLAY: The DISPLAY command causes a console display of certain system status information.

HOLD: The HOLD command causes the system to temporarily prevent one job or all jobs from being selected for processing.

LOG: The LOG command is used to enter information into the system log.

MODIFY: The MODIFY command tells the system to change the characteristics of a functioning initiator or output writer.

MOUNT: The MOUNT command tells the system to assign a device so a particular volume can be mounted on it. This device can then be assigned by the system to any job step that requires that volume.

RELEASE: The RELEASE command tells the system to resume job selection, which had been suspended by the HOLD command or TYPRUN=HOLD on the JOB statement.

REPLY: The REPLY command is used to reply to messages from the system or from a processing program that requests information.

RESET: The RESET command tells the system to change the class, or priority, or both, of a job in an input, hold, or system output queue.

SET: The SET command is used to establish the values of certain variables, such as the time of day and the date.

START: The START command tells the system to start a particular system process, e.g, an input reader, graphic job processor, initiator, etc.

STOP: The STOP command tells the system to stop a system process that had been previously started by a START command or to stop the console display effected by the DISPLAY command.

SWAP: The SWAP command allows Dynamic Device Reconfiguration of two volumes.

UNLOAD: The UNLOAD command tells the system to remove the volume previously mounted in response to a MOUNT command.

VARY: The VARY command tells the system to place an I/O device or path into an online or offline status. In a system with Model 65 Multiprocessing (M65MP), this command is used to place I/O devices, CPU, channel, and storage units in online or offline status.

WRITELOG: The WRITELOG command tells the system to have the system output writer write out the contents of the system log.

Part II: JCL for Compilers, Linkage Editors, and Loader

Part II shows the JCL statements needed to compile and link edit or load your source program using the following IBM processing programs:

- ALGOL (Tables 23, 24 and 25)
- Assembler E (Tables 26, 27, and 28)
- Assembler F (Tables 29, 30, and 31)
- COBOL E (Tables 32, 33, and 34)
- COBOL F (Tables 35, 36, and 37)
- American National Standard COBOL (Tables 38, 39, and 40)
- FORTRAN E (Tables 41, 42, and 43)
- FORTRAN G (Tables 44, 45, and 46)
- FORTRAN H (Tables 47, 48, and 49)
- PL/I F (Tables 50, 51, and 52)
- Linkage editor (Tables 53, 54, and 55)
- Loader (Tables 56, 57, and 58)

There are three tables for each processing program. The first table shows the input deck. The second table shows the values that can be given to the PARM field of the EXEC statement. The third table shows how to code each DD statement in the input deck.

These tables are a summary of the information in the Programmer's Guide associated with each of the listed programs. They are presented here for your convenience in preparing a few sample decks while learning JCL. For further details and up-to-date information you should refer to the appropriate publication.

The JCL statements required to execute your program can only be determined by you. You should code them following the instructions in Part I of this publication.

A section with examples follows the tables.

ALGOL

Table 23. ALGOL - Input Deck

ALGOL			
//jobname	JOB	parameters	
//[name]	EXEC	PGM=ALGOL[,PARM=parameters]	Required
//SYSUT1	DD	parameters	Required
//SYSUT2	DD	parameters	Required
//SYSUT3	DD	parameters	Required
//SYSABEND	DD	parameters	Optional
//SYSPRINT	DD	parameters	Required
//SYSPUNCH	DD	parameters	Required
//SYSLIN	DD	parameters	Required
//SYSIN	DD	* or parameters	Required

Table 24. ALGOL - PARM Field of EXEC Statement

PARM	Meaning
{PROGRAM / PROCEDURE}	The source program is an ALGOL program. / The source program is an ALGOL procedure.
{SHORT / LONG}	Internal representation of real values is in fullwords. / Internal representation of real values is in doublewords.
{DECK / NODECK}	The object module is placed on the device specified in the SYSPUNCH DD statement. / The object module is not placed in the device specified by the SYSPUNCH DD statement.
{LOAD / NOLOAD}	The object module is placed on the device specified by the SYSLIN DD statement. / The object module is not placed on the device specified by the SYSLIN DD statement.
{SOURCE / NOSOURCE}	A source program and identifier table listing is produced. / The listing is not produced.
{EBCDIC / ISO}	The source program is keypunched in the EBCDIC character set. / The source program is keypunched in the standard character set defined by the International Standards Organization (ISO).
{TEST / NOTEST}	The object module is to contain testing information. / The object module will not contain testing information.
SIZE=number	Specifies the size of main storage, in bytes, that is available to the compiler.
Note: Underscore the default values selected by your installation during system generation.	

Table 25. ALGOL - DD Statements

ddname	Data Set Information	Location of the Data Set	Size of the Data Set	Data Attributes	Special Processing Options	Comments
SYSUT1 SYSUT2	Temporary. You can omit DSNAME and DISP.	Magnetic tape or direct access. UNIT is required. You can omit VOLUME and LABEL.	Required if on direct access.	Omit.	You can request separation (SEP)	Work data sets.
SYSUT3	Temporary. You can omit DSNAME and DISP.	Direct access. UNIT is required. You can omit VOLUME and LABEL.	Required	Omit.	Omit.	Work data sets.
SYSABEND		System output device, printer or intermediate storage. SYSOUT=A is usually specified.				Optional. Abnormal termination dump is produced on this data set. Code as described for "Abnormal Termination Dump" in the section "Special DD Statements."
SYSPRINT		System output device, printer or intermediate storage. SYSOUT=A is usually specified.		You can specify BLKSIZE.		Listings are produced on this data set.
SYSPUNCH		System output device, card punch, or intermediate storage. SYSOUT=B is usually specified.		You can specify BLKSIZE.	You can use DUMMY while testing the program.	Object module is produced on this data set. (Copied from SYSLIN.)
SYSLIN	If object module is to be link edited or loaded in a subsequent step of the same job use DSNAME=&&dsname and DISP=(,PASS). If it is to be link edited or loaded in another job, define a cataloged or kept data set.	Magnetic tape or direct access. UNIT is required. You can omit VOLUME and LABEL.	Required if on direct access.	You can specify BLKSIZE.	You can use DUMMY while testing the program.	Object module is produced on this data set.
SYSIN		Usually in the input stream; use *.		You can specify BLKSIZE.		Input to the ALGOL compiler (source statements).

Assembler E

Table 26. Assembler E - Input Deck

Assembler E			
//jobname	JOB	parameters	
//[name]	EXEC	PGM=IETASM[,PARM=parameters]	Required
//SYSLIB	DD	DSNAME=SYS1.MACLIB,DISP=SHR	Optional
//SYSUT1	DD	parameters	Required
//SYSUT2	DD	parameters	Required
//SYSUT3	DD	parameters	Required
//SYSPRINT	DD	parameters	Optional
//SYSPUNCH	DD	parameters	Optional
//SYSIN	DD	* or parameters	Required

Table 27. Assembler E - PARM Field of EXEC Statement

PARM=	Meaning
{DECK / NODECK}	The object module is placed on the device specified by the SYSPUNCH DD statement. An object module is not produced.
{LOAD / NOLOAD}	The object module is placed on the device specified in the SYSPUNCH DD statement. An object module is not produced.
{LIST / NOLIST}	An assembler listing is produced. An assembler listing is not produced.
{TEST / NOTEST}	The object module (if produced) contains the source symbol table required by TESTRAN. The object module does not contain the table required by TESTRAN.
{XREF / NOXREF}	A cross-reference table of symbols is listed. A cross-reference table is not produced.
LINECNT=nn	Specifies as a number from 01 to 99, the number of lines to be printed between headings in the listing.
Note: Underscore the default values selected by your installation during system generation.	

Table 28. Assembler E - DD Statements

ddname	Data Set Information	Location of the Data Set	Size of the Data Set	Data Attributes	Special Processing Options	Comments
SYSLIB	Code DSNAME=SYS1.MACLIB and DISP=SHR or DSNAME=userlibrary Alternatively, the user library may be concatenated to SYS1.MACLIB.	Omit.	Omit.	Omit.	Omit.	Optional. Existing partitioned data set.
SYSUT1 SYSUT2 SYSUT3	Temporary. You can omit DSNAME and DISP.	Magnetic tape or direct access. UNIT is required. You can omit VOLUME and LABEL.	Required if on direct access.	Omit.	You can request separation (SEP).	Work data sets.
SYSPRINT		System output device, printer, or intermediate storage. SYSOUT=A is usually specified.				Optional. Listings are produced on this data set.
SYSPUNCH	If object module is to be link edited or loaded in a subsequent step of the same job, use DSNAME= &&dsname and DISP= (,PASS). If it is to be link edited or loaded in another job, define a cataloged or kept data set. Omit DSNAME and DISP to produce a punched deck.	System output device, card punch, magnetic tape or direct access. SYSOUT=B can be used. If you do not use SYSOUT, code UNIT; you can omit VOLUME and LABEL.	Required if on direct access.		You can use DUMMY while testing the program.	Optional. Object module is produced on this data set.
SYSIN		Usually in the input stream; use *.				Input to the assembler (source statements).

Assembler F

Table 29. Assembler F - Input deck

Assembler F			
//jobname	JOB	parameters	
//[name]	EXEC	PGM=IEUASM[,PARM=parameters]	Required
//SYSLIB	DD	DSNAME=SYS1.MACLIB,DISP=SHR	Optional
//SYSUT1	DD	parameters	Required
//SYSUT2	DD	parameters	Required
//SYSUT3	DD	parameters	Required
//SYSPRINT	DD	parameters	Optional
//SYSPUNCH	DD	parameters	Optional
//SYSGO	DD	parameters	Optional
//SYSIN	DD	* or parameters	Required

Table 30. Assembler F - PARM Field of EXEC Statement

PARM=	Meaning
{DECK / NODECK}	The object module is placed on the device specified by the SYSPUNCH DD statement. The object module is not placed on the device specified by the SYSPUNCH DD statement.
{LOAD / NOLOAD}	The object module is placed on the device specified by the SYSGO DD statement. The object is not placed on the device specified by the SYSGO DD statement.
{LIST / NOLIST}	An assembler listing is produced. An assembler listing is not produced.
{TEST / NOTEST}	The object module contains the source symbol table required by TESTRAN. The object module does not contain the table required by TESTRAN.
{XREF / NOXREF}	A cross-reference table of symbols is listed. A cross-reference table is not produced.
{RENT / NORENT}	The source program is checked for coding violations of program reenterability. The source program is not checked.
LINECNT=nn	Specifies, as a number from 01 to 99, the number of lines to be printed between headings in the listing.
{ALGN / NOALGN}	The alignment error diagnostic message is not suppressed. The alignment error diagnostic message is suppressed.
{OS / DOS}	The assembler will have complete Operating System Assembler F capability. The assembler will behave like Disk Operating System (DOS) Assembler F. The DOS option is incompatible with the LOAD, TEST, RENT, or NOALGN options.
Note: Underscore the default values selected by your installation during system generation.	

Table 31. Assembler F - DD Statements

ddname	Data Set Information	Location of the Data Set	Size of the Data Set	Data Attributes	Special Processing Options	Comments
SYSLIB	Code DSNAME=SYS1.MACLIB and DISP=SHR or use DSNAME=userlibrary Alternatively, the user library may be concatenated to SYS1.MACLIB.	Omit.	Omit.	Omit.	Omit.	Optional. Existing partitioned data set.
SYSUT1 SYSUT2 SYSUT3	Temporary. You can omit DSNAME and DISP.	Magnetic tape or direct access. UNIT is required. You can omit VOLUME and LABEL.	Required if on direct access.	Omit.	You can request separation (SEP)	Work data sets.
SYSPRINT		System output device, printer, or intermediate storage. SYSOUT=A is usually specified.				Optional. Listings are produced on this data set.
SYSPUNCH		System output device, card punch or intermediate storage. SYSOUT=B usually specified.			You can use DUMMY while testing the program.	Optional. Object module is produced on this data set.
SYSGO	If object module is to be link edited or loaded in a subsequent step of the same job, use DSNAME= &&dsname and DISP= (,PASS). If it is to be link edited or loaded in another job, define a cataloged or kept data set.	Magnetic tape or direct access. UNIT is required. You can omit VOLUME and LABEL.	Required if on direct access.		You can use DUMMY while testing the program.	Optional. Same as SYSPUNCH but can be used as immediate input to the linkage editor or loader. You can define a punched deck as in SYSPUNCH.
SYSIN		Usually in the input stream; use *.				Input to the assembler (source statements).

COBOL E

Table 32. COBOL E - Input Deck

COBOL E			
//name			
//jobname	JOB	parameters	
//[name]	EXEC	PGM=IEPCBL00[,PARM=parameters]	Required
//SYSLIB	DD	parameters	Optional
//SYSUT1	DD	parameters	Required
//SYSUT2	DD	parameters	Required
//SYSUT3	DD	parameters	Required
//SYSUT4	DD	parameters	Optional
//SYSPRINT	DD	parameters	Required
//SYSPUNCH	DD	parameters	Required
//SYSIN	DD	* or parameters	Required

Table 33. COBOL E - PARM Field of EXEC Statement

PARM=	Meaning
BUFSIZE=	Indicates the size of each of the work buffers used during a COBOL compilation.
{DECK / NODECK}	The object module is written on the device specified in the SYSPUNCH DD statement. An object module is not produced.
{FLAGE / FLAGW}	The compiler suppresses warning diagnostic messages. The compiler will produce error and warning diagnostic messages.
{LIST / NOLIST}	A source listing is produced. A source listing is not produced.
{DMAP / NODMAP}	A data-name listing is produced. A data-name listing is not produced.
{PMAP / NOPMAP}	A listing of object code for each statement in the Procedure Division is produced. A listing of object code is not produced.
{MAPS / NOMAPS}	Equivalent to specifying both DMAP and PMAP. Equivalent to specifying both NODMAP and NOPMAP.
{DISPCK / NODISPCK}	The compiler generates object code that tests whether a field to be displayed exceeds the record length of the device on which it is to be written. The test code will not be generated.
{REGED / INVED}	Specifies that the character "." represents a decimal point and the character "," represents an insertion character. The comma represents a decimal point and the period an insertion character.
LINECNT=nn	Specifies, as a number from 10 to 99, the number of lines to be printed on each page of the compilation output listing.
Note: Underscore the default values selected by your installation during system generation.	

Table 34. COBOL E - DD Statements

ddname	Data Set Information	Location of the Data Set	Size of the Data Set	Data Attributes	Special Processing Options	Comments
SYSLIB	Code DSNAME=userlibrary and DISP=OLD or DISP=SHR.	Required if data set was kept.	Omit.			Optional. User source program library.
SYSUT1 SYSUT2 SYSUT3	Temporary. You can omit DSNAME and DISP.	Magnetic tape or direct access. UNIT is required. You can omit VOLUME and LABEL.	Required if on direct access.	Omit.	Omit.	Work data sets.
SYSUT4	Temporary. You can omit DSNAME and DISP.	Magnetic tape or direct access. UNIT is required. You can omit VOLUME and LABEL.	Required if on direct access.	Omit.	Omit.	Optional. Work data set needed when using debug packets.
SYSPRINT		System output device, printer, or intermediate storage. SYSOUT=A is usually specified				Listings are produced on this data set.
SYSPUNCH	If object module is to be link edited or loaded in a subsequent step of the same job, use DSNAME= &&dsname and DISP= (,PASS). If it is to be link edited or loaded in another job, define a cataloged or kept data set. Omit DSNAME and DISP to produce punched deck.	System output device, card punch, magnetic tape or direct access. SYSOUT=B can be used. If you do not use SYSOUT, code UNIT; you can omit VOLUME and LABEL.	Required if on direct access.		You can use DUMMY while testing this program.	Object module is produced on this data set.
SYSIN		Usually in the input stream; use *.				Input to the COBOL E compiler (source statements).

COBOL F

Table 35. COBOL F - Input Deck

COBOL F			
//jobname	JOB	parameters	
//[name]	EXEC	PGM=IEQCBL00[,PARM=parameters]	Required
//SYSLIB	DD	parameters	Optional
//SYSUT1	DD	parameters	Required
//SYSUT2	DD	parameters	Required
//SYSUT3	DD	parameters	Required
//SYSUT4	DD	parameters	Required
//SYSPRINT	DD	parameters	Required
//SYSPUNCH	DD	parameters	Optional
//SYSLIN	DD	parameters	Optional
//SYSIN	DD	* or parameters	Required

Table 36. COBOL F - PARM Field of EXEC Statement (Part 1 of 2)

PARM=	Meaning
SIZE=number	Indicates the amount of main storage, in bytes, available for compilation.
BUF=number	Indicates the amount of main storage, in bytes, to be allocated to buffers.
{SOURCE / NOSOURCE}	A listing of the source module is produced. A listing of the source module is not produced.
{CLIST / NOCLIST}	A condensed listing is to be produced. A condensed listing is not to be produced.
{MAP / NOMAP}	A listing of a glossary, global tables, object code listing, and assembler language expansion of the source module is to be produced. The listing is not to be produced.
{DMAP / NODMAP}	A listing of a glossary is to be produced. A listing of a glossary is not to be produced.
{PMAP / NOPMAP}	A listing of global tables and assembler language expansion is to be produced. The listing is not to be produced.
{LOAD / NOLOAD}	The object module is to be link edited or loaded. (Specify the SYSLIN DD statement.) The object module is not to be link edited or loaded.
{DECK / NODECK}	The object module is to be punched. (Specify the SYSPUNCH DD statement.) The object module is not to be punched.
{SEQ / NOSEQ}	The compiler is to check the sequence of the source statements. The compiler is not to check the sequence.
{FLAGW / FLAGE}	All warning and diagnostic messages are to be listed. Only diagnostic messages are to be listed.

Table 36. COBOL F - PARM Field of EXEC Statement (Part 2 of 2)

PARM=	Meaning
SUPMAP NOSUPMAP	The object code listing, object module and output deck are to be suppressed if an E or D level message is generated by the compiler. Those items are not to be suppressed.
BASIS NOBASIS	A BASIS card may be in the source module. A BASIS card is not in the source module.
COPY NOCOPY	A COPY or INCLUDE statement may be in the source module. A COPY or INCLUDE statement is not in the source module.
LINECNT=nn	Specifies, as a number from 01 to 99, the number of lines to be printed on each page of the compilation output listing.
SPACE1 SPACE2 SPACE3	The listing of the source module has single spacing. Double spacing. Triple spacing.
Note: Underscore the default values selected by your installation during system generation.	

Table 37. COBOL F - DD Statements

ddname	Data Set Information	Location of the Data Set	Size of the Data Set	Data Attributes	Special Processing Options	Comments
SYSLIB	Code DSNAME=userlibrary and DISP=OLD or DISP=SHR.	Required if data set was kept.	Omit.			Optional. User source program library.
SYSUT1	Temporary. You can omit DSNAME and DISP.	Direct access. UNIT is required. You can omit VOLUME and LABEL.	Required	Omit.	Omit.	Work data sets.
SYSUT2 SYSUT3 SYSUT4	Temporary. You can omit DSNAME and DISP.	Magnetic tape or direct access. UNIT is required. You can omit VOLUME and LABEL.	Required if on direct access.	Omit.	Omit.	Work data sets.
SYSPRINT		System output device, printer or intermediate storage. SYSOUT=A usually specifed.		You can specify BLKSIZE and LRECL.		Listings are produced on this data set.
SYSPUNCH		System output device, card punch or intermediate storage. SYSOUT=B usually specified.		You can specify BLKSIZE or LRECL.	You can use DUMMY while testing this program.	Optional. Object module is produced on this data set.
SYSLIN	If the object module is to be link edited or loaded in a subsequent step of the same job, use DSNAME=&&dsname and DISP=(,PASS). If it is to be link edited or loaded in another job, define a cataloged or kept data set.	Magentic tape or direct access. UNIT is required. You can omit VOLUME and LABEL.	Required if on direct access.	You can specify BLKSIZE or and LRECL.	You can use DUMMY while testing this program.	Optional. Same as SYSPUNCH but can be used as intermediate input to linkage editor or loader.
SYSIN		Usually in the input stream; use *.		You can specify BLKSIZE		Input to the COBOL F compiler (source statements).

American National Standard COBOL

Table 38. American National Standard COBOL - Input Deck

//jobname	JOB	parameters	
//[name]	EXEC	PGM=IKFCBL00[,PARM=parameters]	Required
//SYSLIB	DD	parameters	Optional
//SYSUT1	DD	parameters	Required
//SYSUT2	DD	parameters	Required
//SYSUT3	DD	parameters	Required
//SYSUT4	DD	parameters	Required
//SYSPRINT	DD	parameters	Required
//SYSPUNCH	DD	parameters	Optional
//SYSLIN	DD	parameters	Optional
//SYSIN	DD	* or parameters	Required

Table 39. American National Standard COBOL - PARM Field of EXEC Statement (Part 1 of 2)

PARM=	Meaning
SIZE=nunber	Indicates the amount of main storage, in bytes, available for compilation.
BUF=number	Indicates the amount of main storage, in bytes, to be allocated to buffers.
{SOURCE / NOSOURCE}	A listing of the source module is produced. A listing of the source module is not produced.
{CLIST / NOCLIST}	A condensed listing is to be produced. CLIST and PMAP are mutually exclusive. A condensed listing is not to be produced.
{XREF / NOXREF}	A cross-reference listing is to be produced. A cross-reference listing is not to be produced.
{DMAP / NODMAP}	A listing of a glossary is to be produced. A listing of a glossary is not to be produced.
{PMAP / NOPMAP}	A listing of global tables and assembler language expansion is to be produced. PMAP and CLIST are mutually exclusive. The listing is not to be produced.
{LOAD / NOLOAD}	The object module is to be link edited or loaded. (Specify the SYSLIN DD statement.) The object module is not to be link edited or loaded.
{DECK / NODECK}	The object module is to be punched. (Specify the SYSPUNCH DD statement.) The object module is not to be punched.
{SEQ / NOSEQ}	The compiler is to check the sequence of the source statements. The compiler is not to check the sequence.
{FLAGW / FLAGE}	All warning and diagnostic messages are to be listed. Only diagnostic messages are to be listed.

Table 39. American National Standard COBOL - PARM Field of EXEC Statement (Part 2 of 2)

PARM=	Meaning
SUPMAP NOSUPMAP	The object code listing, object module and output deck are to be suppressed if an E level message is generated by the compiler. Those items are not to be suppressed.
QUOTE APOST	The double quote (") is used to delineate literals and in the generation of figurative constants. The apostrophe (') is used to delineate literals in the generation of figurative constants.
TRUNC NOTRUNC	A COMPUTATIONAL arithmetic item is truncated to the nunber of digits specified in the PICTURE clause when moved. The movement of the item depends on the size of the field (halfword or fullword).
LINECNT=nn	Specifies, as a nunber from 01 to 99, the number of lines to be printed on each page of the compilation output listing.
SPACE1 SPACE2 SPACE3	The listing of the source module has single spacing. Double spacing. Triple spacing.
Note: Underscore the default values selected by your installation during system generation	

Table 40. American National Standard COBOL - DD Statements

ddname	Data Set Information	Location of the Data Set	Size of the Data Set	Data Attributes	Special Processing Options	Comments
SYSLIB	Code DSNAME=userlibrary and DISP=OLD or DISP=SHR	Required if data set was kept.	Omit.			Optional. User source program library.
SYSUT1	Temporary. You can omit DSNAME and DISP.	Direct access. UNIT is required. You can omit VOLUME and LABEL.	Required	Omit.	Omit.	Work data sets.
SYSUT2 SYSUT3 SYSUT4	Temporary. You can omit DSNAME and DISP.	Magnetic tape or direct access. UNIT is required. You can omit VOLUME and LABEL.	Required if on direct access.	Omit.	Omit	Work data sets.
SYSPRINT		System output device, printer or intermediate storage. SYSOUT=A usually specified.		You can specify BLKSIZE and LRECL.		Listings are produced on this data set.
SYSPUNCH		System output device, card punch or intermediate storage. SYSOUT=B usually specified.		You can specify BLKSIZE or LRECL.	You can use DUMMY while testing this program.	Optional. Object module is produced on this data set.
SYSLIN	If the object module is to be link edited or loaded in a subsequent step of the same job, use DSNAME=&&dsname and DISP=(,PASS). If it is to be link edited or loaded in another job, define a cataloged or kept data set.	Magnetic tape or direct access. UNIT is required. You can omit VOLUME and LABEL.	Required If on direct access.	You can specify BLKSIZE or and LRECL.	You can use DUMMY while testing this program	Optional. Same as SYSPUNCH but can be used as intermediate input to linkage editor or loader.
SYSIN		Usually in the input stream; use *.		You can specify BLKSIZE		Input to the COBOL F compiler (source statements).

FORTRAN E

Table 41. FORTRAN E - Input Deck

FORTRAN E			
//jobname	JOB	parameters	
//[name]	EXEC	PGM=IEJFAAA0[,PARM=parameters]	Required
//SYSUT1	DD	parameters	Required
//SYSUT2	DD	parameters	Required
//SYSPRINT	DD	parameters	Required
//SYSPUNCH	DD	parameters	Optional
//SYSLIN	DD	parameters	Optional
//SYSIN	DD	* or parameters	Required

Table 42. FORTRAN E - PARM Field of EXEC Statement

PARM=	Meaning
SIZE={nnnnnnn / nnnnK}	Specifies the maximum number of bytes available to the compiler. You can specify a number from 15360 to 999999, or 15K to 9999K. (K=1024 bytes.)
LINELNG=nnn	Specifies, as a number from 001 to 999, the maximum length of a record written under control of a FORMAT statement.
NAME=name	Specifies the name assigned to the compiled module. The name can have up to six alphameric characters, the first of which must be a letter.
{BCD / EBCDIC}	The source program is written in BCD. The source program is written in EBCDIC.
{SOURCE / NOSOURCE}	The source program is listed on the data set specified by the SYSPRINT DD statement. The source program is not listed.
{DECK / NODECK}	The object module is written on the device specified by the SYSPUNCH DD statement. The object module is not written.
{MAP / NOMAP}	A storage map is produced on the device specified by the SYSPRINT DD statement. A storage map is not produced.
{LOAD / NOLOAD}	The object module is written on the data set specified by the SYSLIN DD statement. The object module is not written.
{SPACE / PRFRM}	Specifies that 15360 bytes of main storage are used for compilations and the remainder of the space specified with SIZE is used for the dictionary, overflow table, and I/O buffers. Specifies that the first 19456 bytes specified in SIZE are used for compilations, and the rest is used for dictionary, overflow table and I/O buffers.
{ADJUST / NOADJUST}	The source program is to be scanned for meaningful blanks, embedded blanks and keywords used as variable names and put into a form acceptable to the compiler. The source program is not to be scanned.
Note: Underscore the default values selected by your installation during system generation.	

Table 43. FORTRAN E - DD Statements

ddname	Data Set Information	Location of the Data Set	Size of the Data Set	Data Attributes	Special Processing Options	Comments
SYSUT1	Temporary. You can omit DSNAME and DISP.	Magnetic tape or direct access. UNIT is required. You can omit VOLUME and LABEL.	Required if on direct access.	Omit.	Omit.	Work data sets.
SYSPRINT		System output device, printer or intermediate storage. SYSOUT=A is usually specified.		You can specify BLKSIZE if you use the PRFRM option.		Listings are produced on this data set.
SYSPUNCH		System output device, card punch or intermediate storage. SYSOUT=B usually specified.		You can specify BLKSIZE if you use the PRFRM option.	You can use DUMMY while testing the program.	Optional. Object module is produced on this data set.
SYSLIN	If the object module is to be link edited or loaded in a subsequent step of the same job, use DSNAME=&&dsname and DISP=(,PASS). If it is to be link edited or loaded in another job, define a cataloged or kept data set.	Magnetic tape or direct access. UNIT is required. You can omit VOLUME and LABEL.	Required if on direct access.	You can specify BLKSIZE if you use the PRFRM option.	You can use DUMMY while testing the program.	Optional. Same as SYSPUNCH but can be used as intermediate input to linkage editor or loader.
SYSIN		Usually in the input stream; use *.		You can specify BLKSIZE if you use the PRFRM option.		Input to the FORTRAN E compiler (source statements).

FORTRAN G

Table 44. FORTRAN G - Input Deck

FORTRAN G			
//jobname	JOB	parameters	
//[name]	EXEC	PGM=IEYFORT[,PARM=parameters]	Required
//SYSPRINT	DD	parameters	Required
//SYSPUNCH	DD	parameters	Optional
//SYSLIN	DD	parameters	Optional
//SYSIN	DD	* or parameters	Required

Table 45. FORTRAN G - PARM Field of EXEC Statement

PARM=	Meaning
NAME=name	Specifies the name assigned to the compiled module. The name can have up to six alphameric characters, the first of which must be a letter.
{BCD / EBCDIC}	The source program is written in BCD. The source program is written in EBCDIC.
{SOURCE / NOSOURCE}	The source program is listed on the data set specified by the SYSPRINT DD statement. The source program is not listed.
{DECK / NODECK}	The object module is written on the device specified by the SYSPUNCH DD statement. The object module is not written.
{MAP / NOMAP}	A table of names is written on the data set specified by the SYSPRINT DD statement. A table of names is not written.
{LOAD / NOLOAD}	The object module is written on the data set specified by the SYSLIN DD statement. The object module is not written.
{LIST / NOLIST}	The object module is listed on the device specified by the SYSPRINT DD statement. The object module is not listed.
LINECNT=nn	Specifies, as a number from 01 to 99, the number of lines to be printed on each page of the source listing.
{ID / NOID}	Internal statement numbers are to be generated for statements that call a subroutine or contain external function references. Internal statement numbers are not to be generated.
Note: Underscore the default values selected by your installation during system generation.	

Table 46. FORTRAN G - DD Statements

ddname	Data Set Information	Location of the Data Set	Size of the Data Set	Data Attributes	Special Processing Options	Comments
SYSPRINT		System output device, printer or intermediate storage. SYSOUT=A is usually specified.		Omit.		Listings are produced on this data set.
SYSPUNCH		System output device, card punch or intermediate storage. SYSOUT=B usually specified.		Omit.	You can use DUMMY while testing the program.	Optional. Object module is produced on this data set.
SYSLIN	If the object module is to be link edited or loaded in a subsequent step of the same job, use DSANME=&&dsname and DISP=(,PASS). If it is to be link edited or loaded in another job, define a cataloged or kept data set.	Magnetic tape or direct access. UNIT is required. You can omit VOLUME and LABEL.	Required if on direct access.	Omit.	You can use DUMMY while testing this program.	Optional. Same as SYSPUNCH, but can be used as intermediate input to linkage editor or loader. You can define a punched deck as in SYSPUNCH.
SYSIN		Usually in the input stream; use *.		Omit.		Input to the FORTRAN G compiler (source statements).

FORTRAN H

Table 47. FORTRAN H - Input Deck

FORTRAN H			
//jobname	JOB	parameters	
//[name]	EXEC	PGM=IEKAA00[,PARM=parameters]	Required
//SYSUT1	DD	parameters	Optional
//SYSUT2	DD	parameters	Optional
//SYSABEND	DD	parameters	Required
//SYSPRINT	DD	parameters	Required
//SYSPUNCH	DD	parameters	Optional
//SYSLIN	DD	parameters	Optional
//SYSIN	DD	* or parameters	Required

Table 48. FORTRAN H - PARM Field of EXEC Statement

PARM=	Meaning
NAME=name	Specifies the name assigned to the compiled module. The name can have up to six alphameric characters the first of which must be a letter.
BCD EBCDIC	The source program is written in BCD. The source program is written in EBCDIC.
SOURCE NOSOURCE	The source program is listed on the data set specified by the SYSPRINT DD statement. The source program is not listed.
DECK NODECK	The object module is written on the device specified by the SYSPUNCH DD statement.. The object module is not written.
MAP NOMAP	A table of names is written on the data set specified by the SYSPUNCH DD statement. A table of names is not written.
LOAD NOLOAD	The object module is written on the data set specified by the SYSLIN DD statement. The object module is not written.
LIST NOLIST	The object module is listed on the device specified by the SYSPRINT DD statement. The object module is not listed.
LINECNT=nn	Specifies as a number from 01 to 99, the number of lines to be printed on each page of the source listing.
ID NOID	Internal statement numbers are to be generated for statements that call a subroutine or contain external function references. Internal statement numbers are not to be generated.
OPT=0 OPT=1 OPT=2	The object module is not to be optimized. The object module receives full register assignment and basic otimization. The object module receives full register assignment and complete optimization.
EDIT NOEDIT	A structured source listing is written on the data set specified by the SYSPRINT DD statement. (OPT=2 and a SYSUT1 DD statement are required.) A structured source listing is not written.
XREF NOXREF	A cross-reference listing is written on the data set specified by the SYSPRINT DD statement. (A SYSUT2 DD statement is required.) A cross-reference listing is not written.
Note: Underscore the default values selected by your installation during system generation.	

Table 49. FORTRAN H - DD Statements

ddname	Data Set Information	Location of the Data Set	Size of the Data Set	Data Attributes	Special Processing Options	Comments
SYSUT1	Temporary. You can omit DSNAME and DISP.	Magnetic tape or direct access. UNIT is required. You can omit VOLUME and LABEL.	Required if on direct access.	You can specify BLKSIZE.	Omit.	Required only if you use EDIT option.
SYSUT2	Temporary. You can omit DSNAME and DISP.	Magnetic tape or direct access. UNIT is required. You can omit VOLUME and LABEL.	Required if on direct access.	You can specify BLKSIZE.	Omit.	Required only if you use XREF option.
SYSABEND or SYSUDUMP		System output device, printer or intermediate storage. SYSOUT=A is usually specified.				Optional. Abnormal termination dump is usually produced on this data set. Code as described for "Abnormal Termination Dump" in the section "Special DD Statements."
SYSPRINT		System output device, printer or intermediate storage. SYSOUT=A is usually specified.		You can specify BLKSIZE.		Listings are produced on this data set.
SYSPUNCH		System output device, card punch or intermediate storage. SYSOUT=B usually specified.		You can specify BLKSIZE.	You can use DUMMY while testing the program.	Optional. Object module is produced on this data set.
SYSLIN	If object module is to be link edited or loaded in a subsequent step of the same job, use DSNAME=&&dsname and DISP=(,PASS). If it is to be link edited or loaded in another job, define a cataloged or kept data set.	Magnetic tape or direct access. UNIT is required. You can omit VOLUME and LABEL.	Required if on direct access.	You can specify BLKSIZE.	You can use DUMMY while testing this program.	Optional. Same as SYSPUNCH, but can be used as intermediate input to define a punched deck as in SYSPUNCH.
SYSIN		Usually in the input stream; use *.		You can specify BLKSIZE.		Input to the FORTRAN H compiler (source statements).

PL/I F

Table 50. PL/I F - Input Deck

PL/I F			
//jobname	JOB	parameters	
//[name]	EXEC	PGM=IEMAA[,PARM=parameter]	Required
//SYSLIB	DD	parameters	Optional
//SYSUT1	DD	parameters	Optional
//SYSUT3	DD	parameters	Optional
//SYSPRINT	DD	parameters	Required
//SYSPUNCH	DD	parameters	Optional
//SYSLIN	DD	parameters	Optional
//SYSIN	DD	* or parameters	Required

Table 51. PL/I F - PARM Field of EXEC Statement (Part 1 of 2)

PARM=	Meaning
SIZE={nnnnnn / nnnK}	Specifies the maximum number of bytes available to the compiler. You can specify a number from 45056 to 999999 or 44K to 999K. (K=1024 bytes.)
LINECNT=nn	Specifies, as a number from 10 to 99, the number of lines to be printed on each page of the output listing.
{SOURCE / NOSOURCE}	The source program is to be listed on the device specified by the SYSPRINT DD statement. The source program is not to be listed.
{LIST / NOLIST}	The object module is to be listed on the device specified by the SYSPRINT DD statement. The object module is not to be listed.
{OPLIST / NOOPLIST}	The list of compiler options is to be listed on the device specified by the SYSPRINT DD statement. The list of compiler options is not to be listed.
{EXTREF / NOEXTREF}	The external symbol dictionary is listed on the device specified by the SYSPRINT DD statement. The external symbol dictionary is not listed.
{ATR / NOATR}	The attribute table is to be listed on the device specified by the SYSPRINT DD statement. The attribute table is not to be listed.
{XREF / NOXREF}	A cross-reference table is to be listed on the device specified by the SYSPRINT DD statement. A cross-reference table is not to be listed.
{DECK / NODECK}	The object module is written on the device specified by the SYSPUNCH DD statement. The object module is not written.
{LOAD / NOLOAD}	The object module is written on the data set specified by the SYSLIN DD statement. The object module is not written.
OBJNM=name	Specifies the name assigned to the object module.

Table 51. PL/I F - PARM Field of EXEC Statement (Part 2 of 2)

PARM=	Meaning
{OPT=00}	The object module is to be optimized unless it increases object time storage requirements.
{OPT=01}	The object module is to be optimized even if it increases object time storage requirements.
{BCD } {ECBDIC}	The source program is written in DCB. The source program is written in EBCDIC.
SORMGIN= (m,n,[,c])	Specifies the margin for scanning source statements. The following conditions must be met: 2≤m≤n≤100 c=b|0|-|+|1
{CHAR60}	The character set used to write the source program has 60 characters.
{CHAR48}	The character set used to write the source program has 48 characters.
{STMT }	Diagnostic messages produced during execution of object program are to contain source program statement numbers.
{NOSTMT}	Diagnostic messages are not to contain source program statement numbers.
{FLAGW}	Working messages, error messages and severe error messages are to be printed.
{FLAGE}	Only error messages and severe error messages are to be printed.
{FLAGS}	Only severe error messages are to be printed.
{MACRO } {NOMACRO}	Compile time processing is required. Compile time processing is not required.
{SOURCE2 }	The input to the compile-time processor is to be listed on the device specified by the SYSPRINT DD statement.
{NOSOURCE2}	The input to the compile-time processor is not to be listed.
{COMP }	Compilation is to proceed after compile-time processing ends.
{NOCOMP}	Compilation is not to proceed after compile-time processing.
{EXTDIC } {NOEXTDIC}	A large dictionary is to be used. A normal dictionary is to be used.
{NEST }	The depth of nesting is indicated in the source listing.
{NONEST}	The depth of nesting is not indicated.
{MACDCK }	The output of the compile-time processor is written on the device specified by the SYSPUNCH DD statement.
{NOMACDCK}	The output of the compile-time processor is not to be written.
{M91 } {NOM91}	The object program is to be executed on a Model 91. The object program is not to be executed on a Model 91.
Note: Underscore the default values selected by your installation during system generation.	

Table 52. PL/I F - DD Statements

ddname	Data Set Information	Location of the Data Set	Size of the Data Set	Data Attributes	Special Processing Options	Comments
SYSUT1	Temporary. You can omit DSNAME and DISP.	Direct access. UNIT is required. You can omit VOLUME and LABEL.	Required	You can specify OPTCD=W.	Omit.	Required if SIZE is less than 52K, or if source program does not fit in main storage. It is recommended that you always use this data set.
SYSUT3	Temporary. You can omit DSNAME and DISP.	Magnetic tape or direct access. UNIT is required. You can omit VOLUME and LABEL.	Required if on direct access.	You can specify BLKSIZE.	Omit.	Required for compile-time processing and for programs that use the 48-character set.
SYSLIB	Code DSNAME=SYS1.PL1LIB and DISP=SHR. You can also use user libraries with a different ddname.	Omit.	Omit.	Omit.	Omit.	Required for compile-time processing.
SYSPRINT		System output device, printer or intermediate storage. SYSOUT=A usually specified.		You can specify BLKSIZE.		Listings are produced on this data set.
SYSPUNCH		System output device, card punch or intermediate storage. SYSOUT=B usually specified.		You can specify BLKSIZE.	You can use DUMMY while testing the program.	Optional. Object module is produced on this data set.
SYSLIN	If object module is to be link edited or loaded in a subsequent step of the same job, use DSNAME=&&dsname and DISP=(,PASS). If it is to be link edited or loaded in another job, define a cataloged or kept data set.	Mangetic tape or direct access. UNIT is required. You can omit VOLUME and LABEL.	Required if on direct access.	You can specify BLKSIZE.	You can use DUMMY while testing the program.	Optional. Same as SYSPUNCH but can be used as immediate input to linkage editor or loader.
SYSIN		Usually in the input stream; use *.		You can specify BLKSIZE.		Input to the PL/I F compiler (source statements).

Linkage Editor

The linkage editor processes the compiled source program (object module) and prepares it for execution. After the object module is link edited it is called a load module; that is the load module is your program in executable form. Execution of the load module is performed in a subsequent job step (called a go step). The linkage editor can be used for compile-link edit-go, compile-link edit, link edit, and link edit-go jobs.

Table 53. Linkage Editor - Input Deck

LINKAGE EDITOR			
//[name]	EXEC	PGM=(see note)[,PARM=parameters]	Required
//SYSLIB	DD	parameters	Optional
//SYSUT1	DD	parameters	Required
//SYSPRINT	DD	parameters	Required
//SYSLMOD	DD	parameters	Required
//SYSLIN	DD	* or parameters	Required

Note: Specify the name of the linkage editor you want to use in the PGM parameter:

Name	Linkage Editor
IEWLE150	15K-design level E
IEWLE180	18K-design level E
IEWLF440	44K-design level F
IEWLF880	88K-design level F
IEWLF128	128K-design level F
IEWL, LINKEDIT	Largest design level available in your system

Table 54. Linkage Editor - PARM Field of EXEC Statement (Part 1 of 2)

PARM=	Meaning
LIST	All linkage editor control statements included in the step are listed on the device specified by the SYSPRINT DD statement.
MAP, XREF	A load module map is produced on the device specified by the SYSPRINT DD statement. A cross-reference table of the load module is produced on the device specified by the SYSPRINT DD statement. (XREF includes MAP.)
LET	The load module is marked executable even though a severity 2 error condition was found during processing.
XCAL	The load module is marked executable even though valid exclusive references between segments have been made.
NCAL	Library members are not called to resolve external references within the linkage editor input. The load module is marked executable even though unresolved external references have been recognized.

Table 54. Linkage Editor - PARM Field of EXEC Statement (Part 2 of 2)

PARM=	Meaning
SIZE=(value$_1$,value$_2$)	Value$_1$ specifies the maximum number of bytes of main storage to be used by the level F linkage editor. Value$_2$ specifies the maximum portion of value to be used as the load module buffer.
DCBS	Means that the SYSLMOD DD statement specifies the BLKSIZE subparameter in its DCB parameter. For level F only.
{OVLY}	The load module is to be in an overlay structure suitable only for block loading.
{SCTR}	The load module is to be in a format suitable for block or scatter loading.
{HIAR}	Control sections within the load module are marked for loading (block or scatter) into the hierarcnies specified with the HIARCHY control statement.
REFR	The load module is marked refreshable.
{RENT}	The load module is marked reenterable.
{REUS}	The load module is marked serially reusable.
NE	The load module has no external symbol dictionary (ESD) and it cannot be reprocessed by the linkage editor.
OL	The load module can be brought into main storage only by the LOAD macro instruction. (Only loadable.)
DC	Load modules processed by the level F linkage editors can be reprocessed by level E linkage editors. (Downward compatible.)
TEST	TESTRAN symbol tables within the input modules are placed in the load module. (Do not specify RENT or REUS when you specify TEST.)

Table 55. Linkage Editor - DD Statements (Part 1 of 2)

ddname	Data Set Information	Location of the Data Set	Size of the Data Set	Data Attributes	Special Processing Options	Comments
SYSLIB	Code DSNAME=dsname and DISP=SHR. Usually the language library for your compiler, that is SYS1.ALGLIB, SYS1.COBLIB, SYS1.FORTLIB, or SYS1.PL1LIB. Any user library can be defined. Alternatively the user library can be concatenated.	UNIT and VOLUME are required if library is kept.	Omit.	Omit.	Omit.	Call library. Partitioned data set.
SYSUT1	Temporary. You can omit DSNAME and DISP.	Direct access. UNIT is required. You can omit VOLUME and LABEL.	Required	Omit.	Omit.	Work data set.
SYSPRINT		System output device, printer or intermediate storage. SYSOUT=A usually specified.		You can specify BLKSIZE if you are using level F.		Listings are produced on this data set.
SYSLMOD	The partitioned data set can be temporary or non-temporary. If you do not include a NAME control statement specify DSNAME=dsname(member) or DSNAME=&&dsname(member). If you are creating the data set, code (or omit) NEW in DISP parameter. If the data set exists, code MOD in DISP parameter. If "go" step is in the same job, code DISP=(,PASS) or DISP=(MOD,PASS). If "go" step is not in the same job, keep or catalog the data set.	Direct access. • If you are creating the data set, code UNIT. You can omit VOLUME and LABEL. • If data set is cataloged, you can omit UNIT, VOLUME and LABEL. • If data set is kept, code UNIT and VOLUME. You can omit LABEL.	Required if you are creating the data set.			Load module(s) is produced on this partitioned data set.

Table 55. Linkage Editor - DD Statements (Part 2 of 2)

ddname	Data Set Information	Location of the Data Set	Size of the Data Set	Data Attributes	Special Processing Options	Comments
SYSLIN	If object module was created in a previous step of the same job, make a backward reference, i.e., DSNAME=*.stepname. ddname and DISP=OLD. If object module was created in another job, code DSNAME=dsname and DISP=OLD. If object module is a card deck, omit DSNAME and DISP. You can concatenate input data sets as long as they have the same RECFM, BLKSIZE, DEN, and TRTCH.	If object module was created in a previous step, you can omit UNIT, VOLUME and LABEL. If object module is kept, code UNIT and VOLUME. You can omit LABEL. If object module is card deck use *	Omit.	If object module is on unlabled tape, specify RECFM, BLKSIZE, DEN, and TRTCH.	Omit.	Input to the linkage editor (object module).

Loader

The loader processes the compiled source program (object module) for execution and passes control to it directly. Loader processing is performed in a load step, which is equivalent to the link edit-go steps. The loader can be used for compile-load, and load jobs.

Table 56. Loader - Input Deck

Loader			
//[name]	EXEC	PGM=LOADER[,PARM='loaderparm/programparm']	Required
//SYSLIB	DD	parameters	Optional
//SYSLOUT	DD	parameters	Optional
//SYSLIN	DD	* or parameters	Required
DD statements and data required for loaded program			
Note: The ddnames required by the loader can be changed during system generation.			

Table 57. Loader - PARM Field of EXEC Statement (Part 1 of 2)

PARM=	Meaning
{MAP / NOMAP}	A map of the loaded program is produced on the device specified by the SYSOUT DD statement. A map is not produced.
{RES / NORES}	An automatic search of the link pack area is made. If you specify RES, CALL is assumed. The link pack area is not to be searched.
{CALL / NOCALL}	An automatic search of the data set defined by the SYSLIB DD statement is to be made. The SYSLIB data set is not be searched. NOCALL can also be coded as NCAL. If you code NOCALL or NCAL, NORES is assumed.
{LET / NOLET}	The loader tries to execute the loaded program even though a severity 2 error condition is found. The loader does not try to execute the loaded program if a severity 2 error is found.
SIZE=size	Specifies the amount of main storage, in bytes, that can be used by the loader.
EP=name	Specifies the external name to be assigned as the entry point of the loaded program. For FORTRAN, ALGOL, and PL/I, you must specify MAIN, IHIFSAIN, and IHENTRY, respectively.
{PRINT / NOPRINT}	Diagnostic messages are produced on the device specified by the SYSOUT DD statement. Diagnostic messages are not produced.

Table 57. Loader - PARM Field of EXEC Statement (Part 2 of 2)

PARM=	Meaning
program parm	You may code PARM subparameters required by your program after any loader subparameters you use. The loaded program subparameters, if any, must be separated from the loader subparameters by a slash (/). If there are no loader options, the program subparameters must begin with a slash.
Note: Underscore the default options selected by your installation during system generation.	

Table 58. Loader - DD Statements

ddname	Data Set Information	Location of the Data Set	Size of the Data Set	Data Attributes	Special Processing Options	Comments
SYSLIB	Code DSNAME=dsname and DISP=SHR or DISP=OLD. Usually the language library for your compiler; that is SYS1.ALGLIB, SYS1.COBLIB, SYS1.FORTLIB, or SYS1.PL1LIB. Any user library can be defined. Alternatively, the user libraries can be concatenated.	UNIT and VOLUME are required if library is kept.	Omit.	Omit.	Omit.	Call library. Partitioned data set.
SYSLOUT		System output device, printer or intermediate storage. SYSOUT=A usually specified.				Listings are produced on this data set.
SYSLIN	If object module was created in a previous step of the same job, make a backward reference, i.e., DSNAME=*.stepname ddname and DISP=OLD. If object module was created in another job code DSNAME= dsname and DISP= OLD. If object module is a card deck, omit DSNAME and DISP. You can concatenate input data sets.	If object module was created in a previous step, you can omit UNIT, VOLUME and LABEL. If input is kept, code UNIT and VOLUME. You can omit LABEL. If object module is card deck, use *.	Omit.		Omit.	Input to the loader (object modules and load modules).
Include any DD statements required to execute your program.						

Examples

Examples are shown in this section for different types of job.

1. Compile (COBOL E)

   In the following example, a COBOL E source program is compiled. The source program is in the input stream (SYSIN). The object module is placed on a cataloged data set named COMP.COBOL. This data set resides on tape. Debug packets are not used, so the SYSUT4 DD statement is not needed. The user source program library ((SYSLIB) is required.

```
//EX1       JOB  79324,SMITH
//COMP      EXEC PGM=IEPCBL00,PARM=(DECK,MAPS,LIST)
//SYSLIB    DD   DSNAME=SYS1.COBLIB,DISP=SHR
//SYSUT1    DD   UNIT=2311,SPACE=(TRK,(50,10))
//SYSUT2    DD   UNIT=2400
//SYSUT3    DD   UNIT=2400
//SYSPRINT  DD   SYSOUT=A
//SYSPUNCH  DD DSNAME=COMP.COBOL,DISP=(,CATLG),UNIT=2400
//SYSIN     DD   *
      source statements
/*
//
```

2. Compile-link edit (Assembler F)

   In the following example a source program is assembled and link edited. The source program is in the input stream (SYSIN). The object module is placed on temporary data set on direct access (SYSGO) and punched (SYSPUNCH). Channel separation is requested for the work data set. Output listings are requested. A cataloged user module library is required for the link edit step. The load module produced by the linkage editor is placed on a new partitioned data set named MODULE. The name of the load module is PROG1.

```
//EX2       JOB  MSGLEVEL=(1,1)
//ASM       EXEC PGM=IEUASM,PARM=(DECK,LOAD,LIST,XREF)
//SYSLIB    DD   DSNAME=SYS1.MACLIB,DISP=SHR
//SYSUT1    DD   UNIT=2314,SPACE=(400,(400,50))
//SYSUT2    DD   UNIT=2314,SPACE=(400,(400,50))
//SYSUT3    DD   UNIT=2314,SPACE=(400,(400,50)),
//               SEP=(SYSUT1,SYSUT2)
//SYSPUNCH  DD   SYSOUT=B
//SYSGO     DD   DSNAME=&&TEMP,DISP=(PASS),
//               UNIT=2311,SPACE=(TRK,(50,20))
//SYSIN     DD   *
      source program
/*
//LE        EXEC PGM=IEWL,PARM=(LIST,XREF)
//SYSLIB    DD   DSN=PVTMOD,DISP=SHR
//SYSUT1    DD   UNIT=2311,SPACE=(1024,(50,20))
//SYSPRINT  DD   SYSOUT=A
//SYSLMOD   DD   DSNAME=MODULE(PROG1),DISP=(CATLG),
//               UNIT=2301,SPACE=(TRK,(50,,1))
//SYSLIN    DD   DSNAME=*.ASM.SYSGO,DISP=OLD
```

3. Compile-link edit-go (FORTRAN G)

In the following example a FORTRAN G source program is compiled, link edited, and executed. The source program is in the input stream (SYSIN). The object module is placed on a temporary tape data set (SYSLIN).

The FORTRAN library is required for the link edit step. The load module is placed on a temporary partitioned data set named &&PDS. The name of the load module is MAIN.

The program needs three data sets for execution

```
//EX3       JOB  ,JONES
//FG        EXEC PGM=IEYFORT,PARM=(BCD,NODEC,,LOAD)
//SYSPRINT  DD   SYSOUT=A
//SYSLIN    DD   DSNAME=&&OBJ,DISP=(,PASS),UNIT=2400
//SYSIN     DD   *
      source program
/*
//LE        EXEC PGM=IEWLF440,PARM=(MAP,DCBS),COND=(4,LT,FG)
//SYSLIB    DD   DSNAME=SYS1.FORTLIB,DISP=SHR
//SYSUT1    DD   DNIT=DISK,SPACE=(1024,(200,20))
//SYSPRINT  DD   SYSOUT=A
//SYSLMOD   DD   DSNAME=&&PDS(MAIN),DISP=(,PASS),
//                UNIT=(2311,SEP=SYSUT1),
//                SPACE=(TRK,(100,10,1),RLSE),DCB=BLKSIZE=3625
//SYSLIN    DD   DSNAME=*.FG.SYSLIN,DISP=OLD
//GO        EXEC PGM=*.LE.SYSLMOD
//FT10F001  DD   DSNAME=DATA,DISP=OLD,UNIT=2311,VOLUME=SER=FT9742
//FT11F001  DD   UNIT=2311,SEP=FT10F001),SPACE=(500,(20,10))
//FT12F001  DD   DSNAME=REPORT,DISP=(,KEEP),UNIT=2400,
//                LABEL=(,NSL)
```

4. Link Edit-go

In the following example, the COBOL program compiled in Example 1 is link edited and executed. The object program was placed in a cataloged data set named COMP.COBOL (SYSLIN). The load module is placed on a temporary partitioned data set named &&LM. The name of the load module is TTW. The program needs four data sets for execution. The data set defined by the INPUT DD statement is three concatenated data sets.

```
//EK4       JOB  7934,MSGLEVEL=(1,0)
//LK        EXEC PGM=LINKEDIT,PARM=(XREF,LIST,LET)
//SYSLIB    DD   DSNAME=SYS1.COBLIB,DISP=SHR
//SYSUT1    DD   UNIT=2311,VOLUME=SER=UTIL12,SPACE=(TRK,(10,10))
//SYSPRINT DD    SYSOUT=A
//SYSLMOD   DD   DSNAME=&&LM(TTW),DISP=(,PASS),UNIT=2314,
//                SPACE=(TRK,(15,,1))
//SYSLIN    DD   DSNAME=COMP.COBOL,DISP=(OLD,DELETE,KEEP)
//PROG      EXEC PGM=*.LK.SYSLMOD
//MASTER    DD   DSNAME=TRANS.DAY,DISP=MOD
//COPY      DD   DSNAME=BACKUP,DISP=MOD,UNIT=(2311,P),
//               VOLUME=SER=(FILE11,FILE12,FILE13),SEP=MASTER
//PRINT     DD   SYSOUT=A
//INPUT     DD   DSNAME=LISTC,DISP=(OLD,DELETE),UNIT=2400,
//               VOLUME=SER=1234B2,LABEL=(,SUL),SEP=(MASTER,COPY)
//          DD   DSNAME=FILE#2,DISP=OLD,AFF=INPUT
//          DD   *
      input data
/*
//
```

5. Go

In the following example, the program compiled and link edited in Example 2 is executed. The program resides on a private library named MODULE (JOBLIB). The name of the program is PROG1. PROG1 needs five data sets for execution. Note that PROG1 can accept PARM information because it was written in assembler language.

```
//EX5     JOB
//JOBLIB DD    DSNAME=7ODULE,DISP=OLD
//GOGO    EXEC PGM=PROG1,PARM=(SUMUP,FORMAT)
//NEWDS1 DD    DSNAME=GEORGE,DISP=(,KEEP),UNIT=(2314,3),
//               VOLUME=(PRIVATE,,,5),SPACE=(CYL,(150,100)),
//               DCB=(LRECL=600,BLKSIZE=6000)
//NEWDS2 DD    DSNAME=BRIEF.GEORGE,DISP=(,CATLG),UNIT=2301,
//               VOLUME=SER=230101,SPACE=(TRK,(20,5))
//WORK1   DD   UNIT=(2311,2),VOLUME=(,,,2),SPACE=(TRK,(100,100))
//WORK2   DD   UNIT=2400,LABEL=(,NL)
//OLDDS   DD   DDNAME=INPUT
//         DD  DSNAME=FILE,DISP=OLD
//         DD  DSNAME=LIST,DISP=OLD,UNIT=2400,VOLUME=SER=12345
//INPUT   DD   *
      input data
/*
//
```

6. Compile-load (ALGOL)

In the following example, an ALGOL source program is compiled and loaded. The source program is in the input stream (SYSIN). The object module is placed on a temporary data set on direct access (SYSLIN). Operations are suspended on the SYSPUNCH data set. SYS1.ALGLIB is needed for the load step. Three data sets required to execute your program are included in the load step.

```
//EX6       JOB  ,MARIA
//A         EXEC PGM=ALGOL,PARM=(LOAD,ISO)
//SYSUT1    DD   UNIT=2400
//SYSUT2    DD   UNIT=2400,SEP=SYSUT1
//SYSUT3    DD   UNIT=2311,SPACE=(2000,(20,5))
//SYSABEND  DD   SYSOUT=A
//SYSPRINT  DD   SYSOUT=A
//SYSPUNCH  DD   DUMMY
//SYSLIN    DD   DSNAME=&&TEMP,DISP=(,PASS),SPACE=(TRK,(10,5))
//SYSIN     DD   *
      source program
/*
//LINK EXEC      PGM=LOADER,PARM='MAP,EP=IHIFSAIN'
//SYSLIB    DD   DSNAME=SYS1.ALGLIB,DISP=SHR
//SYSLOUT   DD   SYSOUT=A
//SYSLIN    DD   DSNAME=*.A.SYSLIN,DISP=OLD
//SYSIN     DD   DSNAME=SERIES,DISP=OLD
//SYSPRINT  DD   SYSOUT=A
//ALGLDD04  DD   DSNAME=GRID,DISP=(,KEEP),UNIT=2314,
//                VOLUME=PRIVATE,SPACE=(TRK,(200,20),RLSE,ALX)
//
```

7. Compile-compile-load (PL/I F)

In the following example, two PL/I source programs are compiled and then combined and executed by the loader. The first source program resides in member SP10 of a partitioned data set named source. The second program resides in member SP11 of the same data set. The object modules are placed on separate temporary data sets. SYS1.PL1LIB is needed for all compile and load steps. The two data sets that contain the object modules are concatenated in the SYSLIN DD statement. Four data sets required to execute your program are included in the load step.

```
//EX7       JOB  ,JONES
//PL1#1     EXEC PGM=IEMAA,PARM=(CHAR48,LOAD,NODECK)
//SYSLIB    DD   DSNAME=SYS1.PL1LIB,DISP=SHR
//SYSUT1    DD   UNIT=DISK,SPACE=(1024,(100,100))
//SYSUT3    DD   UNIT=(DISK,SEP=SYSUT1),SPACE=(1024,(100,100))
//SYSPRINT  DD   SYSOUT=A
//SYSLIN    DD   DSNAME=&&OBJMOD,DISP=(,PASS),UNIT=2400
//SYSIN     DD   DSNAME=SOURCE(SP10),DISP=OLD,UNIT=2314,
//              VOLUME=SER=SORVOL
//PL1#2     EXEC PGM=IEMAA,PARM=(LOAD,NODECK)
//SYSLIB    DD   SYS1.PL1LIB,DISP=SHR
//SYSUT1    DD   DNIT=DISK,SPACE=(1024,(100,100))
//SYSPRINT  DD   SYSOUT=A
//SYSLIN    DD   DSNAME=&&OBJ,DISP=(,PASS),UNIT=2400
//SYSIN     DD   DSNAME=SOURCE(SP11),DISP=OLD,
//              VOLUME=REF=*.PL1#1.SYSIN
//LOAD      EXEC PGM=LOADER,PARM='MAP,EP=IHENTRY'
//SYSLIB    DD   DSNAME=SYS1.PL1LIB,DISP=SHR
//SYSLOUT   DD   SYSOUT=A
//SYSLIN    DD   DSNAME=*.PL1#1.SYSLIN,DISP=OLD
//              DD   DSNAME=*.PL1#2.SYSLIN,DISP=OLD
//OUTPUT    DD   UNIT=1403,UCS=QN,DCB=(PRTSP=2,RECFM=F)
//TABLE     DD   DSNAME=TABLE.PARMS,DISP=OLD
//FILE      DD   DSNAME=MASTER,DISP=MOD
//INPUT     DD   *
      input data
/*
//
```

8. Load

In the following example, the COBOL program compiled in Example 1 is loaded. The object program was placed in a cataloged data set named COMP.COBOL. The program needs four data sets for execution.

Compare this example to Example 4. In both examples, the same program is prepared for execution and executed. These functions are performed in a load step in this example, and in link edit-go steps in Example 4. Note how the data sets defined with the four DD statements required for execution are different from those defined in Example 4. However, the four ddnames must be the same each time the program is executed.

```
//EX8       JOB  2476,MSGLEVEL=(1,1)
//LOAD      EXEC PGM=LOADER,PARM=(MAP,LET,CALL)
//SYSLIB    DD   DSNAME=SYS1.COBLIB,DISP=SHR
//SYSLOUT   DD   SYSOUT=A
//SYSLIN    DD   DSNAME=COMP.COBOL,DISP=OLD
//MASTER    DD   DSNAME=ORDERS.WEEK,DISP=MOD
//COPY      DD   DSNAME=ARCHIVE,DISP=MOD,SEP=MASTER
//PRINT     DD   SYSOUT=A
//INPUT     DD   *
      input data
/*
//
```

9. The following example shows the JCL statements required for the execution of a four-step job. This job accounts for received inventory items and outstanding purchase orders.

The first step receives daily purchase receipts and purchase orders in card form (CARDIN) and transfers them to a passed temporary data set on tape (TRANS). This step also checks the input cards for errors. If an error is found in a card, that card is punched out (ERROR). Only the cards without errors are copied to tape (TRANS). The contents of the tape are listed (PRINT).

The second step uses the IBM sort/merge program to sort the input transactions according to inventory part number. The input transactions are in temporary tape data set passed from STEP1 (SORTIN) the sorted transactions are produced on another temporary tape data set (SORTOUT).

The third step updates the product inventory. The new transactions are in the temporary data set passed from STEP2 (INPUT). They are compared against the latest inventory (OLDMASTR) and a new inventory is produced (NEWMASTR). The inventories are generations of a generation data group. Two addiiional data sets are produced. The first one is a report of all erroneous purchases and purchase orders (ERRFILE). The second one is a management report of the new inventory (REPORT).

The fourth step prints the error report and the management report produced in STEP3.

```
//DAILY    JOB  7591A2B,ACCTDEPT,MSGLEVEL=(1,1)
//JOBLIB   DD   DSNAME=ACCTNG.INVTRY,DISP=SHR
//STEP1    EXEC PGM=CRDTP
//TRANS    DD   DSNAME=&&SORTIN,DISP=(,PASS),UNIT=2400-3
//ERROR    DD   SYSOUT=B
//PRINT    DD   SYSOUT=A
//CARDIN   DD   *
      input cards
/*
//STEP2    EXEC PGM=SORT,REGION=26K
//SYSOUT   DD   SYSOUT=A
//SORTLIB  DD   DSNAME=SYS1.SORTLIB,DISP=SHR
//SORTWK01 DD   UNIT=2314,SPACE=(CYL,5,,CONTIG)
//SORTWK02 DD   UNIT=(2314,SEP=SORTWK01),SPACE=(CYL,5,,CONTIG)
//SORTWK03 DD   UNIT=(2314,SEP=(SORTWK01,SORTWK02)),
//              SPACE=(CYL,5,,CONTIG)
//SORTWK04 DD   UNIT=(2314,SEP=(SORTWK01,SORTWK02,SORTWK03)),
//              SPACE=(CYL,5,,CONTIG)
//SORTWK05 DD   UNIT=(2314,SEP=(SORTWK01,SORTWK02,SORTWK03,
//              SORTWK04)),SPACE=(CYL,5,,CONTIG)
//SORTWK06 DD   UNIT=(2314,SEP=(SORTWK01,SORTWK02,SORTWK03,
//              SORTWK04,SORTWK05)),
//              SPACE=(CYL,5,,CONTIG)
//SORTIN   DD   DSNAME=&SORTIN,DISP=(OLD,DELETE),
//              DCB=(RECFM=FB,LRECL=74,BLKSIZE=1850)
//SORTOUT  DD   DSNAME=&SORTOUT,UNIT=2400-3,DISP=(NEW,PASS),
//              DCB=(RECFM=FB,LRECL=74,BLKSIZE=1850)
//SYSIN    DD   *
      sort control statements
/*
//STEP3    EXEC PGM=UPDT
//INPUT    DD   DSNAME=*.STEP.SORTOUT,DISP=OLD
//OLDMASTR DD   DSNAME=INVEN.PARTS(0),DISP=OLD
//NEWMASTR DD   DSNAME=INVEN.PARTS(+1),DISP(,CATLG),UNIT=2400-3,
//              VOLUME=SER=ACCT25
//ERRFILE  DD   DSNAME=&&ERRORS,DISP=(,PASS),UNIT=2311,
//              SPACE=(CYL,(5,1))
//REPORT   DD   DSNAME=&&MGT,DISP=(,PASS),UNIT=2311,
//              SPACE=(CYL,(10,5))
//STEP4    EXEC PGM=PRINT
//ERR      DD   DSNAME=*.STEP3.ERRFILE,DISP=OLD
//REP      DD   DSNAME=*.STEP3.REPORT,DISP=OLD
//PRINT    DD   SYSOUT=D
```

10. The following example shows how orders are processed in a book club. The book club maintains its inventory of books on an indexed sequential data set named CLUB.CATALOG. The list of club members is maintained on another indexed sequential data set named CLUB.MEMBERS. Both data sets are kept in unloaded form on tape. The member's orders are on punched cards. Each card has the member's account number and the catalog number of the book selected. The purpose of the job is to:

- Produce an order form for the stock room requesting the necessary number of books of each kind to be mailed out.
- Print order forms for depleted books.
- Update CLUB.CATALOG.
- Punch a card that is both a mailing label and invoice for each club member who ordered a book.
- Update CLUB.MEMBERS.

The first step uses the IEBISAM utility program to load the CLUB.CATALOG data set from tape (SYSUT1) to 2314 volumes (SYSUT2). In its unloaded form, CLUB.CATALOG resides in the latest member of a generation data group.

The second step performs the same function for CLUB.MEMBERS.

The third step transfers the customer orders from card (INPUT) to a passed temporary data set on tape (TAPE). The cards are also listed (PRINT).

The fourth step uses the IBM sort/merge program to sort the input orders by catalog number of the book requested. The input orders are in the temporary tape data set passed from STEP3 (SORTIN). The sorted orders are produced on another temporary tape data set (SORTOUT).

The fifth step compares the orderd (INBOOKS) to the CLUB.CATALOG data set (MASTER). A listing is produced stating the number of books of each class needed and their location in the stockroom (SOLD). If any books are depleted, or if there are less than 20 books left, an order form is printed for that book (ORDERS). One work data set is needed (WORK). The CLUB.CATALOG data set is updated in place.

The sixth step uses the IBM sort/merge program to sort the input orders by account number. The input orders are in the temporary data set passed from STEP3 (SORTIN). The sorted orders are produced on another temporary data set (SORTOUT).

The seventh step compares the orders (INACCT) to the CLUB.MEMBERS data set (MASTER). A card is punched for each order that is both a mailing label and an invoice (LABELS). The cards also state the name of the book ordered so the stock clerks can attach them to the proper package. A management report of account activity is prepared (REPORT). This management report is stored on an existing partitioned data set and is listed at a later time when all monthly reports are ready. The CLUB.MEMBERS data set is updated in place. Two work data sets are needed (WORK1 and WORK2).

The eighth step uses the IEBISAM utility program to unload the CLUB.CATALOG data set from 2314 volumes (SYSUT1) to a tape volume (SYSUT2). The unloaded data set becomes a member of a generation data group.

The ninth step performs the same function for CLUB.MEMBERS.

Note: Indexed sequential data sets are described in Appendix A.

```
//ORDERS   JOB  DECEMBER69,'SIMPLE BOOK CLUB',MSGLEVEL=(1,1)
//JOBLIB   DD   DSNAME=CLUB.PROGLIB,DISP=SHR
//STEP1    EXEC PGM=IEBISAM,PARM=LOAD
//SYSUT1   DD   DSNAME=CLUB.CATALOG(0),DISP=OLD
//SYSUT2   DD   DSNAME=CLUB.CATALOG(INDEX),DISP=(,KEEP),,
//              UNIT=2314,VOLUME=SER=222222,SPACE=(CYL,1),
//              DCB=(DSORG=IS,OPTCD=IR,KEYLEN=16,RKP=1)
//         DD   DSNAME=CLUB.CATALOG(PRIME),DISP=(,KEEP),
//              UNIT=2314,2)VOLUME=SER=(111111,231403),
              SPACE=(CYL,75),DCB=*.SYSUT2
//         DD   DSNAME=CLUB.CATALOG,(OVFLOW),DISP=(,KEEP),
//              UNIT=2314,VOLUME=SER=222222,SPACE=(CYL,5),
//              DCB=*SYSUT2
//SYSPRINT DD   SYSOUT=A
//STEP2    EXEC PGM=IEBISAM,PARM=LOAD
//SYSUT1   DD   DSNAME=CLUB.MEMBERS(0),DISP=OLD
//SYSUT2   DD   DSNAME=CLUB.MEMBERS(INDEX),DISP=(,KEEP),
//              UNIT=2314,VOLUME=SER=111111,SPACE=(CYL,1),
//              DCB=(DSORG=IS,OPTCD=YI,KEYLEN=4,RKP=4)
//         DD   DSNAME=CLUB.MEMBERS(PRIME),DISP=(,KEEP),
//              UNIT=2314,VOLUME=SER=222222,SPACE=(CYL,(15)),
//              DCB=*.SYSUT2
//         DD   DSNAME=CLUB.MEMBERS(OVFLOW),DISP=(,KEEP),
//              UNIT=2314,VOLUME=SER=111111,SPACE=(CYL,(1)),
//              DCB=*.SYSUT2
//SYSPRINT DD   SYSOUT=A
//STEP3    EXEC PGM=CTOT
//TAPE     DD   DSNAME=&&SORTIN,DISP=(,PASS),UNIT=2400
//PRINT    DD   UNIT=1443
//INPUT    DD   *
       input cards
/*
//STEP4    EXEC PGM=SORT
//SYSOUT   DD   SYSOUT=A
//SORTLIB  DD   DSNAME=SYS1.SORTLIB,DISP=SHR
//SORTWK01 DD   UNIT=2314,SPACE=(CYL,20,,CONTIG)
//SORTWK02 DD   UNIT=(2314,SEP=SORTWK01),
              SPACE=(CYL,20,,CONTIG)
//SORTWK03 DD   UNIT=(2314,SEP=(SORTWK01,SORTWK02)),
//              SPACE=(CYL,20,,CONTIG)
//SORTWK04 DD   UNIT=(2314,SEP=(SORTWK01,SORTWK02,SORTWK03)),
//              SPACE=(CYL,20,,CONTIG)
//SORTWK05 DD   UNIT=(2314,SEP=(SORTWK01,SORTWK02,SORTWK03,
//              SORTWK04)),SPACE=(CYL,20,,CONTIG)
//06       DD   UNIT=(2314,SEP=(SORTWK01,SORTWK02,SORTWK03,
//              SORTWK04,SORTWK05)),
//              SPACE=(CYL,20,,CONTIG)
//SORTIN   DD   DSNAME=&&SORTIN,UNIT=2400-3,DISP=(OLD,PASS),
//              DCB=(RECFM=FB,LRECL=114,BLKSIZE=114)),
//SORTOUT  DD   DSNAME=&&BOOKS,UNIT=2314,DISP=(,PASS,DELETE),
//              SPACE=(CYL,3),RLSE),VOLUME=SER=111111,
//              DCB=(RECFM=FB,LRECL=114,BLKSIZE=114)
//SYSIN    DD   *
       sort control statements
/*
```

```
//STEP5     EXEC PGM=STOCK
//MASTER    DD   DSNAME=CLUB.CATALOG,DISP=OLD,UNIT=(2314,3),
//               VOLUME=SER=(222222,111111,231403),
//               DCB=DSORG=IS
//INBOOKS   DD   DSNAME=*.STEP4.SORTOUT,DISP=OLD
//SOLD      DD   UNIT=1443
//ORDERS    DD   UNIT=1403,UCS=(PCAN,,VERIFY)
//WORK      DD   UNIT=2311,SPACE=(TRK,(20,5))
//STEP6     EXEC PGM=SORT
//SYSOUT    DD   SYSOUT=A
//SORTLIB   DD   DSNAME=SYS1.SORTLIB,DISP=SHR
//SORTWK01  DD   UNIT=2314,SPACE=(CYL,20,,CONTIG)
//SORTWK02  DD   UNIT=(2314,SEP=SORTWK01),
//               SPACE=(CYL,20,,CONTIG)
//SORTWK03  DD   UNIT=(2314,SEP=SORTWK01,SORTWK02)),
//               SPACE=(CYL,20,,CONTIG)
//SORTWK04  DD   UNIT=(2314,SEP=(SORTWK01,SORTWK02,SORTWK03)),
//               SPACE=(CYL,20,,CONTIG)
//SORTWK05  DD   UNIT=(2314,SEP=(SORTWK01,SORTWK02,SORTWK03,
//               SORTWK04)),SPACE=(CYL,(20,,CONTIG)
//SORTWK06  DD   UNIT=(2314,SEP=(SORTWK01,SORTWK02,SORTWK03,
//               SORTWK04,SORTWK05))
//               SPACE=(CYL,20,,CONTIG)
//SORTIN    DD   DSNAME=&&SORTIN,UNIT=2400-3,DISP=(OLD,DELETE),
//               DCB=(RECFM=FB,LRECL=114,BLKSIZE=114))
//SORTOUT   DD   DSNAME=&&ACCTS,UNIT=2314,DISP=(,PASS,DELETE),
//               SPACE=(CYL,(3,1),RLSE),VOLUME=SER=111111,
//               DCB=(RECFM=FB,LRECL=114,BLKSIZE=1140)
//SYSIN     DD   *
      sort control statements
/*
//STEP7     EXEC PGM=BILLS
//MASTER    DD   DSNAME=CLUB.MEMBERS,DISP=OLD,UNIT=(2314,2)
//               VOLUME=SER=(111111,222222),DCB=DSORG=IS
//LABELS    DD   SYSOUT=B
//REPORT    DD   DSNAME=REPORT.DEC69(ORDERS),DISP=MOD
//WORK1     DD   UNIT=2311,SPACE=(CYL,(10,1))
//WORK2     DD   UNIT=(2311,SEP=WORK1),SPACE=(CYL,(10,2))
//STEP8     EXEC PGM=IEBISAM,PARM=UNLOAD
//SYSPRINT  DD   SYSOUT=A
//SYSUT1    DD   DSNAME=CLUB.CATALOG,DISP=(OLD,DELETE,KEEP),
//               UNIT=(2314,3),VOLUME=SER=(222222,111111,
//               231403),DCB=DSORG=IS
//SYSUT2    DD   DSNAME=CLUB.CATALOG(+1),DISP=(,CATLG),
//               UNIT=2400,VOLUME=SER=T79321
//STEP9     EXEC PGM=IEBISAM,PARM=UNLOAD
//SYSPRINT  DD   SYSOUT=A
//SYSUT1    DD   DSNAME=CLUB.MEMBERS,DISP=(OLD,DELETE,KEEP),
//               UNIT=(2314,2),VOLUME=SER=(111111,222222),
//               DCB=DSORG=IS
//SYSUT2    DD   DSNAME=CLUB.MEMBERS(+1),DISP=(,CATLG),
//               UNIT=2400,VOLUME=SER=T89425
//
```

Part III: Cataloged and In-stream Procedures

A cataloged procedure is a set of job control statements that has been assigned a name and placed in a partitioned data set known as the procedure library. (The IBM-supplied procedure library is named SYS1.PROCLIB; at your installation, there may be additional procedure libraries which would have different names.) An in-stream procedure is a set of job control statements in the form of cards that have been placed in the input stream of a card reader. An in-stream procedure can be executed any number of times during the job in which it appears. Both cataloged and in-stream procedures statements that has been placed in the procedure library can consist of one or more steps; each step is called a procedure step. Each procedure step consists of an EXEC statement and DD statements. The EXEC statement identifies to the system what program is to be executed. The DD statements define the data sets to be used by the program.

You can use a cataloged procedure by coding the procedure name on an EXEC statement. You can use an in-stream procedure by coding the procedure name that is on the PROC statement on an EXEC statement. With both cataloged and in-stream procedures, you can follow this EXEC statement with DD statements that modify the procedure for the duration of the job step.

Part III consists of two chapters. The first chapter, "Using Cataloged and In-stream Procedures" describes how to call a procedure, how to assign values to symbolic parameters, how to override parameters on the EXEC and DD statement, and how to add DD statements to a procedure. The second chapter, "Writing Procedures: Cataloged and In-stream" the makeup of a procedure, how to use symbolic parameters, how to place a set of job control statements in the procedure library, and how to modify a procedure.

Note: This section is essentially the same as Appendix A of the publication IBM System/360 Operating System: Job Control Language Reference. It is presented in both publications for the sake of completeness.

Using Cataloged and In-stream Procedures

How to Call a Cataloged Procedure

To use a cataloged procedure, submit a JOB statement followed by an EXEC statement. On the EXEC statement you identify the cataloged procedure in one of two ways:

1. Code, as the first operand, the name assigned to the procedure; or
2. Code PROC= followed by the name assigned to the procedure as the first operand.

When you call a procedure, the system finds the control statements in the prodecure library and then executes the programs identified on the EXEC statements in the procedure.

Besides identifying the procedure on the EXEC statement, you can assign values to symbolic parameters and override parameters that are coded on the EXEC statements contained in the procedure. You follow the EXEC statement with DD statements when you want to override DD statements in the procedure or add DD statements to the procedure.

When a cataloged procedure is written as part of the system output listing (i.e., MSGLEVEL=(1,0), MSGLEVEL=(1,1), or MSGLEVEL=1 is coded on the JOB statement), the procedure statements can be easily identified. An XX appears in columns 1 and 2 of a procedure statement that you did not override; X/ appears in in columns 1 and 2 of a procedure statement that you did override; XX* appears in columns 1 through 3 of a procedure statement, other than a comment statement, that the system considered to contain only comments; and *** appears in columns 1 through 3 of a comment statement. In addition, if the procedure contains symbolic parameters, the output listing will show the symbolic parameters and the values assigned to them.

How to Call an In-stream Procedure

To use an in-stream procedure, submit a JOB statement followed by the in-stream procedure to be used. The in-stream procedure can appear immediately following the JOB statement, the JOBLIB DD statement, or the SYSCHK DD statement. The in-stream procedure cannot appear before the JOB statement or after the EXEC statement that calls it. An in-stream procedure can appear after a SYSIN DD * statement; however, this is not advisable because of the allocation of a data set when SYSIN DD * is specified.

To call the procedure, you identify the in-stream procedure on an EXEC statement in one of two ways:

1. Code, as the first operand, the name on the PROC statement of the procedure; or

2. Code PROC= followed by the name on the PROC statement of the procedure.

When you call an in-stream procedure, the system finds the control statements that have been written on a direct access device and then executes the programs identified on the EXEC statements of the procedure.

Besides identifying the procedure on the EXEC statement, you can assign values to symbolic parameters and override parameters that are coded on the EXEC statements contained in the procedure. You follow the EXEC statement with DD statements when you want to override DD statements in the procedure or add DD statements to the procedure.

When an in-stream procedure is written as part of the system output listing (i.e., MSGLEVEL=(1,0), MSGLEVEL(1,1), MSGLEVEL=1 or MSGLEVEL=2 is coded on the JOB statement), the procedure statements can be easily identified. An ++ appears in columns 1 and 2 of a procedure statement that you did not override; +// appears in columns 1 and 2 of a procedure statement that you did override; +++* appears in columns 1 through 3 of a procedure statement, other than a comment statement, that the system considered to contain only comments; and *** appears in columns 1 through 3 of a comment statement. In addition, if the procedure contains symbolic parameters and you assign values to these on the EXEC statement that calls the procedure, the output listing will show the symbolic parameters and the values assigned to them.

Assigning Values to Symbolic Parameters

The cataloged procedure you call may contain symbolic parameters. A symbolic parameter is characterized by a name preceded by an ampersand (&) and appears in the operand field of a cataloged procedure statement or a DD statement used to override a DD statement in the procedure. A symbolic parameter stands as a symbol for a parameter, a subparameter, or a value. Symbolic parameters are used so that the procedure can be modified easily when it is called by a job step.

The following are examples of symbolic parameters:

```
//STEP1 EXEC PGM=COB,PARM='P1,&P2,P3'
//DD1 DD DSNAME=FIX,UNIT=&DEVICE,SPACE=(CYL,(&SPACE,10))
//DD2 DD DSNAME=CHAG,UNIT=2400,DCB=BLKSIZE=&LENGTH
```

Symbolic parameters must either be assigned values or nullified before the procedure is executed. There are two ways that a symbolic parameter can be assigned a value:

1. You assign a value to the symbolic parameter on the EXEC statement that calls the procedure.

2. The PROC statement, which can appear as the first statement in a cataloged procedure and must appear as the first statement in an in-stream procedure, assigns a default value to the symbolic parameter.

Any default value assigned to a symbolic parameter on the PROC statement is overridden when you assign a value to the same symbolic parameter on the EXEC statement that calls the procedure.

If the cataloged procedures contain symbolic parameters, the installation should provide you with a list of the symbolic parameters used, what meaning is associated with each symbolic parameter, and what default value has been assigned to each of the symbolic parameters on the PROC statement. (The PROC statement is optional for cataloged procedures; therefore, there may be no default values assigned to the symbolic parameters used in a cataloged procedure.) You need this information to determine what the symbolic parameter represents and to decide whether to use the default value or to assign a value to the symbolic parameter on the EXEC statement that calls the procedure.

To assign a value to a symbolic parameter, you code on the EXEC statement that calls the procedure:

```
symbolic parameter=value
```

Omit the ampersand that precedes the symbolic parameter. For example, if the symbolic parameter &NUMBER appears on a DD statement in the procedure, code

```
NUMBER=value
```

on the EXEC statement that calls the procedure. Any value you assign to a symbolic parameter is in effect only during the current execution of the procedure.

There are some rules you should keep in mind as you assign values to symbolic parameters:

1. The value you assign can be any length, but it cannot be continued onto another statement.

2. If the value contains special characters, enclose the value in apostrophes (the enclosing apostrophes are not considered part of the value). If the special characters include apostrophes, each must be shown as two consercutive apostrophes.

3. If, on the EXEC statement, you assign more than one value to a symbolic parameter, the first value encountered is used.

4. If the symbolic parameter is concatenated with some other information (e.g., &JOBNO.321), this information and the value you assign to the symbolic parameter cannot exceed a combined total of 120 characters.

Nullifying a Symbolic Parameter

Besides assigning values to symbolic parameters, you can nullify a symbolic parameter, i.e., tell the system to ignore the symbolic parameter.

To nullify a symbolic parameter, code on the EXEC statement that calls the procedure:

```
symbolic parameter=
```

Omit the ampersand that precedes the symbolic parameter and do not follow the equal sign with a value.

For example, if a DD statement in a procedure named TIMES is

```
//DD8 DD UNIT=1403,UCS=&UCSINFO
```

and you want to nullify the symbolic parameter &UCSINFO, you would code

```
//CALL EXEC TIMES,UCSINFO=
```

Example of Assigning Values to Symbolic Parameters

1. The following are the first four statements of a cataloged procedure named ASSEMBLE that contains symbolic parameters. The PROC statement assigns a default to the symbolic parameter &OBJECT and nullifies the symbolic parameter &LIST. Notice that the symbolic parameter &DEPT is not assigned a value on the PROC statement; therefore, the job step that calls this procedure must assign a value to &DEPT.

```
//DEF PROC OBJECT=NODECK,LIST=
//ASN EXEC PGM=IEUASM,PARM=(LINECNT=50,&LIST.LIST,&OBJECT)
//SYSLIB DD DSNAME=SYS1.MACLIB,DISP=OLD
//       DD DSNAME=LIBRARY.&DEPT.MACS,DISP=OLD
```

When you call this procedure, you can assign values to the symbolic parameters by coding:

```
//STEP3 EXEC ASSEMBLE,DEPT=D82,OBJECT=DECK
```

The value assigned to &OBJECT in this EXEC statement overrides the value assigned to &OBJECT in the PROC statement. Since no value is assigned to &LIST in this EXEC statement, LIST is nullified -- because that is the default specified in the PROC statement.

While the procedure is being executed, the first four statements of this procedure would appear as shown below.

```
//DEF PROC OBJECT=NODECK,LIST=
//ASM EXEC PGM=IEUASM,PARM=(LINECNT=50,LIST,DECK)
//SYSLIB DD DSNAME=SYS1.MACLIB,DISP=OLD
//       DD DSNAME=LIBRARY.D82MACS,DISP=OLD
```

The above example applies to in-stream procedures as well as cataloged procedures. However, you must refer to the name on the PROC statement of the in-stream procedure when calling the procedure.

2. The following is an in-stream procedure that contains symbolic parameters. The PROC statement marks the beginning of the in-stream procedure and in this example assigns defaults to symbolic parameters &D,&U,&V, and &S. The procedure is named INSTREAM.

```
//INSTREAM PROC D='(NEW,CATLG)',U=2311,V='SER=66655',S='(TRK,(1,1,1))
//IN1      EXEC PGM=IEWL,PARM='XREF,LIST,NCAL'
//SYSPRINT DD   SYSOUT=A
//SYSUT1   DD   DSNAME=UTC,DISP=OLD,UNIT=2311,VOLUME=SER=66651
//SYSLIN   DD   DSNAME=UTE,DISP=OLD,UNIT=2311,VOLUME=SER=66652
//SYSLMOD  DD   DSNAME=&&LOAD,DISP=&D,UNIT=&U,VOLUME=&V,SPACE=&S
//         PEND
```

When you call this procedure, you must code the name on the PROC statement on the EXEC statement. You can assign values to the symbolic parameters by coding:

```
//CALL     EXEC INSTREAM,D='(NEW,PASS)',V='SER=66653'
```

The values assigned to &D and &V in this EXEC statement override the values assigned to these symbolic parameters in the PROC statement.

Since no value is assigned to &U or &S, the defaults specified on the PROC statement are used when the procedure is executed.

While the procedure is being executed, it would appear as shown below.

```
//INSTREAM PROC D='(NEW,CATLG)',U=2311,V='SER=66655',S='(TRK,(1,1,1))
//IN1      EXEC PGM=IEWL,PARM='XREF,LIST,NCAL'
//SYSPRINT DD   SYSOUT=A
//SYSUT1   DD   DSNAME=UTC,DISP=OLD,UNIT=2311,VOLUME=SER=66651
//SYSLIN   DD   DSNAME=UTE,DISP=OLD,UNIT=2311,VOLUME=SER=66652
//SYSLMOD  DD   DSNAME=&&LOAD,DISP=(NEW,PASS),UNIT=2311,
//              VOLUME=SER=66653,SPACE=(TRK,(1,1,1))
```

The PEND statement is printed but is not executed.

Overriding, Adding, and Nullifying Parameters on an EXEC Statement

You can override, add, or nullify parameters coded on EXEC statements contained in a cataloged procedure. You make these changes on the EXEC statement that calls the procedure. You cannot, however, change the PGM parameter. The changes you make are in effect during the current execution of the procedure.

Overriding EXEC Statement Parameters

To override an EXEC statement parameter in a procedure, identifying on the EXEC statement that calls the procedure the parameter you are overriding, the name of the EXEC statement on which the parameter appears, and the change to be made. The format required to override a parameter is:

```
parameter.procstepname=change
```

For example, if one of the EXEC statements in the procedure named FILL is:

```
//STEP3 EXEC PGM=DEF,REGION=100K
```

and you want to change REGION=100K to REGION=80K, you would code:

```
//CALL EXEC FILL,REGION.STEP3=80K
```

You can change more than one EXEC statement parameter in the procedure. For example, if one of the EXEC statements in the procedure name JKW is:

```
//STEP2 EXEC PGM=OUT,TIME=(2,30),REGION=120K
```

and you want to change TIME=(2,30) to TIME=4 and REGION=120K to REGION=200K, you would code:

```
//STEP3 EXEC JKW,TIME.STEP2=4,REGION.STEP2=200K
```

If you want to change different parameters that appear on different EXEC statements in the procedure, you must code all overriding parameters for one procedure step before those for the next step. For example, if the first three EXEC statements in a procedure named DART are:

```
//STEP1 EXEC PGM=JCTSB,PARM='*14863',REGION=100K
//STEP2 EXEC PGM=JCTRC,REGION=80K
//STEP3 EXEC PGM=JCTQD,COND=(8,LT),TIME=3
```

and you want to override the PARM and REGION parameters on the first EXEC statement, the REGION parameter on the second EXEC statement, and the TIME parameter on the third EXEC statement, the EXEC statement that calls the procedure would appear as:

```
//STEPC EXEC DART,PARM.STEP1='*86348',
//           REGION.STEP1=120K,REGION.STEP2=100K,
//           TIME.STEP3=(4,30)
```

You can code an EXEC statement parameter and omit the term "procstepname." When you do this, the procedure is modified as follows:

- If the PARM parameter is coded, it applies only to the first procedure step. If a PARM parameter appears in a later EXEC statement, it is nullified.
- If the TIME parameter is coded, it applies to the total procedure. If the TIME parameter appears on any of the EXEC statements in the procedure, it is nullified.
- If any other parameter is coded, it applies to every step in the procedure. If the parameter appears on an EXEC statement, it is nullified; if the parameter does not appear on an EXEC statement, it is added.

For example, assume the EXEC statements in a procedure named RYIN are:

```
//STEP1 EXEC PGM=SECT,PARM=140947,REGION=100K
//STEP2 EXEC PGM=PARA,PARM=105600,COND=EVEN
//STEP3 EXEC PGM=SENT,PARM=L1644,REGION=80K
```

If you want to override the PARM parameter in the first procedure step, nullify all other PARM parameters in the procedure, and have all procedure steps assigned the same region size, the EXEC statement that calls the procedure would appear as:

```
//SPAA EXEC RYINPARM=L1644,REGION=136K
```

While the procedure named RYIN is being executed, these three EXEC statements would appear as:

```
//STEP1 EXEC PGM=SECT,PARM=L1644,REGION=136K
//STEP2 EXEC PGM=PARA,COND=EVEN,REGION=136K
//STEP3 EXEC PGM=SENT,REGION=136K
```

Adding EXEC Statement Parameters

To add a parameter to an EXEC statement in the procedure, identify on the EXEC statement that calls the procedure the parameter you are adding, the name of the EXEC statement to which you want to add the parameter, and the value you are assigning to the parameter. The format required to add a parameter is:


```
parameter.procstepname=value
```

Parameters you are adding and overriding for a step must be coded before those parameters you are adding and overriding for the next step.

For example, if the first two EXEC statements of a procedure named GLEAN are:

```
//STEP1 EXEC PGM=FAC,COND=(8,LT)
//STEP2 EXEC PGM=UP,PARM=377685
```

and you want to override the COND parameter and add the ROLL parameter to the first EXEC statement, and add the REGION parameter to the second EXEC statement, the EXEC statement that calls the procedure would appear as:

```
//STPA EXEC GLEAN,COND.STEP1=(12,LT),
//          ROLL.STEP1=(NO,NO),REGION.STEP2=88K
```

Nullifying EXEC Statement Parameters

To nullify a parameter on an EXEC statement in the procedure, identify, on the EXEC statement that calls the procedure, the parameter you want to nullify and the name of the EXEC statement on which the parameter appears. The format required to nulify a parameter is:

```
parameter.procstepname=
```

Parameters that you are nullifying, overriding, and adding to a step must be coded before those for the next step.

For example, if the first two EXEC statements of a procedure named GINN are:

```
//STEP1 EXEC PGM=INV,PARM='146,899',RD=R
//STEP2 EXEC PGM=DET,PARM=XYA34,COND=(80,GT)
```

and you want to nullify the PARM parameter and add the COND parameter to the first EXEC statement, and override the COND parameter on the second EXEC statement, the EXEC statement that calls the procedure would appear as:

```
//STEPY EXEC GINN,PARM.STEP1=,COND.STEP1=(25,EQ),
//           COND.STEP2=(80,GE)
```

Example of Overriding, Adding, and Nullifying Parameters on an EXEC Statement

1. You want to call the following cataloged procedure named ESEAP:

```
//STEPA EXEC PGM=FLIER,PARM=7121190,ACCT=(4805,UNASGN)
//DDA   DD   DSNAME=LIBRARY.GROUP67,DISP=OLD
//DDB   DD   DSNAME=STAND.FIVE,DISP=OLD
//STEPB EXEC PGM=VERSE,DPRTY=(11,13),PARM=780684,RD=R
//DDC   DD   UNIT=2311,SPACE=(TRK,(10,2))
//DDD   DD   DSNAME=COL,DISP=OLD
//DDE   DD   DDNAME=IN
```

You want to make the following modifications to the procedure:

- Add the REGION parameter to both EXEC statements.
- Add the DPRTY parameter to the first EXEC statement.
- Override the ACCT parameter on the first EXEC statement.
- Nullify the RD parameter on the second EXEC statement.
- Add the COND parameter to the second EXEC statement.

The EXEC statement that calls the procedure would appear as:

```
//MINC EXEC ESEAP,REGION=86K,DPRTY.STEPA=(11,13),
//          ACCT.STEPA=(4805,7554),RD.STEPB=,COND.STEPB=(60,LE)
```

The two EXEC statements in the procedure would appear as shown below while the procedure is being executed. These modifications do not appear on an output listing.

```
//STEPA EXEC PGM=FLIER,PARM=7121190,ACCT=(4805,7554),
//           REGION=86K,DPRTY=(11,13)
//STEPB EXEC PGM=VERSE,DPRTY=(11,13),REGION=86K,COND=(60,LE)
```

2. You want to call the following in-stream procedure named INLINE:

```
//INLINE  PROC
//STEP1   EXEC  PGM=COMP,ACCT=(7037,2361),REGION=86K
//DD1     DD    DSNAME=INFORM,DISP=OLD,UNIT=2311,VLOLUME=SER=75250
//DD2     DD    DSNAME=LCJWC,DISP=OLD,UNIT=2311,VOLUME=SER=76250
//STEP2   EXEC  PGM=CHECKS,PARM=212334,COND=(50,LE),ACCT=(2001,0539)
//DD3     DD    DSNAME=PAY,DISP=OLD,UNIT=2311,VOLUME=SER=MEMORY
//DD4     DD    DSNAME=INCREAS,DISP=OLD,UNIT=2311,VOLUME=SER=33333
//        PEND
```

You want to make the following modifications to the procedure:

- Add DPRTY parameter to both EXEC statements.
- Nullify the REGION parameter on the first EXEC statement.
- Override the ACCT parameter on the second EXEC statement.

The EXEC statement that calls the procedure would appear as:

```
//CALLER  EXEC  INLINE,DPRTY=(11,13),REGION.STEP1=,ACCT.STEP2=(4710,
//              5390)
```

The two EXEC statements in the procedure would appear as shown below while the procedure is being executed. These modifications do not appear on an output listing.

```
//STEP1   EXEC  PGM=COMP,ACCT=(7037,2361), DPRTY=(11,13)
//STEP2   EXEC  PGM=CHECKS,PARM=212334,COND=(50,LE),DPRTY=(11,13),
//              ACCT=(4710,5390)
```


Overriding, Adding, and Nullifying Parameters on a DD Statement

You can override, add, or nullify parameters coded on a DD statement contained in a cataloged procedure. You make these changes at the time the procedure is called; these changes are in effect during the current execution of the procedure. Use one DD statement to override, add, and nullify parameters on the same DD statement in the procedure.

Overriding DD Statement Parameters

To override a parameter on a DD statement in the procedure, you must include a DD statement following the EXEC statement that calls the procedure. The ddname of this DD statement must identify the DD statement that contains the parameter you are overriding and the procedure step in which the DD statement appears. Code, in the operand field of this DD statement, the parameter you are overriding and the change; or code a mutually exclusive parameter that is to take the place of a parameter. The format required for a DD statement following the EXEC statement is:

```
//procstepname.ddname DD parameter=change
or
//procstepname.ddname DD mutually exclusive parameter=value
```

For example, if one of the DD statements in a procedure step named STEP4 is:

```
//DD2 DD DSNAME=ABIN,DISP=OLD,VOLUME=SER=54896,UNIT=2400
```

and you want to change UNIT=2400 to UNIT=180, you would code:

```
//STEP4.DD2 DD UNIT=180
```

When you code a mutually exclusive parameter on an overriding DD statement, the system replaces the parameter on the specified DD statement with the mutually exclusive parameter. For example, the parameters SYSOUT and DISP are mutually exclusive parameters. If one of the DD statements in a procedure step named PRINT is:

```
//DD8 DD SYSOUT=C
```

and you do not want the data set printed, you would code:

```
//PRINT.DD8 DD DUMMY,DISP=(NEW,DELETE)
```

You have replaced the SYSOUT parameter with the DISP parameter, and added the DUMMY parameter. (The DUMMY parameter causes this DD statement to define a dummy data set.)

You can change more than one parameter that appears on a DD statement in the procedure. For example, if one of the DD statements in a procedure step named STEP5 is:

```
//DDX DD DSNAME=FIES,DISP=OLD,UNIT=2400-2,VOLUME=REF=*.STEP2.DDC
```

and you want this DD statement to define a new data set, you would code:

```
//STEP5.DDX DD DSNAME=RVA1,DISP=(NEW,KEEP)
```

If you want to change parameters that appear on different DD statements in the same procedure step, the overriding DD statements must be in the same order in the input stream as the corresponding DD statements in the procedure step. For example, if the first step of a procedure named AJG is:

```
//STEP1 EXEC PGM=MGR,REGION=80K
//DD1 DD DSNAME=LONE,DISP=(NEW,DELETE),UNIT=2400,VOLUME=SER=568998
//DD2 DD UNIT=TAPE
//DD3 DD UNIT=2311,DISP=(,PASS),SPACE=(TRK,(20,2))
```

and you want to change the UNIT and VOLUME parameters on the first DD statement and the SPACE parameter on the third DD statement, you would code:

```
//CATP EXEC AJG
//STEP1.DD1 DD UNIT=2400-3,VOLUME=SER=WORK18
//STEP1.DD3 DD SPACE=(CYL,(4,1))
```

If you want to change parameters that appear in different procedure steps in the cataloged procedure you are calling, the overriding DD statements must be in the same order as are the procedure steps.

The DCB Parameter: If you want to change some of the subparameters in the DCB parameter, you need not recode the entire DCB parameter. Instead, code only those subparameters that you are changing and any mutually exclusive subparameters that are to replace particular subparameters. For example, if one of the DD statements in a procedure step named NED is:

```
//DD3 DD DSNAME=PER,DISP=(,KEEP),UNIT=2311,SPACE=(TRK,(88,5)),
//       DCB=(BUFNO=1,BLKSIZE=80,RECFM=F,BUFL=80)
```

and you want to change BLKSIZE=80 to BLKSIZE=320 and BUFL=80 to BUFL=320, you would code:

```
//NED.DD3 DD DCB=(BLKSIZE=320,BUFL=320)
```

The DCB subparameters BUFNO and RECFM remain unchanged.

Adding DD Statement Parameters

To add a parameter to a DD statement in the procedure, you must include a DD statement following the EXEC statement that calls the procedure. The ddname of this DD statement must identify the DD statement to which you are adding a parameter and the procedure step in which the DD statement appears. Code, in the operand field of this DD statement, the parameter you are adding. The format required for a DD statement following the EXEC statement is:

```
//procstepname.ddname DD parameter=value
```

For example, if one of the DD statements in a procedure step named STPTWO is:

```
//DDM DD DSNAME=TYPE,DISP=(,KEEP),UNIT=2400
```

and you want to add the VOLUME parameter, you would code:

```
//STPTWO.DDM DD VOLUME=SER=569433
```

If you want to add parameters or change parameters that appear on different DD statements, the overriding DD statements must be in the same order in the input stream as the corresponding DD statemetns in the procedure.

Nullifying DD Statement Parameters

There may be parameters on a DD statement that you do not want to override, but you want the system to ignore. Also, when you modify a DD statement in a procedure by overriding certain parameters or adding parameters, there may be some parameters remaining that no longer have meaning for your data set definition but would effect processing of the data set. To temporarily remove these parameters, you can nullify them. (If you are replacing a parameter with a mutually exclusive parameter, do not nullify the parameter that is being replaced.)

To nullify a parameter on a DD statement in the procedure, you must include a DD statement following the EXEC statement that calls the procedure. The ddname of this DD statement must identify the DD statement that contains the parameter you are nullifying and the procedure step in which the DD statement appears. Code in the operand field of this DD statement the parameter you are nullifying followed by an equal sign; do not follow the equal sign with a value. The format required for a DD statement following the EXEC statement is:

```
//procstepname.ddname DD parameter=
```

For example, if one of the DD statements in a procedure named SALLS is:

```
//DDP DD DSNAME=STEP,DISP=OLD,UNIT=2314,VOLUME=SER=556978
```

and you are overriding the DSNAME, DISP, and UNIT parameters, adding the DCB parameter, and want the VOLUME parameter ignored, you would code:

```
//SALLS.DDP DD DSNAME=&&TEMP,DISP=(,PASS),UNIT=2400-2,
//             DCB=(DEN=2,TRTCH=ET),VOLUME=
```

To nullify the DCB parameter, each DCB subparameter must be nullified individually. For example, if a DD statement contains

```
DCB=(RECFM=FBA,BLKSIZE=160,LRECL=80)
```

then

```
DCB=(RECFM=,BLKSIZE=,LRECL=)
```

must be coded on the overriding DD statement in order to nullify the DCB parameter.

To nullify a DUMMY parameter, code the DSNAME parameter on the overriding DD statement, but do not use the data set name NULLFILE. (Coding DSNAME=NULLFILE has the same effect as coding the DUMMY parameter.)

CAUTION: When you are overriding a procedure DD statement that contains the SPACE parameter and the overriding DD statement defines an existing data set, be sure to nullify the SPACE parameter. When a secondary quantity is coded on the procedure DD statement, the system uses this value to assign additional space to the data set instead of the secondary quantity you may have specified when the data set was created. Also, the RLSE subparameter, when specified on the procedure statement, causes the system to release any of the existing data set's unused space.

If you want to nullify, add, or override parameters that appear on different DD statements, the overriding DD statements must be in the same order in the input stream as the corresponding DD statements in the procedure.

Example of Overriding, Adding, and Nullifying Parameters on a DD Statement

1. You want to call the following procedure named SALL:

```
//STP1 EXEC PGM=GLF14
//DD11 DD DSNAME=XTRA.LEVEL,DISP=OLD
//DD12 DD DSNAME=CONDS,DISP=(,PASS),UNIT=2400
//DD13 DD DUMMY,DSNAME=LAST,VOLUME=REF=*.DD11,DISP=(,CATLG)
//STP2 EXEC PGM=FAIR
//DD21 DD DSNAME=*.STP1.DD12,DISP=(OLD,DELETE)
//DD22 DD DSNAME=JETZ,DISP=(NEW,KEEP),
//        UNIT=2311,SPACE=(CYL,(3,1),RLSE)
//DD23 DD SYSOUT=G
```

You want to modify the procedure as follows:

- Change the data set name on the DD12 statement from CONDS to C8495, and add the VOLUME parameter.
- Add the VOLUME parameter to the statement named DD12.
- Nullify the DUMMY parameter on the DD13 statement.
- Change the disposition on the DD21 statement from DELETE to KEEP.
- Define an existing data set on the DD22 statement.
- Add the parameters UNIT and SPACE on the DD23 statement.

The EXEC statement that calls the procedure and the overriding DD statements that follow it would appear as:

```
//CALL EXEC SALL
//STP1.DD12 DD DSNAME=C8495,VOLUME=SER=979354
//STP1.DD13 DD DSNAME=LAST
//STP2.DD21 DD DISP=(OLD,KEEP)
//STP2.DD22 DD SPACE=,DSNAME=GR1833,DISP=OLD,LABEL=(,NL),
//             VOLUME=SER=577632
//STP2.DD23 DD UNIT=2314,SPACE=(CYL,(150,15))
```

The cataloged procedure would appear as shown below while the procedure is being executed. These modifications do not appear on an output listing.

```
//STP1 EXEC PGM=GLF14
//DD11 DD DSNAME=XTRA.LEVEL,DISP=OLD
//DD12 DD DSNAME=C8495,DISP=(,PASS),UNIT=2400,VOLUME=SER=979354
//DD13 DD DSNAME=LAST,VOLUME=REF=*.DD11,DISP=(,CATLG)
//STP2 EXEC PGM=FAIR
//DD21 DD DSNAME=*.STP1.DD12,DISP=(OLD,KEEP)
//DD22 DD DSNAME=GR1833,DISP=OLD,UNIT=2311,LABEL=(,NL),
//        VOLUME=SER=577632
//DD23 DD SYSOUT=G,UNIT=2314,SPACE=(CYL,(150,15))
```

2. You want to call the following in-stream procedure named CARDS:

```
//CARDS   PROC
//STEPA   EXEC  PGM=FIGURE
//DDA1    DD    DSNAME=NUMBERS,DISP=OLD
//DDA2    DD    DSNAME=PROCESS,DISP=(,PASS),UNIT=2311,
//              SPACE=(TRK,(1,1,1))
//STEPB   EXEC  PGM=RESULT
//DDB1    DD    DSNAME=VSC,DISP=OLD
//DDB2    DD    DSNAME=*.STEPA.DDA2,DISP=(OLD,KEEP)
//DDB3    DD    SYSOUT=C
//        PEND
```

You want to modify the procedure as follows:

- Change the data set name on the DDA1 statement from NUMBERS to NAMES.
- Add the VOLUME parameter to the DDA2 statement.
- Add the parameters UNIT and SPACE on the DDB3 statement.

The EXEC statement that calls the procedure and the overriding DD statements that follow it would appear as:

```
//CALL         EXEC  CARDS
//STEPA.DDA1   DD    DSNAME=NAMES
//STEPA.DDA2   DD    VOLUME=SER=5858
//STEPB.DDB3   DD    UNIT=2311,SPACE=(TRK,(150,15))
```

The in-stream procedure would appear as shown below while the procedure is being executed. These modifications do not appear on an output listing. The PROC statement is processed only when it contains symbolic parameters.

```
//STEPA   EXEC  PGM=FIGURE
//DDA1    DD    DSNAME=NAMES,DISP=OLD
//DDA2    DD    DSNAME=PROCESS,DISP=(,PASS),UNIT=2311,
//              SPACE=(TRK,(1,1,1)),VOLUME=SER=5858
//STEPB   EXEC  PGM=RESULT
//DDB1    DD    DSNAME=VSC,DISP=OLD
//DDB2    DD    DSNAME=*.STEPA.DDA2,DISP=(OLD,KEEP)
//DDB3    DD    SYSOUT=C,UNIT=2311,SPACE=(TRK,(150,15))
```

Overriding DD Statements That Define Concatenated Data Sets

When a concatenation of data sets is defined in a cataloged procedure and you attempt to override the concatenation with one DD statement, only the first (named) DD statement is overridden. To override others, you must include an overriding DD statement for each DD statement; the DD statements in the input stream must be in the same order as the DD statements in the procedure. The second and subsequent overriding statements must not be named. If you do not wish to change one of the concatenated DD statements, leave the operand field blank on the corresponding DD statement in the input stream. (This is the only case where a blank operand field for a DD statement is valid.)

For example, suppose you are calling a procedure that includes the following sequence of DD statements in STEPC:

```
//DD4 DD DSNAME=A.B.C,DISP=OLD
//    DD DSNAME=STRP,DISP=OLD,UNIT=2311,VOLUME=SER=X12182
//    DD DSNAME=TYPE3,DISP=OLD,UNIT=2311,VOLUME=SER=BL142
//    DD DSNAME=A.B.D,DISP=OLD
```

If you want to override the DD statements that define the data sets named STRP and A.B.D, the sequence of DD statements in the input stream would appear as:

```
//STEPC.DD4 DD
//          DD DSNAME=INV.CLS,DISP=OLD
//          DD
//          DD DSNAME=PAL8,DISP=OLD,UNIT=2311,VOL=SER=125688
```

Adding DD Statements to a Procedure

You can add DD statements to a procedure when you call the procedure. These additional DD statements are in effect only while the procedure is being executed.

To add a DD statement to a procedure step, follow the EXEC statement that calls the procedure and any overriding DD statements for that step with the additional DD statement. The ddname of this DD statement must identify the procedure step to which this statement is to be added and must be assigned a name that is different from all the ddnames in the procedure step. The format required for a DD statement following the EXEC statement is:

```
//procstepname.ddname DD parameters
```

For example, if the first step of a cataloged procedure named MART is:

```
//STEP1 EXEC PGM=DATE
//DDM DD DSNAME=BPS(MEMG),DISP=OLD,UNIT=2311,VOLUME=SER=554982
//DDN DD UNIT=SYSQE
```

And you want to change the UNIT parameter on the DD statement named DDN and add a DD statement, you would code:

```
//PROC EXEC MART
//STEP1.DDN DD UNIT=180
//STEP1.DDO DD UNIT=181
```

In PCP, if you are adding more than one DD statement to a procedure step and one of the statements is a DD * or DD DATA statement, the DD * or DD DATA statement must be last.

Example of Adding DD Statements to a Procedure

1. You want to call the following procedure named D995A:

```
//SA EXEC PGM=ANALY
//DDA1 DD DSNAME=PROJ.C843,DISP=OLD
//DDA2 DD DDNAME=SYSIN
//DDA3 DD SYSOUT=B
//SB EXEC PGM=MANM03
//DDB1 DD UNIT=2400
//DDB2 DD UNIT=2400
//DDB3 DD DSNAME=X54,VOLUME=SER=(36544,36545),UNIT=(2400,2),
//         DISP=(OLD,KEEP)
```

You want to modify the procedure as follows:

1. Supply the data set definition for the DDA2 statement by adding a DD statement.
2. Change the SYSOUT parameter on the DDA3 statement to UNIT=1403.
3. Add a DD statement to the step named SB.

The EXEC statement that calls the procedure and the overriding and additional DD statements that follow it would appear as:

```
//PROCED EXEC D995A
//SA.DDA3 DD UNIT=1403
//SA.SYSIN DD *
.
.
.
 data
.
.
.
/*
//SB.DDB4 DD UNIT=(2400,SEP=(DDB1,DDB2))
```

The cataloged procedure would appear as shown below while the procedure is being executed. These modifications do not appear on output listing.

```
//SA EXEC PGM=ANALY
//DDA1 DD DSNAME=PROJ.C843,DISP=OLD
//DDA2 DD *
//DDA3 DD UNIT=1403
//SB EXEC PGM=MANM03
//DDB1 DD UNIT=2400
//DDB2 DD UNIT=2400
//DDB3 DD DSNAME=X54,VOLUME=SER=(36544,36545),UNIT=(2400,2),
//         DISP=(OLD,KEEP)
//DDB4 DD UNIT=(2400,SEP=(DDB1,DDB2))
```

2. You want to call the following in-stream procedure named WORK:

```
//WORK  PROC
//STP1  EXEC  PGM=PROD
//DD1   DD    DSNAME=PROJECT,DISP=OLD
//DD2   DD    DDNAME=SYSIN
//      PEND
```

You want to modify the procedure by supplying the data set definition for the DD2 statement by adding a DD statement.

The EXEC statement that calls the procedure and the additional DD statement that follows it would appear as

```
//ADD          EXEC  WORK
//STP1.SYSIN   DD    *
.
.
.
data
.
.
.
.
/*
```

The in-stream procedure would appear as shown below while the procedure is being executed. These modifications do not appear on the output listing.

```
//STP   EXEC  PGM=PROD
//DD1   DD    DSNAME=PROJECT,DISP=OLD
//DD2   DD    *
```

Writing Procedures: Cataloged and In-stream

Why Catalog JCL Statements

Applications performed at your installation on a regular basis and applications that require many control statements can be simplified when the control statements for these applications are cataloged. Once the job control statements for an application are cataloged on the procedure library, any programmer who wants to perform the application need only submit a JOB and EXEC statement. On the EXEC statement, he refers the system to the control statements required to perform the application. If there are modifications the programmer wants to make for the duration of the job step, he assigns values to symbolic parameters on the EXEC statement and follows the EXEC statement with overriding DD statements.

Why Use In-stream Procedures

While in-stream procedures do not have to be added to the procedure library, they eliminate the necessity of repeating the same set of control statements in ajob. An in-stream procedure can be executed any number of times during a job in which it appears and fifteen uniquely named in-stream procedures can appear in one job. In-stream procedures can be modified just as cataloged procedures. They also provide you with a means of testing procedures before adding them to the procedure library as cataloged procedures. Because an in-stream procedure exists in the form of cards, it can be considered a "portable procedure" in that it can easily be moved from one input stream to another.

The Contents of Cataloged and In-stream Procedures

Cataloged and in-stream procedures contain one or more EXEC statements, each followed by associated DD statements. Each EXEC statement identifies the program to be executed, and the DD statements that follow define the input, output, and work data sets to be used by the program. Each EXEC statement and its associated DD statements are called a procedure step.

Cataloged and in-stream procedures cannot contain:

1. EXEC statements that refer to other cataloged procedures.
2. JOB, delimiter, or null statements.
3. DD statements with the ddname JOBLIB.
4. DD statements with * or DATA coded in the operand field.

A cataloged or in-stream procedure can contain a DD statement with the ddname STEPLIB. If a procedure step requires use of a program in a private library other than SYS1.LINKLIB, you define that library on this DD statement. If the DD statement is not overridden when the procedure is called, it makes the private library available to the step. (For information on the STEPLIB DD statement, see "Special DD Statements" in the section "The DD Statement.")

For ease in modifying a cataloged procedure, you can include symbolic parameters in the procedure. How to use symbolic parameters is described next.

Using Symbolic Parameters in a Procedure

When you prepare the control statements that you plan to catalog, you can include symbolic parameters. A symbolic parameter is characterized by a name preceded by an ampersand (&) and appears in the operand field of a cataloged procedure statement. A symbolic parameter stands for a parameter, a subparameter, or a value.

Symbolic parameters allow a programmer who calls the procedure to easily modify the procedure for the duration of the job step. When the programmer calls the procedure, he assigns values to the symbolic parameters on the EXEC statement. When you prepare control statements that you plan to catalog, you can include a PROC statement and assign default values to any of the symbolic parameters that are included.

When you prepare control statements to be used as an in-stream procedure, you must include a PROC statement which can be used to assign default values to any of the symbolic parameters that are included.

A symbolic parameter is one to seven alphameric and national (#,@,$) characters preceded by a single ampersand. The first character must be alphabetic or national. Since a single ampersand defines a symbolic parameter, you code double ampersands when you are not defining a symbolic parameter. For example, if you want to pass 543&LEV to a processing program by means of the PARM parameter on an EXEC statement, you must code

```
PARM='54&&LEV'
```

The system treats the double ampersands as if a single ampersand has been coded, and only one ampersand appears in the results.

The following are examples of symbolic parameters:

```
//STEP1 EXEC PGM=COB,PARM='P1,&P2,P3'
//DD1 DD DSNAME=&&FIX,UNIT=&DEVICE,SPACE=(CYL,(&SPACE,10))
//DD2 DD DSNAME=&&CHAG,UNIT=2400,DCB=BLKSIZE=&LENGTH
```

Keyword parameters that can be coded on an EXEC statement cannot be used to define symbolic parameters. For example, &PGM and &REGION cannot be used as symbolic parameters.

Any parameter, subparameter, or value in the procedure that may vary each time the procedure is called is a good candidate for definition as a symbolic parameter. For example, if different values can be passed to a processing program by means of the PARM parameter on one of the EXEC statements, you might define the PARM parameter field as one or more symbolic parameters,

```
PARM=&ALLVALS
```

or

```
PARM=&DECK&CODE
```

If symbolic parameters are defined in the cataloged procedures used at your installation, the definitions should be consistent. For example, every time the programmer is to assign his department number to a symbolic parameter, no matter which procedure he is calling, the symbolic parameter could be defined as &DEPT. In different procedures you could code

```
ACCT=(43877,&DEPT)
```

and

```
DSNAME=LIBRARY.&DEPT.MACS
```

The programmer would assign his department number on the EXEC statement that calls the procedure whenever &DEPT appears in a procedure. Of course, in order for the programmer to know that he is to assign his department number to the symbolic parameter &DEPT, the installation must make this information available to all the programmers that may be using the cataloged procedures.

You can define two or more symbolic parameters in succession without including a comma to delimit the symbolic parameters, for example, &P1&P2. You can also define a potion of a parameter, subparameter, or value as a symbolic parameter. You do this by placing the symbolic parameter before, after, or in between the information that is not variable.

If you place a symbolic parameter after some information that does not vary, it is not necessary to code a delimiter. The system recognizes a symbolic parameter when it encounters the single ampersand.

If you place a symbolic parameter before some information that does not vary, a period may be required following the symbolic parameter to distinguish the end of the symbolic parameter and the beginning of the information that does not vary. A period is required following the symbolic parameter when:

1. The character following the symbolic parameter is an alphabetic or numeric character.

2. The character following the symbolic parameter is a left parenthesis or a period.

In these cases, the system recognizes the period as a delimiter, and the period does not appear after a value is assigned to the symbolic parameter. (A period will appear after a value is assigned to the symbolic parameter when two consecutive periods are coded.)

The following examples are valid ways of combining symbolic parameters and information that does not vary.

Placing a symbolic parameter after information that does not vary;

1. LIBRARY(&MEMBER)

2. USERLIB.&LEVEL

Placing a symbolic parameter before information that does not vary;

1. '&OPTION+15'

2. &PASS.A43B8

   The period is required because an alphabetic character follows the symbolic parameter.

3. &URNO.54328

   The period is required because a numeric character follows the symbolic parameter.

4. &LIBRARY.(MEMG)

   The period is required because a left parenthesis follows the symbolic parameter.

5. &FILL..GROUP5

A period is to appear in the results; therefore, two consecutive periods are coded.

When a value is assigned to the symbolic parameter, this value and the parameter, subparameter, or value that this is a portion of cannot exceed 120 characters.

The programmer who calls a procedure assigns values to the symbolic parameters contained in the procedure. He can also nullify symbolic parameters. A delimiter, such as a leading comma or a trailing comma, next to a symbolic parameter is not automatically removed when the symbolic parameter is nullified. For example, if the operand field contains

```
VOLUME=SER=(111111,&KEY)
```

the comma preceding &KEY is not removed when &KEY is nullified. If the symbolic parameter that is nullified is a positional parameter, a comma must remain to indicate its absence. In other cases, a delimiter that is not removed when the symbolic parameter is nullified may cause a syntax error. To help the programmer who nullifies a symbolic parameter avoid this error condition, define those symbolic parameters that may be nullified without the delimiter. For example, you could code

```
VOLUME=SER=(111111&KEY)
```

The delimiter is included when a value is assigned to the symbolic parameter. For example, the programmer would code

```
KEY=',222222'
```

A cataloged procedure statement may utilize DDNAME and DCB parameters to define data in the input stream. Such a statement should not contain symbolic parameters when the automatic SYSIN batching reader is used. (Information on the cataloged procedure for the automatic SYSIN batching reader is contained in the publication IBM System/360 Operating System: System Programmer's Guide.)

The PROC Statement: When establishing cataloged or in-stream procedures that contain symbolic parameters it is generally good practice to assign default values to the symbolic parameters. These default values are used if the programmer who calls the procedure does not assign values to one or more of the symbolic parameters.

You assign default values on a PROC statement. The PROC statement is optional for cataloged procedures; if it is used, the PROC statement must be the first statement in the procedure. The PROC statement is described in the section "The PROC Statement."

The PEND statement which is used to mark the end of an in-stream procedure is described in Section IX.

Adding and Modifying Cataloged Procedures

You add procedures to the procedure library by using the IEBUPDTE utility program. You also use this utility program to permanently modify existing procedures. How to use this utility program for adding and modifying cataloged procedures is described in the chapter "The IEBUPDTE Program" in the publication IBM System/360 Operating System: Utilities.

If you use MFT or MVT when you add a cataloged procedure to the procedure library, that procedure cannot be executed before the job that adds it to the procedure library terminates. If you use MFT or MVT when you modify an existing cataloged procedure, the operator must be notified. What the operator must do before he allows the job to be executed is described in the chapter "How to Run Jobs That Update System Data Sets" in the publication IBM System/360 Operating System: Operator's Guide

Part IV: Examples of Cataloged Procedures for Compilations, Link Edits, and Executions

Part IV shows examples of the cataloged procedures provided by IBM for various compilers. They are:

- PL/I F

PL1DFC	(Compile with deck output)	Figure 45
PL1LFC	(Compile with load module output)	Figure 46
PL1LFCL	(Compile-Link Edit)	Figure 47
PL1LFLG	(Link Edit-Go)	Figure 48
PL1LFCLG	(Compile-Link Edit-Go)	Figure 49
PL1LFCG	(Compile-Load)	Figure 50
PL1LFG	(Load)	Figure 51

These cataloged procedures are presented here for your convenience while learning JCL. For further details and up-to-date information you should refer to the Programmer's Guide associated with each compiler.

A section with examples of how to call and modify the procedures follows the Figures.

ALGOL

```
//ALGOL     EXEC PGM=ALGOL,REGION=48K
//SYSPRINT  DD   SYSOUT=A
//SYSPUNCH  DD   SYSOUT=B
//SYSLIN    DD   DSN=&LOADSET,UNIT=SYSSQ,SEP=SYSPUNCH,
//               SPACE=(3600,(10,4)),DISP=(MOD,PASS)
//SYSUT1    DD   DSN=&SYSUT1,UNIT=SYSSQ,SEP=SYSPRINT,
//               SPACE=(1024,(50,10))
//SYSUT2    DD   DSN=&SYSUT2,UNIT=SYSSQ,SEP=SYSUT1,
//               SPACE=(1024,(50,10))
//SYSUT3    DD   DSN=&SYSUT3,UNIT=SYSDA,SPACE=(1024,(40,10))
```

NOTE: You must provide an ALGOL.SYSIN DD statement when you call the procedure.

Figure 11. ALGOFC (Compile)

```
//ALGOL     EXEC PGM=ALGOL,REGION=48K
//SYSPRINT  DD   SYSOUT=A
//SYSPUNCH  DD   SYSOUT=B
//SYSLIN    DD   DSN=&LOADSET,UNIT=SYSSQ,SEP=SYSPUNCH,
//               SPACE=(3600,(10,4)),DISP=(MOD,PASS)
//SYSUT1    DD   DSN=&SYSUT1,UNIT=SYSSQ,SEP=SYSPRINT,
//               SPACE=(1024,(50,10))
//SYSUT2    DD   DSN=&SYSUT2,UNIT=SYSSQ,SEP=SYSUT1,
//               SPACE=(1024,(50,10))
//SYSUT3    DD   DSN=&SYSUT3,UNIT=SYSDA,SPACE=(1024,(40,10))
//LKED      EXEC PGM=IEWL,PARM='XREF,LIST,LET',COND=(5,LT,ALGOL),
//               REGION=96K
//SYSPRINT  DD   SYSOUT=A
//SYSLIN    DD   DSN=&LOADSET,DISP=(OLD,DELETE)
//          DD   DDNAME=SYSIN
//SYSLIB    DD   DSN=SYS1.ALGLIB,DISP=SHR
//SYSLMOD   DD   DSN=&GOSET(GO),UNIT=SYSDA,DISP=(MOD,PASS),
//               SPACE=(1024,(50,20,1))
//SYSUT1    DD   DSN=&SYSUT1,UNIT=SYSDA,SEP=(SYSLIB,SYSLMOD),
//               SPACE=(1024,(50,20))
```

Note: You must provide an ALGOL.SYSIN DD statement when you call the procedure.

Figure 12. ALGOFCL (Compile-Link Edit)

```
//ALGOL     EXEC PGM=ALGOL,REGION=48K
//SYSPRINT  DD   SYSOUT=A
//SYSPUNCH  DD   SYSOUT=B
//SYSLIN    DD   DSN=&LOADSET,UNIT=SYSSQ,SEP=SYSPUNCH,
//               SPACE=(3600,(10,4)),DISP=(MOD,PASS)
//SYSUT1    DD   DSN=&SYSUT1,UNIT=SYSSQ,SEP=SYSPRINT,
//               SPACE=(1024,(50,10))
//SYSUT2    DD   DSN=&SYSUT2,UNIT=SYSSQ,SEP=SYSUT1,
//               SPACE=(1024,(50,10))
//SYSUT3    DD   DSN=&SYSUT3,UNIT=SYSDA,SPACE=(1024,(40,10))
//LKED      EXEC PGM=IEWL,PARM='XREF,LIST,LET',COND=(5,LT,ALGOL),
//               REGION=96K
//SYSPRINT  DD   SYSOUT=A
//SYSLIN    DD   DSN=&LOADSET,DISP=(OLD,DELETE)
//          DD   DDNAME=SYSIN
//SYSLIB    DD   DSN=SYS1.ALGLIB,DISP=SHR
//SYSLMOD   DD   DSN=&GOSET(GO),UNIT=SYSDA,DISP=(MOD,PASS),
//               SPACE=(1024,(50,20,1))
//SYSUT1    DD   DSN=&SYSUT1,UNIT=SYSDA,SEP=(SYSLIB,SYSLMOD),
//               SPACE=(1024,(50,20))
//GO        EXEC PGM=*.LKED.SYSLMOD,COND=((5,LT,ALGOL),(5,LT,LKED))
//ALGLDD01  DD   SYSOUT=A
//SYSPRINT  DD   SYSOUT=A
//SYSUT1    DD   DSN=&SYSUT1,UNIT=SYSSQ,SPACE=(1024,(20,10))
```

Note: You must provide an ALGOL.SYSIN DD statement when you call the procedure.

Figure 13. ALGOFCLG (Compile-Link Edit-Go)

```
//ALGOL     EXEC PGM=ALGOL,REGION=48K
//SYSPRINT  DD   SYSOUT=A
//SYSPUNCH  DD   SYSOUT=B
//SYSLIN    DD   DSN=&LOADSET,UNIT=SYSSQ,SEP=SYSPUNCH,
//               SPACE=(3600,(10,4)),
//               DISP=(MOD,PASS)
//SYSUT1    DD   DSN=&SYSUT1,UNIT=SYSSQ,SEP=SYSPRINT,
//               SPACE=(1024,(50,10))
//SYSUT2    DD   DSN=&SYSUT2,UNIT=SYSSQ,SEP=SYSUT1,
//               SPACE=(1024,(50,10))
//SYSUT3    DD   DSN=&SYSUT3,UNIT=SYSDA,SPACE=(1024,(40,10))
//GO        EXEC PGM=LOADER,PARM=(MAP,LET,PRINT),COND=(5,LT,ALGOL)
//SYSLIN    DD   DSN=&LOADSET,DISP=(OLD,DELETE)
//SYSLIB    DD   DSN=SYS1.ALGLIB,DISP=SHR
//LOADPRNT  DD   SYSOUT=A
//SYSPRINT  DD   SYSOUT=A
//ALGLDD01  DD   SYSOUT=A
//SYSUT1    DD   DSN=&SYSUT1,UNIT=SYSSQ,SPACE=(1024,(20,10))
```

Note: You must provide an ALGOL.SYSIN DD statement when you call the procedure.

Figure 14. ALGOFCG (Compile-Load)

```
//ASM      EXEC PGM=IETASM
//SYSLIB   DD   DSNAME=SYS1.MACLIB,DISP=OLD
//SYSUT1   DD   UNIT=SYSSQ,SPACE=(400,(400,50))
//SYSUT2   DD   UNIT=SYSSQ,SPACE=(400,(400,50))
//SYSUT3   DD   UNIT=(SYSSQ,SEP=(SYSUT2,SYSUT1,SYSLIB)),
//              SPACE=(400,(400,50))
//SYSPRINT DD   SYSOUT=A
//SYSPUNCH DD   UNIT=SYSCP
```

Note: You must provide an ASM.SYSIN DD statement when you call the procedure.

Figure 15. ASMEC (Compile)

```
//ASM      EXEC PGM=IETASM
//SYSLIB   DD   DSNAME=SYS1.MACLIB,DISP=OLD
//SYSUT1   DD   UNIT=SYSSQ,SPACE=(400,(400,50))
//SYSUT2   DD   UNIT=SYSSQ,SPACE=(400,(400,50))
//SYSUT3   DD   UNIT=(SYSSQ,SEP=(SYSUT2,SYSUT1,SYSLIB)),
//              SPACE=(400,(400,50))
//SYSPRINT DD   SYSOUT=A
//SYSPUNCH DD   DSNAME=&LOADSET,UNIT=SYSSQ,SPACE=(80,(200,50)),
//              DISP=(MOD,PASS)
//LKED     EXEC PGM=IEWL,PARM=(XREF,LIST,NCAL)
//SYSLIN   DD   DSNAME=&LOADSET,DISP=(OLD,DELETE)
//         DD   DDNAME=SYSIN
//SYSLMOD  DD   DSNAME=&GOSET(GO) UNIT=SYSDA,SPACE=(1024,(50,20,1))
//SYSUT1   DD   UNIT=(SYSDA,SEP=(SYSLIN,SYSLMOD)),
//              SPACE=(1024,(50,20))
//SYSPRINT DD   SYSOUT=A
```

Note: You must provide an ASM.SYSIN DD statement when you call the procedure.

Figure 16. ASMECL (Compile-Link Edit)

```
//ASM      EXEC PGM=IETASM
//SYSLIB   DD   DSNAME=SYS1.MACLIB,DISP=OLD
//SYSUT1   DD   UNIT=SYSSQ,SPACE=(400,(400,50))
//SYSUT2   DD   UNIT=SYSSQ,SPACE=(400,(400,50))
//SYSUT3   DD   UNIT=(SYSSQ,SEP=(SYSUT2,SYSUT1,SYSLIB)),
//              SPACE=(400,(400,50))
//SYSPRINT DD   SYSOUT=A
//SYSPUNCH DD   DSNAME=&LOADSET,UNIT=SYSSQ,SPACE=(80,(200,50)),
//              DISP=(MOD,PASS)
//LKED     EXEC PGM=IEWL,PARM=(XREF,LET,LIST,NCAL)
//SYSLIN   DD   DSNAME=&LOADSET,DISP=(OLD,DELETE)
//         DD   DDNAME=SYSIN
//SYSLMOD  DD   DSNAME=&GOSET(GO),UNIT=SYSDA,SPACE=(1024,(50,20,1)),
//              DISP=(MOD,PASS)
//SYSUT1   DD   UNIT=(SYSDA,SEP=(SYSLIN,SYSLMOD)),
//              SPACE=(1024,(50,20))
//SYSPRINT DD   SYSOUT=A
//GO       EXEC PGM=*.LKED.SYSLMOD
```

Note: You must provide an ASM.SYSIN DD statement when you call the procedure.

Figure 17. ASMECLG (Compile-Link Edit-Go)

```
//ASM      EXEC PGM=IETASM,PARM='LOAD'
//SYSLIB   DD   DSNAME=SYS1.MACLIB,DISP=OLD
//SYSUT1   DD   UNIT=SYSSQ,SPACE(400,(400,50))
//SYSUT2   DD   UNIT=SYSSQ,SPACE=(400,(400,50))
//SYSUT3   DD   UNIT=(SYSSQ,SEP=(SYSUT2,SYSUT1,SYSLIB)),
//              SPACE=(400,(400,50))
//SYSPRINT DD   SYSOUT=A
//SYSPUNCH DD   DSNAME=&LOADSET,UNIT=SYSSQ,
//              SPACE=(80,(200,50)),DISP=(MOD,PASS)
//GO       EXEC PGM=LOADER,PARM='MAP,PRINT,NOCALL,LET'
//SYSLIN   DD   DSNAME=&LOADSET,DISP=(OLD,DELETE)
//SYSLOUT  DD   SYSOUT=A
```

Note: You must provide an ASM.SYSIN DD statement when you call the procedure.

Figure 18. ASMECG (Compile-Load)

```
//ASM      EXEC PGM=IEUASM,REGION=50K
//SYSLIB   DD   DSNAME=SYS1.MACLIB,DISP=SHR
//SYSUT1   DD   DSNAME=&SYSUT1,UNIT=SYSSQ,SPACE=(1700,(400,50)),
//              SEP=SYSLIB
//SYSUT2   DD   DSNAME=&SYSUT2,UNIT=SYSSQ,SPACE=(1700,(400,50))
//SYSUT3   DD   DSNAME=&SYSUT3,UNIT=(SYSSQ,SEP=(SYSUT2,SYSUT1,
//              SYSLIB)),SPACE=(1700,(400,50))
//SYSPRINT DD   SYSOUT=A
//SYSPUNCH DD   SYSOUT=B
```

Note: You must provide an ASM.SYSIN DD statement when you call the procedure.

Figure 19. ASMFC (Compile)

```
//ASM      EXEC PGM=IEUASM,PARM=LOAD,REGION=50K
//SYSLIB   DD   DSNAME=SYS1.MACLIB,DISP=SHR
//SYSUT1   DD   DSNAME=&SYSUT1,UNIT=SYSSQ,SPACE=(1700,(400,50)),
//              SEP=SYSLIB
//SYSUT2   DD   DSNAME=&SYSUT2,UNIT=SYSSQ,SPACE=(1700,(400,50))
//SYSUT3   DD   DSNAME=&SYSUT3,UNIT=(SYSSQ,SEP=(SYSUT2,SYSUT1,
//              SYSLIB)),SPACE=(1700,(400,50))
//SYSPRINT DD   SYSOUT=A
//SYSPUNCH DD   SYSOUT=B
//SYSGO    DD   DSNAME=&LOADSET,UNIT=SYSSQ,SPACE=(80,(100,50)),
//              DISP=(MOD,PASS)
//LKED     EXEC PGM=IEWL,PARM=(XREF,LIST,NCAL),REGION=96K,
//              COND=(8,LT,ASM)
//SYSLIN   DD   DSNAME=&LOADSET,DISP=(OLD,DELETE),
//         DD   DDNAME=SYSIN
//SYSLMOD  DD   DSNAME=&GOSET(GO),UNIT=SYSDA,SPACE=(1024,(50,20,1)),
//              DISP=(MOD,PASS)
//SYSUT1   DD   DSNAME=&SYSUT1,UNIT=(SYSDA,SEP=(SYSLIN,SYSLMOD)),
//              SPACE=(1024,(50,20))
//SYSPRINT DD   SYSOUT=A
```

Note: You must provide an ASM.SYSIN DD statement when you call the procedure.

Figure 20. ASMFCL (Compile-Link Edit)

```
//ASM      EXEC PGM=IEUASM,PARM=LOAD,REGION=50K
//SYSLIB   DD   DSNAME=SYS1.MACLIB,DISP=SHR
//SYSUT1   DD   DSNAME=&SYSUT1,UNIT=SYSSQ,SPACE=(1700,(400,50)),
//              SEP=SYSLIB
//SYSUT2   DD   DSNAME=&SYSUT2,UNIT=SYSSQ,SPACE=(1700,(400,50))
//SYSUT3   DD   DSNAME=&SYSUT3,UNIT=(SYSSQ,SEP=(SYSUT2,SYSUT1,
//              SYSLIB)),SPACE=(1700,(400,50))
//SYSPRINT DD   SYSOUT=A
//SYSPUNCH DD   SYSOUT=B
//SYSGO    DD   DSNAME=&LOADSET,UNIT=SYSSQ,SPACE=(80,(100,50)),
//              DISP=(MOD,PASS)
//LKED     EXEC PGM=IEWL.PARM=(XREF,LET,LIST,NCAL),REGION=96K,
//              COND=(8,LT,ASM)
//SYSLIN   DD   DSNAME=&LOADSET,DISP=(OLD,DELETE)
//        DD    DDNAME=SYSIN
//SYSLMOD  DD   DSNAME=&GOSET(GO),UNIT=SYSDA,SPACE=(1024,(50,20,1)),
//              DISP=(MOD,PASS)
//SYSUT1   DD   DSNAME=&SYSUT1,UNIT=(SYSDA,SEP=(SYSLIN,SYSLMOD)),
//              SPACE=(1024,(50,20))
//SYSPRINT DD   SYSOUT=A
//GO       EXEC PGM=*.LKED.SYSLMOD,COND=((8,LT,ASM),(4,LT,LKED))
```

Note: You must provide an ASM.SYSIN DD statement when you call.

Figure 21. ASMFCLG (Compile-Link Edit-Go)

```
//ASM      EXEC PGM=IEUASM,PARM='LOAD',REGION=50K
//SYSLIB   DD   DSNAME=SYS1.MACLIB,DISP=SHR
//SYSUT1   DD   DSNAME=&SYSUT1,UNIT=SYSSQ,SPACE=(1700,(400,50)),
//              SEP=(SYSLIB)
//SYSUT2   DD   DSNAME=&SYSUT2,UNIT=SYSSQ,SPACE=(1700,(400,50))
//SYSUT3   DD   DSNAME=&SYSUT3,SPACE=(1700,(400,50)),
//              UNIT=(SYSSQ,SEP=(SYSUT2,SYSUT1,SYSLIB))
//SYSPRINT DD   SYSOUT=A
//SYSPUNCH DD   SYSOUT=B
//SYSGO    DD   DSNAME=&LOADSET,UNIT=SYSSQ,SPACE=(80,(200,50)),
                DISP=(MOD,PASS)
//GO       EXEC PGM=LOADER,PARM='MAP,PRINT,NOCALL,LET'
//SYSLIN   DD   DSNAME=&LOADSET,DISP=(OLD,DELETE)
//SYSLOUT  DD   SYSOUT=A
```

Note: You must provide an ASM.SYSIN DD statement when you call the procedure.

Figure 22. ASMFCG (Compile-Load)

```
//COB       EXEC PGM=IEPCBL00,REGION=24K
//SYSPRINT  DD   SYSOUT=A
//SYSPUNCH  DD   UNIT=SYSCP
//SYSUT1    DD   UNIT=SYSDA,SPLIT=(2,CYL,(40,10))
//SYSUT2    DD   UNIT=SYSDA,SPLIT=4
//SYSUT3    DD   UNIT=SYSDA,SPLIT=4
//SYSIN     DD   DDNAME=SYSIN,DCB=BLKSIZE=80
```

Note: You must provide a COB.SYSIN DD statement when you call the procedure.

Figure 23. COBEC (Compile)

```
//LKED      EXEC PGM=IEWL,REGION=96K,PARM=(XREF,LIST,LET)
//SYSLIB    DD   DSNAME=SYS1.COBLIB,DISP=(SHR,KEEP)
//SYSLMOD   DD   DSNAME=&GODATA(RUN),DISP=(NEW,PASS),
//               UNIT=SYSDA,SPACE=(1024,(50,20,1))
//SYSUT1    DD   UNIT=(SYSDA,SEP=(SYSLIN,SYSLMOD)),
//               SPACE=(1024,(50,20))
//SYSPRINT  DD   SYSOUT=A
//SYSLIN    DD   DDNAME=SYSIN
//GO        EXEC PGM=*.LKED.SYSLMOD,COND=(5,LT,LKED)
//SYSABEND  DD   SYSOUT=A
//         DD   SYSOUT=A,DCB=(BLKSIZE=120,LRECL=120)
```

Note: You must provide a LKED.SYSIN DD statement when you call the procedure.

Figure 24. COBELG (Link Edit-Go)

```
//COB       EXEC PGM=IEPCBL00,REGION=24K
//SYSPRINT  SS   SYSOUT=A
//SYSUT1    DD   UNIT=SYSDA,SPLIT=(2,CYL,(40,10))
//SYSUT2    DD   UNIT=SYSDA,SPLIT=4
//SYSUT3    DD   UNIT=SYSDA,SPLIT=4
//SYSPUNCH  DD   DSNAME=&LOADSET,DISP=(MOD,PASS),
//               UNIT=SYSDA,SPACE=(TRK,(50,10))
//SYSIN     DD   DDNAME=SYSIN,DCB=BLKSIZE=80
//LKED      EXEC PGM=IEWL,PARM=(XREF,LIST,LET),COND=(9,LT,COB),
//               REGION=96K
//SYSLIN    DD   DSNAME=&LOADSET,DISP=(OLD,DELETE)
//         DD   DDNAME=SYSIN
//SYSLMOD   DD   DSNAME=&GODATA(RUN),DISP=(NEW,PASS),
//               UNIT=SYSDA,SPACE=(1024,(50,20,1))
//SYSLIB    DD   DSNAME=SYS1.COBLIB,DISP=(SHR,KEEP)
//SYSUT1    DD   UNIT=(SYSDA,SEP=(SYSLIN,SYSLMOD)),
//               SPACE=(1024,(50,20))
//SYSPRINT  DD   SYSOUT=A
//GO        EXEC PGM=*.LKED.SYSLMOD,COND=((9,LT,COB),(5,LT,LKED))
//SYSABEND  DD   SYSOUT=A
//SYSOUT    DD   SYSOUT=A,DCB=(,BLKSIZE=120,LRECL=120)
```

Note: You must provide a COB.SYSIN DD statement when you call the procedure.

Figure 25. COBCLG (Compile-Link Edit-Go)

COBOL F

```
//COB       EXEC PGM=IEQCBL00,PARM='DECK,NOLOAD',REGION=86K
//SYSPRINT  DD   SYSOUT=A
//SYSPUNCH  DD   SYSOUT=B
//SYSUT1    DD   UNIT=SYSDA,SPACE=(460,(700,100))
//SYSUT2    DD   UNIT=SYSDA,SPACE=(460,(700,100))
//SYSUT3    DD   UNIT=SYSDA,SPACE=(460,(700,100))
//SYSUT4    DD   UNIT=SYSDA,SPACE=(460,(700,100))
```

Note: You must provide a COB.SYSIN DD statement when you call the procedure.

Figure 26. COBFC (Compile)

```
//LKED      EXEC PGM=IEWL,PARM='LIST,XREF,LET',REGION=96K
//SYSLIN    DD   DDNAME=SYSIN
//SYSLMOD   DD   DSNAME=&GODATA(RUN),DISP=(NEW,PASS),UNIT=SYSDA,
//               SPACE=(1024,(50,20,1))
//SYSLIB    DD   DSNAME=SYS1.COBLIB,DISP=SHR
//SYSUT1    DD   UNIT=(SYSDA,SEP=(SYSLIN,SYSLMOD),
//               SPACE=(1024,(50,20))
//SYSPRINT  DD   SYSOUT=A
//GO        EXEC PGM=*.LKED.SYSLMOD,COND=(5,LT,LKED)
//SYSABEND  DD   SYSOUT=A
```

Note: You must provide a LKED.SYSIN DD statement when you call the procedure.

Figure 27. COBFLG (Link Edit-Go)

```
//COB       EXEC PGM=IEQCBL00,REGION=86K
//SYSPRINT  DD   SYSOUT=A
//SYSUT1    DD   UNIT=SYSDA,SPACE=(460,(700,100))
//SYSUT2    DD   UNIT=SUSDA,SPACE=(460,(700,100))
//SYSUT3    DD   UNIT=SYSDA,SPACE=(460,(700,100))
//SYSUT4    DD   UNIT=SYSDA,SPACE=(460,(700,100))
//SYSLIN    DD   DSNAME=&LOADSET,DISP=(MOD,PASS),UNIT=SYSDA,
//               SPACE=(80,(500,100))
//LKED      EXEC PGM=IEWL,PARM='LIST,XREF,LET',COND=(5,LT,COB),
//               REGION=96K
//SYSLIN    DD   DSNAME=&LOADSET,DISP=(OLD,DELETE)
//          DD   DDNAME=SYSIN
//SYSLMOD   DD   DSNAME=&GODATA(RUN),DISP=(NEW,PASS),UNIT=SYSDA,
//               SPACE=(1024,(50,20,1))
//SYSLIB    DD   DSNAME=SYS1.COBLIB,DISP=SHR
//SYSUT1    DD   UNIT=(SYSDA,SEP=(SYSLIN,SYSLMOD)),
//               SPACE=(1024,(50,20))
//SYSPRINT  DD   SYSOUT=A
//GO        EXEC PGM=*.LKED.SYSLMOD,COND=((5,LT,COB),(5,LT,LKED))
//SYSABEND  DD   SYSOUT=A
```

Note: You must provide a COB.SYSIN DD statement when you call the procedure.

Figure 28. COBFCLG (Compile-Link Edit-Go)

```
//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))
```

Note: You must provide a COB.SYSIN DD statement when you call the procedure.

Figure 29. COBUC (Compile)

```
//LKED     EXEC PGM=IEWL,PARM='LIST,XREF,LET',REGION=96K
//SYSLIN   DD   DSNAME=&&LOADSET
//SYSLMOD  DD   DSNAME=&&GOSET(RUN),DISP=(NEW,PASS),UNIT=SYSDA,
//              SPACE=(1024,(50,20,1))
//SYSLIB   DD   DSNAME=SYS1.COBLIB,DISP=SHR
//SYSUT1   DD   DSNAME=&&SYSUT1,UNIT=(SYSDA,SEP=(SYSLIN,SYSLMOD)),
//              SPACE=(1024,(50,20))
//SYSPRINT DD   SYSOUT=A
//GO       EXEC PGM=*.LKED.SYSLMOD,COND=(5,LT,LKED)
```

Note: You must provide a LKED.SYSIN DD statement when you call the procedure.

Figure 30. COBULG (Link Edit-Go)

```
//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,
//              SPACE=(80,(500,100))
//LKED     EXEC PGM=IEWL,PARM='LIST,XREF,LET',COND=(5,LT,COB),
//              REGION=96K
//SYSLIN   DD   DSNAME=&&LOADSET,DISP=(OLD,DELETE)
//         DD   DSNAME=SYSIN
//SYSLMOD  DD   DSNAME=&&GOSET(RUN),DISP=(NEW,PASS),UNIT=SYSDA,
//              SPACE=(1024,(50,20,1))
//SYSLIB   DD   DSNAME=SYS1.COBLIB,DISP=SHR
//SYSPRINT DD   SYSOUT=A
//GO       EXEC PGM=*.LKED.SYSLMOD,COND=((5,LT,COB),(5,LT,LKED))
```

Note: You must provide a COB.SYSIN DD statement when you call the procedure.

Figure 31. COBUCLG (Compile-Link Edit-Go)

```
//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),
//              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
```

Note: You must provide a COB.SYSIN DD statement when you call the procedure.

Figure 32. COBUCG (Compile-Load)

```
//FORT     EXEC PGM=IEJFAAA0,REGION=42K
//SYSPRINT DD   SYSOUT=A,DCB=BLKSIZE=121
//SYSPUNCH DD   SYSOUT=B,DCB=BLKSIZE=80
//SYSUT1   DD   DSN=&SYSUT1,UNIT=SYSSQ,SEP=SYSPUNCH,
//              SPACE=(904,(30,20))
//SYSUT2   DD   DSN=&SYSUT2,UNIT=SYSSQ,SEP=SYSUT1,
//              SPACE=(904,(30,20))
//SYSLIN   DD   UNIT=SYSSQ,SEP=SYSPUNCH,DSNAME=&LOADSET,
//              DISP=(MOD,PASS),SPACE=(80,(200,200))
```

Note: You must provide a FORT.SYSIN DD statement when you call the procedure.

Figure 33. FORTEC (Compile)

```
//FORT     EXEC PGM=IEJFAAA0,REGION=42K
//SYSPRINT DD   SYSOUT=A,DCB=BLKSIZE=121
//SYSPUNCH DD   SYSOUT=B,DCB=BLKSIZE=80
//SYSUT1   DD   UNIT=SYSSQ,SEP=SYSPUNCH,SPACE=(904,(30,20))
//SYSUT2   DD   UNIT=SYSSQ,SEP=SYSUT1,SPACE=(904,(30,20))
//SYSLIN   DD   UNIT=SYSSQ,SEP=SYSPUNCH,DSNAME=&LOADSET,
//              DISP=(MOD,PASS),SPACE=(80,(200,200),RLSE)
//LKED     EXEC PGM=IEWL,PARM=(XREF,LET,LIST,NCAL),COND=(4,LT,FORT),
//              REGION=96K
//SYSPRINT DD   SYSOUT=A,DCB=BLKSIZE=121
//SYSLIN   DD   DSNAME=&LOADSET,DISP=(OLD,DELETE),DCB=BLKSIZE=80
//         DD   DDNAME=SYSIN
//SYSLMOD  DD   DSNAME=SYS1.FORTLIB,DISP=OLD
//SYSUT1   DD   DSN=&SYSUT1,UNIT=SYSDA,SEP=SYSLMOD,
//              SPACE=(1024,(30,20))
```

Note: You must provide a FORT.SYSIN DD statement when you call the procedure.

Figure 34. FORTECL (Compile-Link Edit)

```
//LKED     EXEC PGM=IEWL,PARM=(XREF,LET,LIST),REGION=96K
//SYSPRINT DD   SYSOUT=A,DCB=BLKSIZE=121
//SYSLIB   DD   DSNAME=SYS1.FORTLIB,DISP=SHR
//SYSLIN   DD   DDNAME=SYSIN
//SYSLMOD  DD   DSNAME=&GOSET(MAIN),DISP=(NEW,PASS),UNIT=SYSDA,
//              SPACE=(1024,(50,20,1),RLSE)
//SYSUT1   DD   DSN=&SYSUT1,UNIT=SYSDA,SEP=(SYSLMOD,SYSLIB),
//              SPACE=(1024,(30,20)
//GO       EXEC PGM=*.LKED.SYSLMOD,COND=(4,LT,LKED)
//FT01F001 DD   DDNAME=SYSIN
//FT02F001 DD   SYSOUT=B
//FT03F001 DD   SYSOUT=A
```

Note: You must provide a LKED.SYSLIN DD statement when you call the procedure.

Figure 35. FORTELG (Link Edit-Go)

```
//FORT     DD   PGM=IEJFAAA0,REGION=42K
//SYSPRINT DD   SYSOUT=A,DCB=BLKSIZE=121
//SYSPUNCH DD   SYSOUT=B,DCB=BLKSIZE=80
//SYSUT1   DD   DSN=&SYSUT1,UNIT=SYSSQ,SEP=SYSPUNCH,
//              SPACE=(904,(30,20))
//SYSUT2   DD   DSN=&SYSUT2,UNIT=SYSSQ,SEP=SYSUT1,SPACE=(904,(30,20))
//SYSLIN   DD   UNIT=SYSSQ,SEP=SYSPUNCH,DSNAME=&LOADSET,
//              DISP=(MOD,PASS),SPACE=(80,(200,200),RLSE)
//LKED     EXEC PGM=IEWL,PARM=(XREF,LET,LIST),COND=(4,LT,FORT),
//              REGION=96K
//SYSPRINT DD   SYSOUT=A,DCB=BLKSIZE=121
//SYSLIB   DD   DSNAME=SYS1.FORTLIB,DISP=SHR
//SYSLIN   DD   DSNAME=&LOADSET,DISP=(OLD,DELETE),DCB=BLKSIZE=80
//        DD    DDNAME=SYSIN
//SYSLMOD  DD   DSNAME=&GOSET(MAIN),DISP=(NEW,PASS),UNIT=SYSDA,
//              SPACE=(1024,(50,20,1),RLSE)
//SYSUT1   DD   DSN=&SYSUT1,UNIT=SYSDA,SEP=(SYSLMOD,SYSLIB),
//              SPACE=(1024,(30,20))
//GO       EXEC PGM=*.LKED.SYSLMOD,COND=((4,LT,FORT),(4,LT,LKED))
//FT01F001 DD   DDNAME=SYSIN
//FT02F001 DD   SYSOUT=B
//FT03F001 DD   SYSOUT=A
```

Note: You must provide a FORT.SYSIN DD statement when you call the procedure.

Figure 36. FORTECLG (Compile-Link Edit-Go)

```
//FORT     EXEC PGM=IEYFORT,REGION=100K
//SYSPRINT DD   SYSOUT=A
//SYSPUNCH DD   SYSOUT=B
//SYSLIN   DD   DSNAME=&LOADSET,DISP=(MOD,PASS),UNIT=SYSSQ,
//              SPACE=(80,(200,100),RLSE),DCB=BLKSIZE=80
```

Note: You must provide a FORT.SYSIN DD statement when you call the procedure.

Figure 37. FORTGC (Compile)

```
//FORT     EXEC PGM=IEYFORT,REGION=100K
//SYSPRINT DD   SYSOUT=A
//SYSPUNCH DD   SYSOUT=B
//SYSLIN   DD   DSNAME=&LOADSET,DISP=(MOD,PASS),UNIT=SYSSQ,
//              SPACE=(80,(200,100),RLSE),DCB=BLKSIZE=80
//LKED     EXEC PGM=IEWL,REGION=96K,PARM=(XREF,LET,LIST),
//              COND=(4,LT,FORT)
//SYSLIB   DD   DSNAME=SYS1.FORTLIB,DISP=SHR
//SYSLMOD  DD   DSNAME=&GOSET(MAIN),DISP=(NEW,PASS),UNIT=SYSDA,
//              SPACE=(1024,(20,10,1),RLSE),DCB=BLKSIZE=1024
//SYSPRINT DD   SYSOUT=A
//SYSUT1   DD   DSNAME=&SYSUT1,UNIT=SYSDA,SPACE=(1024,(20,10),RLSE),
//              DCB=BLKSIZE=1024
//SYSLIN   DD   DSNAME=&LOADSET,DISP=(OLD,DELETE)
//         DD   DDNAME=SYSIN
```

Note: You must provide a FORT.SYSIN DD statement when you call the procedure.

Figure 38. FORTGCL (Compile-Link Edit)

```
//LKED     EXEC PGM=IEWL,REGION=96K,PARM=(XREF,LET,LIST)
//SYSLIB   DD   DSNAME=SYS1.FORTLIB,DISP=SHR
//SYSLMOD  DD   DSNAME=&GOSET(MAIN),DISP=(NEW,PASS),UNIT=SYSDA,
//              SPACE=(1024,(20,10,1),RLSE),DCB=BLKSIZE=1024
//SYSPRINT DD   SYSOUT=A
//SYSUT1   DD   DSNAME=&SYSUT1,UNIT=SYSDA,SPACE=(1024,(20,10),RLSE),
//              DCB=BLKSIZE=1024
//SYSLIN   DD   DDNAME=SYSIN
//GO       EXEC PGM=*.LKED.SYSLMOD,COND=(4,LT,LKED)
//FT05F001 DD   DDNAME=SYSIN
//FT06F001 DD   SYSOUT=A
//FT07F001 DD   SYSOUT=B
```

Note: You must provide a LKED.SYSIN DD statement when you call the procedure.

Figure 39. FORTGLG (Link Edit-Go)

```
//FORT     EXEC PGM=IEYFORT,REGION=100K
//SYSPRINT DD   SYSOUT=A
//SYSPUNCH DD   SYSOUT=B
//SYSLIN   DD   DSNAME=&LOADSET,DISP=(MOD,PASS),UNIT=SYSSQ,
//              SPACE=(80,(200,100),RLSE),DCB=BLKSIZE=80
//LKED     EXEC PGM=IEWL,REGION=96K,PARM=(XREF,LET,LIST),
//              COND=(4,LT,FORT)
//SYSLIB   DD   DSNAME=SYS1.FORTLIB,DISP=SHR
//SYSLMOD  DD   DSNAME=&GOSET(MAIN),DISP=(NEW,PASS),UNIT=SYSDA,
//              SPACE=(1024,(20,10,1),RLSE),DCB=BLKSIZE=1024
//SYSPRINT DD   SYSOUT=A
//SYSUT1   DD   DSNAME=&SYSUT1,UNIT=SYSDA,SPACE=(1024,(20,10),RLSE),
//              DCB=BLKSIZE=1024
//SYSLIN   DD   DSNAME=&LOADSET,DISP=(OLD,DELETE)
//         DD   DDNAME=SYSIN
//GO       EXEC PGM=*.LKED.SYSLMOD,COND=((4,LT,FORT)(4,LT,LKED))
//FT05F001 DD   DDNAME=SYSIN
//FT06F001 DD   SYSOUT=A
//FT07F001 DD   SYSOUT=B
```

Note: You must provide a FORT.SYSIN DD statement when you call the procedure.

Figure 40. FORTGCLG (Compile-Link Edit-Go)

```
//FORT     EXEC PGM=IEKAA00,REGION=228K
//SYSPRINT DD   SYSOUT=A
//SYSPUNCH DD   SYSOUT=B
//SYSLIN   DD   DSNAME=&LOADSET,UNIT=SYSSQ,DISP=(MOD,PASS),
//              SPACE=(400,(200,50),RLSE)
```

Note: You must provide a FORT.SYSIN DD statement when you call the procedure.

Figure 41. FORTHC (Compile)

```
//FORT     EXEC PGM=IEKAA00,REGION=228K
//SYSPRINT DD   SYSOUT=A
//SYSPUNCH DD   SYSOUT=B
//SYSLIN   DD   DSNAME=&LOADSET,UNIT=SYSSQ,DISP=(MOD,PASS),
//              SPACE=(400,(200,50),RLSE)
//LKED     EXEC PGM=IEWL,REGION=54K,PARM=(MAP,LET,LIST),
//              COND=(4,LT,FORT)
//SYSLIB   DD   DSNAME=SYS1.FORTLIB,DISP=SHR
//SYSPRINT DD   SYSOUT=A
//SYSLMOD  DD   DSNAME=&GOSET(MAIN),UNIT=SYSDA,DISP=(,PASS),
//              SPACE=(3072)(30,10,1),RLSE)
//SYSLIN   DD   DSNAME=&LOADSET,DISP=(OLD,DELETE)
//         DD   DDNAME=SYSIN
//SYSUT1   DD   DSNAME=&SYSUT1,UNIT=SYSDA,SPACE=(1024,(200,20)),
//              SEP=SYSLMOD
```

Note: You must provide a FORT.SYSIN DD statement when you call the procedure.

Figure 42. FORTHCL (Compile-Link Edit)

```
//LKED     EXEC PGM=IEWL,REGION=54K,PARM=(MAP,LET,LIST)
//SYSLIB   DD   DSNAME=SYS1.FORTLIB,DISP=SHR
//SYSPRINT DD   SYSOUT=A
//SYSLIN   DD   DDNAME=SYSIN
//SYSLMOD  DD   DSNAME=&GOSET(MAIN),UNIT=STSDA,DISP=(,PASS),
//              SPACE=(3072,(30,10,1),RLSE)
//SYSUT1   DD   DSNAME=&SYSUT1,UNIT=SYSDA,SPACE=(1024,(200,20)),
//              SEP=SYSLMOD
//GO       EXEC PGM=*.LKED.SYSLMOD,COND=(4,LT,LKED)
//FT05F001 DD   DDNAME=SYSIN
//FT06F001 DD   SYSOUT=A
//FT07F001 DD   SYSOUT=B
```

Note: You must provide a LKED.SYSIN DD statement when you call the procedure.

Figure 43. FORTHLG (Link Edit-Go)

```
//FORT      EXEC PGM=IEKAA00,REGION=228K
//SYSPRINT  DD   SYSOUT=A
//SYSPUNCH  DD   SYSOUT=B
//SYSLIN    DD   DSNAME=&LOADSET,UNIT=SYSSQ,DISP=(MOD,PASS),
//               SPACE=(400,(200,50),RLSE)
//LKED      EXEC PGM=IEWL,REGION=54K,PARM=(MAP,LET,LIST),
//               COND=(4,LT,FORT)
//SYSLIB    DD   DSNAME=SYS1.FORTLIB,DISP=SHR
//SYSPRINT  DD   SYSOUT=A
//SYSLMOD   DD   DSNAME=&GOSET(MAIN),UNIT=SYSDA,DISP=(,PASS),
//               SPACE=(3072,(30,10,1),RLSE)
//SYSUT1    DD   DSNAME=&SYSUT1,UNIT=SYSDA,SPACE=(1024,(200,20)),
//               SEP=SYSLMOD
//SYSLIN    DD   DSNAME=&LOADSET(MAIN),DISP=(OLD,DELETE)
//         DD   DDNAME=SYSIN
//GO        EXEC PGM=*.LKED.SYSLMOD,COND=((4,LT,FORT),(4,LT,LKED))
//FT05F001  DD   DDNAME=SYSIN
//FT06F001  DD   SYSOUT=A
//FT07F001  DD   SYSOUT=B
```

Note: You must provide a FORT.SYSIN DD statement when you call the procedure.

Figure 44. FORTHCLG (Compile-Link Edit-Go)

```
//PL1D     EXEC PGM=IEMAA,PARM='DECK,NOLOAD',REGION=52K
//SYSPRINT DD   SYSOUT=A
//SYSPUNCH DD   SYSOUT=B
//SYSUT3   DD   DSNAME=&&SYSUT3,UNIT=SYSSQ,SPACE=(80,(250,250)),
//              SEP=SYSPRINT
//SYSUT1   DD   DSNAME=&&SYSUT1,UNIT=SYSDA,SPACE=(1024,(60,60),
//              CONTIG),SEP=(SYSUT3,SYSPRINT,SYSPUNCH)
```

Note: You must provide a PL1D.SYSIN DD statement when you call the procedure.

Figure 45. PL1DFC (Compile With Deck Output)

```
//PL1L     EXEC PGM=IEMAA,PARM='LOAD,NODECK',REGION=52K
//SYSPRINT DD   SYSOUT=A
//SYSLIN   DD   DSNAME=&&LOADSET,DISP=(MOD,PASS),UNIT=SYSSQ,
//              SPACE=(80,(250,100))
//SYSUT3   DD   DSNAME=&&SYSUT3,UNIT=SYSSQ,SPACE=(80,(250,250)),
//              SEP=SYSPRINT
//SYSUT1   DD   DSNAME=&&SYSUT1,UNIT=SYSDA,SPACE=(1024,(60,60),
//              CONTIG),SEP=(SYSUT3,SYSPRINT,SYSLIN)
```

Note: You must provide a PL1L.SYSIN DD statement when you call the procedure.

Figure 46. FL1LFC (Compile With Object Module Output

```
//PL1L     EXEC PGM=IEMAA,PARM='LOAD,NODECK',REGION=52K
//SYSPRINT DD   SYSOUT=A
//SYSLIN   DD   DSNAME,&&LOADSET,DISP=(MOD,PASS),UNIT=SYSSQ,
//              SPACE=(80,(250,100))
//SYSUT3   DD   DSNAME=&&SYSUT3,UNIT=SYSSQ,SPACE=(80,(250,250)),
//              SEP=SYSPRINT
//SYSUT1   DD   DSNAME=&&SYSUT1,UNIT=SYSDA,SPACE=(1024,(60,60),,
//              CONTIG),SEP=(SYSUT3,SYSPRINT,SYSLIN)
//LKED     EXEC PGM=IEWL,PARM='XREF,LIST',COND=(9,LT,PL1L),
//              REGION=96K
//SYSLIB   DD   DSNAME=SYS1.PL1LIB,DISP=SHR
//SYSLMOD  DD   DSNAME=&&GOSET(GO),DISP=(MOD,PASS),UNIT=SYSDA,
//              SPACE=(1024,(50,20,1),RLSE)
//SYSUT1   DD   DSNAME=&&SYSUT1,UNIT=SYSDA,SEP=(SYSLMOD,SYSLIB),
//              SPACE=(1024,(200,20))
//SYSPRINT DD   SYSOUT=A
//SYSLIN   DD   DSNAME=&&LOADSET,DISP=(OLD,DELETE)
//         DD   DDNAME=SYSIN
```

Note: You must provide a PL1L.SYSIN DD statement when you call the procedure.

Figure 47. PL1LFCL (Compile-Link Edit)

```
//LKED     EXEC PGM=IEWL,PARM='XREF,LIST',REGION=96K
//SYSLIB   DD   DSNAME=SYS1.PL1LIB,DISP=SHR
//SYSLMOD  DD   DSNAME=&&GOSET(GO),DISP=(MOD,PASS),
//              UNIT=SYSDA,SPACE=(1024,(50,20,1),RLSE)
//SYSUT1   DD   DSNAME=&&SYSUT1,UNIT=SYSDA,SEP=(SYSLMOD,SYSLIB),
//              SPACE=(1024,(200,20))
//SYSPRINT DD   SYSOUT=A
//SYSLIN   DD   DDNAME=SYSIN
//GO       EXEC PGM=*.LKED.SYSLMOD,COND=(9,LT,LKED)
//SYSPRINT DD   SYSOUT=A
```

Note: You must provide a LKED.SYSIN DD statement when you call the procedure.

Figure 48. PL1LFLG (Link Edit-Go)

```
//PL1L     EXEC PGM=IEMAA,PARM='LOAD,NODECK'REGION=52K
//SYSPRINT DD   SYSOUT=A
//SYSLIN   DD   DSNAME=&&LOADSET,DISP=(MOD,PASS),
//              UNIT=SYSSQ,SPACE=(80,(250,100))
//SYSUT3   DD   DSNAME=&&SYSUT3,UNIT=SYSSQ,SPACE=(80,(250,250)),
//              SEP=SYSPRINT
//SYSUT1   DD   DSNAME=&&SYSUT1,UNIT=SYSDA,SPACE=(1024,(60,60),,
//              CONTIG),SEP=(SYSUT3,SYSPRINT,SYSLIN)
//LKED     EXEC PGM=IEWL,PARM='XREF,LIST',
//              COND=(9,LT,PL1L),REGION=96K
//SYSLIB   DD   DSNAME=SYSU.PL1LIB,DISP=SHR
//SYSLMOD  DD   DSNAME=&&GOSET(GO),DI
//              SP=(MOD,PASS),UNIT=SYSDA,SPACE=(1024,(50,20,1),RLSE)
//SYSUT1   DD   DSNAME=&&SYSUT1,UNIT=SYSDA,SEP=(SYSLMOD,SYSLIB),
//              SPACE=(1024,(200,20))
//SYSPRINT DD   SYSOUT=A
//SYSLIN   DD   DSNAME=&&LOADSET,DISP=(OLD,DELETE)
//         DD   DDNAME=SYSIN
//GO       EXEC PGM=*.LKED.SYSLMOD,COND=((9,LT,LKED),(9,LT,PL1L))
//SYSPRINT DD   SYSOUT=A
```

Note: You must provide a PL1L.SYSIN DD statement when you call the procedure.

Figure 49. PL1LFCLG (Compile-Link Edit-Go)

```
//PL1L     EXEC PGM=IEMAA,PARM='LOAD,NODECK',REGION=52K
//SYSPRINT DD   SYSOUT=A
//SYSLIN   DD   DSNAME=&&LOADSET,DISP=(MOD,PASS),UNIT=SYSSQ,
//              SPACE=(80,(250,100))
//SYSUT3   DD   DSNAME=&&SYSUT3,UNIT=SYSSQ,SPACE=(80,(250,250)),
//              SEP=SYSPRINT
//SYSUT1   DD   DSNAME=&&SYSUT1,UNIT=SYSDA,SPACE=(1024,(60,60)),,
//              CONTIG),SEP=(SYSUT3,SYSPRINT,SYSLIN)
//GO       EXEC PGM=LOADER,PARM='MAP,PRINT',REGION=96K,
//              COND=(9,LT,PL1L)
//SYSLIB   DD   DSNAME=SYS1.PL1LIB,DISP=SHR
//SYSLIN   DD   DSNAME=&&LOADSET,DISP=(OLD,DELETE)
//SYSLOUT  DD   SYSOUT=A
//SYSPRINT DD   SYSOUT=A
```

Note: You must provide a PL1.SYSIN DD statement when you call the procedure.

Figure 50. PL1LFCG (Compile-Load)

```
//GO       EXEC PGM=LOADER,PARM='MAP,PRINT',REGION=96K
//SYSLIB   DD   DSNAME=SYS1.PL1LIB,DISP=SHR
//SYSOUT   DD   SYSOUT=A
//SYSPRINT DD   SYSOUT=A
```

Note: You must provide a GO.SYSIN DD statement when you call the procedure.

Figure 51. PL1LFG (Load)

Examples

1. The following example shows how to call the ALGOFCLG cataloged procedure. The source program is in the input stream. No changes are made to the procedure.

```
//EX1          JOB
//STEP         EXEC ALGOFCLG
//ALGOL.SYSIN  DD   *
             source program
/*
```

2. The following example shows how to call the ASMEC cataloged procedure. You want to change the SYSPUNCH DD statement to define a kept data set on tape named OBJMOD. The source program is in the input stream.

```
//EX2          JOB
//               EXEC PROC=ASMEC
//ASM.SYSPUNCH DD   UNIT=2400,DSNAME=OBJMOD,DISP=(,KEEP)
//ASM.SYSIN    DD   *
             source program
/*
```

3. The following example shows how to call the ASMFCL cataloged procedure. You want to change the SYSLMOD DD statement to define member #3 of a cataloged data set named SYST.LIB. The source program is in the input stream.

```
//EX3          JOB
//               EXEC ASMFCL
//ASM.SYSIN    DD   *
             source program
/*
//LKED.SYSLMOD DD   DSNAME=SYST.LIB(#3),DISP=(,CATLG)
```

4. The following example shows how to call the COBELG cataloged procedure. You want to change the REGION parameter of the LKED EXEC statement to REGION=120K, and to change the PARM parameter to show MAP rather than XREF. You also want to add a TIME=5 parameter to the GO EXEC statement. The object program is on a 2501 card reader. You are adding two DD statements to the GO step.

```
//EX4          JOB
//CALL         EXEC  COBELG,REGION.LKED=120K,PARM.LKED=(MAP,
//                   LIST,LET),TIME.GO=5
//LKED.SYSIN   DD    UNIT=2501
//GO.UTIL      DD    UNIT=2314,SPACE=(CYL,(20,5))
//GO.INPUT     DD    *
       input data
/*
```

5. The following example shows two steps of a job. The first step calls the FORTGCL cataloged procedure. A value of OBJECT is assigned &LOADSET symbolic parameter and the DISP parameter is changed to DISP=(OLD,KEEP) in the SYSLIN DD statement of the LKED step. The second step of the job executes the program produced by the linkage editor in the data set defined by the SYSLMOD DD statement.

   Note: IBM-supplied cataloged procedures do not use symbolic parameters. However, in this case, you can take advantage of the fact that &LOADSET is in symbolic parameter form to assign it a different value with the EXEC statement.

```
//EX5          JOB
//STEP1        EXEC  FORTGCL,LOADSET=OBJECT
//FORT.SYSIN   DD    *
       source program
/*
//LKED.SYSLIN DD     DISP=(OLD,KEEP)
//STEP2        EXEC  PGM=*.STEP1.LKED.SYSLMOD
.
.
.
```

6. The following example shows how to call the PL1LFCLG cataloged procedure. You want the object module to be on a 2400 unit. Therefore you must change the UNIT parameter of the SYSLIN DD statement and nullify the SPACE parameter. You also want to nullify the COND parameter of the GO step.

```
//EX6          JOB
//             EXEC  PROC=PL1LFCLG,PARM.GO=
//PL1L.SYSLIN DD     UNIT=2400,SPACE=
//PL1L.SYSIN   DD    *
       source program
/*
```

7. The following example shows how to call the COBUCG cataloged procedure. The source program is in the input stream. You are adding two DD statements to the GO step that are required by your loaded program for execution.

```
//EX7          JOB
//             EXEC  COBUCG
//COB.SYSIN    DD    *
                    source program
/*
//GO.DATA      DD    DSNAME=RECORDS,DISP=OLD
//GO.OUTPUT    DD    SYSOUT=F
```

Appendix A: Indexed Sequential Data Sets

Indexed sequential (ISAM) data sets are created, retrieved, and overwritten using special subsets of DD statement parameters and subparameters. Each data set can occupy up to three different areas of space:

1. Prime area -- This area contains data and related track indexes. It exists for all indexed sequential data sets.

2. Overflow area -- This area contains overflow from the prime area when new data is added. It is optional.

3. Index area -- This area contains master and cylinder indexes associated with the data set. It exists for any indexed sequential data set that has a prime area occupying more than one cylinder. Indexed sequential data sets must reside on direct access volumes.

The data set can reside on more than one volume and the device types of the volumes may in some cases differ.

Creating an Indexed Sequential Data Set

One to three DD statements can be used to define a new indexed sequential data set. There are four types of indexed sequential data sets:

- Separate index, prime and overflow areas -- The DD statements that define each area must be coded in the following order:

  1. Index area
  2. Prime area
  3. Overflow area

- Separate index and prime areas -- No overflow area. The DD statements that define each area must be coded in the following order:

  1. Index area
  2. Prime area

- Separate prime and overflow areas -- The index area is part of the prime area. The DD statements that define each area must be coded in the following order:

  1. Prime area
  2. Overflow area

- Prime Area Only -- The index area is part of the prime area. Only the DD statement that defines the prime area must be coded.

When more than one DD statement is used to define the data set, assign a ddname only to the first DD statement; the name field of the other statements must be blank. In other words, concatenate the DD statements.

Table 59 shows the parameters required to define each area. The parameters for the three areas are discussed together in the following paragraphs. For further information on the parameters refer to "Direct Access Devices" in the section "Creating a New Data Set."

Table 59. Parameters for Creating an Indexed Sequential Data Set (Part 1 of 2)

Area	Parameter type	Parameter	Comments
Index	Data set information	DSNAME	Required. You must code DSNAME=dsname(INDEX) or DSNAME=&dsname (INDEX)
		DISP	Required. You must code the same value as in the DD statement for the prime area.
	Location of the data set	UNIT	Required unless you request (with the VOLUME parameter) the same volume used for an earlier data set. Only one unit can be requested.
		VOLUME	Required if you want a specific volume.
		LABEL	Optional.
	Size of the Data Set	SPACE	Required.
	Data Attributes	DCB	Required.
	Special Processing Option	SEP AFF	Either parameter can be used.
Prime	Data Set information	DSNAME	Required. You must code: DSNAME=dsname(PRIME), DSNAME=&dsname(PRIME) DSNAME=dsname, or DSNAME=&dsname
		DISP	Required.
	Location of the data set	UNIT	Required unless you request (with the VOLUME parameter) the same volume used for an earlier data set.
		VOLUME	Required if you want a specific volume or multiple volumes.
		LABEL	Optional.

Table 59. Parameters for Creating an Indexed Sequential Data Set (Part 2 of 2)

Area	Parameter type	Parameter	Comments
Prime (Cont.)	Size of the data set	SPACE	Required. If you do not include a DD statement for the index area you can either specify an "index quantity" or let the system assign the index area.
	Data Attributes	DCB	Required.
	Special Processing Option	SEP	Either parameter can be used.
		AFF	
Overflow	Data Set Information	DSNAME	Required. You must code DSNAME=dsname (OVFLOW) or DSNAME=&dsname(OVFLOW).
		DISP	Required. You must code the same value as in the DD statement for the prime area.
	Location of the data set	UNIT	Required unless you request (with the VOLUME parameter) the same volume used for an earlier data set.
		VOLUME	Required if you want a specific volume.
		LABEL	Optional.
	Size of the data set	SPACE	Required.
	Data Attributes	DCB	Required.
	Special Processing Option	SEP	Either parameter can be used.
		AFF	

Data Set Information

The DSNAME and DISP parameters are required on any DD statement used to define an indexed sequential data set.

DSNAME Parameter

The DSNAME parameter is required on the DD statement that defines each of the areas. The general format is:

```
DSNAME={dsname(area) }
       {&dsname(area)}
```

Replace the term "dsname" with the fully qualified, unqualified, or temporary name of the data set. Whichever name you use, you must use the same name of the DD statement for each area. The term "area" indicates which area you are defining as decribed below.

Index area: The format of the DSNAME parameter for the index area is:

```
DSNAME={dsname(INDEX) }
       {&dsname(INDEX)}
```

For example,

```
DSNAME=SET.TWO(INDEX)
DSNAME=AREA(INDEX)
DSNAME=&TEMP(INDEX)
```

Prime area: The format of the DSNAME parameter for the prime area is:

```
DSNAME={dsname(PRIME) }
       {&dsname(PRIME)}
```

For example,

```
DSNAME=SET.TWO(PRIME)
DSNAME=AREA(PRIME)
DSNAME=&TEMP(PRIME)
```

If you are using only one DD statement to define the data set, you may omit the term (PRIME). For example:

```
DSNAME=SET.TWO
DSNAME=AREA
DSNAME=&TEMP
```

Overflow area: The format of the DSNAME parameter for the overflow area is:

```
DSNAME={dsname(OVFLOW) }
       {&dsname(OVFLOW)}
```

For example,

```
DSNAME=SET.TWO(OVFLOW)
DSNAME=AREA(OVFLOW)
DSNAME=&TEMP(OVFLOW)
```

Location of the Data Set

The format of the DISP parameter is:

```
DISP=(NEW[,DELETE])
          [,KEEP  ]
```

You may omit the DISP parameter if you are creating and deleting the data set in the same job step.

Index area: You must specify the same value you specify for prime area.

Prime area: If you have separate DD statements for an index or overflow area, or both, you can only specify

DISP=(,KEEP)

If you are defining the data set with only one DD statement area you can specify

DISP=(,CATLG) or DISP=(,PASS)

If the data set was defined on more than one DD statement and the volumes on which the data set resides corresponds to the same device type, you can use the IEHPROGM utility program to catalog the data set. Refer to "The IEHPROGM program" in the publication IBM System/360 Operating System: Utilities for details.

Overflow area: You must specify the same value you specify for the prime area.

LOCATION OF THE DATA SET

The location of each area of the data set is given by the UNIT and VOLUME parameter. The LABEL parameter is optional.

UNIT Parameter

The UNIT parameter indicates which kind of direct access device you want. The format of the UNIT parameter is:

```
UNIT=({unit address}[,unitcount])
      {device type }[,P        ]
      {groupname   }
```

The first positional parameter identifies the direct access device you want to use for the area.

unit address
: is the actual machine address of the device. For example UNIT=190

device type
: is the type of device. You can specify a 2301, 2302, 2303, 2311, 2314, 2321. For example, UNIT=2311

Appendix

groupname
is the name of a collection of devices. For example, UNIT=DISK The second positional parameter can be used only for the prime area when more than one volume are requested in the VOLUME parameter. You can either request a number of devices or parallel mounting.

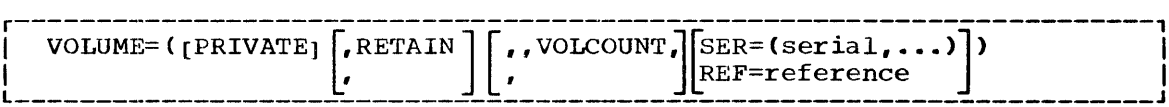
Index area: You can request only one device. It does not have to be the same kind of device as the one used for the prime area. For example, the index area could be on a 2301 and the prime area on a 2314.

Prime area: If there are separate DD statements defining the prime and index areas, you must request the same number of direct access devices for the prime area as there are volumes specified in the VOLUME parameter.

Overflow area: You can request only one device. It does not have to be the same kind of device as the one used for the prime area. For example, the prime area could be on a 2314 and the overflow on a 2311.

VOLUME Parameter

The VOLUME parameter for each of the areas is used in the same manner as for other data sets on direct access devices. The format of the VOLUME parameter is:



The volcount parameter and multiple serial numbers can only be used for the prime area.

Index area: You can make a specific or nonspecific request for any public or private volume. You can request only one volume. It can be one of the volumes used for the prime area, or the volume used for the overflow area.

Prime area: You can request multiple volumes even though a secondary quantity is not allowed in the SPACE parameter for an indexed sequential data set. The first, or only, volume you request cannot be the system residence volume. (The system residence volume is the volume that contains the IPL program.)

You can only request one volume if the prime and index areas are defined in the same DD statement. You define the index area in the same DD statement by specifying an "index quantity" in the SPACE parameter of the DD statement that defines the prime area and by omitting the DD statement for the index area.

Overflow area: You can make a specific or nonspecific request for any public or private volume. You can request only one volume. It can be one of the volumes used for the prime area, or the volume used for the index area.

Note: If you request nonspecific volumes and use more than one DD statement to define the data set, you will not be able to retrieve the data set in another step of the same job. (See the description of the VOLUME parameter in "Retrieving or Extending an Indexed Sequential Data Set.")

LABEL Parameter

The LABEL parameter need only be coded to specify password protection (PASSWORD) or a retention period (EXPDT or RETPD). You must specify the same parameter for each area defined.

Size of the Data Set

You must use the SPACE parameter to indicate the size of each area of the data set. The SPACE parameter lets you request space for your data set in one of two ways:

1. You can request a number of cylinders and let the system assign them specific tracks, or

2. You can request specific tracks.

If more than one DD statement is used to define the data set, all must request space using the same technique.

SPACE Parameter-The System Assigns Tracks

The format of the SPACE parameter is:

```
SPACE=(CYL,(primary quantity[,,index])[,,CONTIG])
```

The space must be requested in units of cylinders. Replace "primary quantity" with the number of cylinders you want. The "index" can only be specified for the prime area as described below. You may request contiguous space with the CONTIG parameter.

Index area: If you code CONTIG for the index area, you must also code it for the other areas.

Prime area: If you code CONTIG for the prime area, you must also code it for the other areas.

If you request more than one volume with the VOLUME parameter, each volume is assigned the number of cylinders requested in the primary quantity. (A secondary quantity is not allowed for indexed sequential data sets.)

If you do not define an index area with a separate DD statement, you can choose to have an index area embedded in the prime area or placed at the end of the prime area.

- Embedded in the prime area -- You must specify the number of cylinders required for the index area with the "index" subparameter. For example, if you want a primary allocation of 10 cylinders and an index area of two cylinders code SPACE=(CYL,(10,,2)). If you use an index quantity the prime area must reside on only one volume.

- End of the Prime Area -- You must increase the primary quantity by the number of cylinders required for the index area. For example, if you want a primary allocation of five cylinders and an index area of one cylinder, code

  SPACE=(CYL,6)

Overflow area: If you code CONTIG for the overflow area, you must also code it for the other areas.

SPACE Parameter-Requesting Specific Tracks

The format of the SPACE parameter is:

```
SPACE=(ABSTR,(primary quantity,address,index ))
```

The number of tracks you request must be equal to one or more whole cylinders. The address of the beginning track must correspond with the first track of a cylinder (other than the first cylinder on the volume). The "index" can only be specified for the prime area as specified below.

Prime Area: If you request more than one volume with the VOLUME parameter, space is allocated for the prime area beginning at the specified address and continuing through the volume and onto the next volume until the request is satisfied. This can only be done if the volume table of contents (VTOC) of the second and all succeeding volumes is contained within the first cylinder of each volume.

If you do not define an index area with a separate DD statement, you can choose to have an index area embedded in the prime area or placed at the end of the prime area.

- Embedded in the prime area -- You must specify the number of tracks required for the index area with the "index" subparameter. The number of tracks must be equal to one or more cylinders. If you use an index quantity, the prime area must reside on only one volume.

- End of the prime area -- You must increase the primary quantity by the number of tracks required for the index area. The total number must be equal to one or more cylinders.

Data Attributes

The DCB parameter must be coded on every DD statement that defines an indexed sequential data set. You must code DSORG=IS or DSORG=ISU. Other DCB subparameters can be coded as required by the language you are using. Indexed sequential data sets can only be used with the assembler, COBOL and PL/1. Table 60 shows the DCB subparameters used for each language. Underscored items are those default values selected if you omit the subparameter. Default values are not shown where the attribute can be specified either in your program or in the DD statement. If values for a given parameter are not shown, it is either specified in your source program or given a fixed value by the compiler. A glossary of DCB subparameter is given in Appendix B. Code only those parameters that apply to your compiler as shown in Table 60.

When more than one DD statement is used to define the data set, code all the DCB subparameters on the first DD statement. Code DCB=*.ddname on the remaining statement or statements; ddname is the name given to the first DD statement. (You can repeat the complete DCB parameter in each of the remaining DD statements, but the backward reference saves you time and eliminates the possibility of error.)

Table 60. DCB Subparameters for Creating an Indexed Sequential Data Set

DCB Subparameter	Assembler	COBOL E	COBOL F	ANS COBOL [2]	PL/I F
BFALN=	F or D [1]				
BLKSIZE=	number of bytes [1]		number of bytes [1]	number of bytes	number of bytes [1]
BUFNO=	number of buffers [1]	number of buffers [1]	number of buffers [1]	number of buffers [1]	number of buffers [1]
CYLOFL=	number of tracks [1]		number of tracks [1]	number of tracks	number of tracks
DSORG=	IS or ISU	IS	IS	IS	IS
HIARCHY=	0 or 1 [1]				
KEYLEN=	number of bytes [1]				number of bytes
LRECL=	number of bytes [1]				number of bytes [1]
NTM=	number of tracks [1]		number of tracks	number of tracks	number of tracks
OPTCD=	[W][M][Y][I][L][R] [1]		[W][M][Y][I][L][R]	[W][M][Y][I][L][R]	[W][M][Y][I][L]
RECFM=	V[B] or F[B] [1]				V[B] or F[B] [1]
RKP=	byte number [1]				byte number

[1] This function can be specified in your program rather than in the DD statement.

[2] American National Standard COBOL.

Retrieving or Extending an Indexed Sequential Data Set

A maximum of three DD statements are needed to retrieve or extend an indexed sequential (ISAM) data set. The DD statements are coded in the following order:

1. First DD statement - defines the indexed area
2. Second DD statement - defines the prime area
3. Third DD statement - defines the overflow area

Only the second DD statement is required. The first DD statement is not needed if either the index area resides on a volume of the same type as the prime area or if the index area is part of the prime area. The third DD statement is not needed if either the overflow area resides on a volume of the same type as the prime area or if there is no overflow area. When the first or third DD statements, or both, are not needed the definition of the index or overflow areas are included in the DD statement for the prime area.

Table 61 shows the parameters required for each DD statement. The parameters for all three DD statements are discribed together in the following paragraphs.

Table 61. Parameters for Retrieving or Extending an Indexed Sequential Data Set (Part 1 of 2)

Area	Parameter Type	Parameter	Comments
Index (used only if index area not on same device type as prime area) (First DD statement)	Data Set Information	DSNAME	Required. You must code the same value as in second DD statement.
		DISP	Required. You must code the same value as in second DD statement.
	Location of the data set	UNIT	Required.
		VOLUME	Required.
	Data Attributes	DCB	Required.
Prime and Overflow; or Index, Prime, and Overflow; or Indexed and Prime (required) (Second DD statement)	Data Set Information	DSNAME	Required.
		DISP	Required. Specifies whether you are retrieving the data set.
	Location of the data set	UNIT	Required. Unless it is a passed data set with all three area on one volume.
		VOLUME	Same requirement as UNIT. If used, code volumes in order they were defined.
	Data Attributes	DCB	Required.

Table 61. Parameters for Retrieving or Extending an Indexed Sequential Data Set (Part 2 of 2)

Area	Parameter Type	Parameter	Comments
Overflow (used only if overflow area not on same device type as prime area) (Third DD statement)	Data Set Information	DSNAME	Required. You must code the same value as in second DD statement
		DISP	Required. You must code the same value as in the second DD statement
	Location of the data set	UNIT	Required.
		VOLUME	Required.
	Data Attributes	DCB	Required.

Data Set Information

The DSNAME and DISP parameters are required.

DSNAME Parameter

The format of the DSNAME parameter is:

```
DSNAME={dsname                          }
       {&dsname                         }
       {*.stepname.ddname               }
       {*.stepname.procstepname.ddname  }
```

Identify the data set by its name, but do not include the term INDEX, PRIME or OVFLOW. If the data set was passed from a previous step, you can identify it by a backward reference as long as it was dh only one DD statement.

First DD statement: You must code the DSNAME parameter exactly as you code it in the second DD statement. You must not code the term INDEX.

Second DD statement: You must not code the term PRIME or OVFLOW.

Third DD Statement: You must not code the term OVFLOW.

DISP Parameter

The format of the DISP parameter is:

```
DISP=({SHR}[,PASS  ]
      {OLD}[,DELETE]
      {MOD}[,KEEP  ]
           [,CATLG ]
           [,UNCATLG])
```

Appendix

You must code either SHR, OLD or MOD. SHR specifies that you are retrieving the data set and that it can be shared with other jobs. OLD specifies that you are retrieving the data set. MOD specifies that you are extending the data set. You can, optionally, assign a disposition as the second subparameter. The same restrictions for PASS and CATLG apply as when you created the data set.

First DD Statement: You must code the DISP parameter exactly as you code it in the second DD statement. This first DD statement is used when the index area resides on a different device type from the prime area.

Third DD Statement: You must code the DISP parameter exactly as you coded it in the second DD statement. This third DD statement is used when the overflow area resides on a volume of a different device type from the prime area.

Location of the Data Set

You must use the UNIT and VOLUME parameter.unless the data set was passed and had all three areas on the same volume, or unless the data set was cataloged.

UNIT Parameter

The format of the UNIT parameter is:

```
UNIT=( { unit address } [ ,unit count ] )
       { device type  }
       { groupname    }  [ ,P          ]
```

The first positional parameter identifies the direct access device you want to use for the data set. The second positional parameter cannot be used in the first DD statement.

First DD statement: The device you specify must not be of the same type as the one specified in the second DD statement. (If they are of the same type, you do not need the first DD statement.) You can only request one device because the index area can only reside on one volume.

Second DD Statement: You must request the same number of devices as there are volumes specified in the VOLUME parameter. If you request parallel mounting you do not have to count the volumes.

Third DD Statement: The device you specify must not be of the same type as the one specified in the second DD statement. (If the device type for the prime area is the same as that for the overflow area, you do not need a third DD statement.)

VOLUME Parameter

The VOLUME parameter identifies the volumes on which your data set resides. The format of the VOLUME parameter is:

```
VOLUME=(SER=(serial,...))
       (REF=reference   )
```

You can only specify one serial number in the first or third DD statement. You can only use the REF subparameter if the data set was defined with only one DD statement or if the data set was cataloged. If the data set was defined with more than one DD statement, making a backward reference would only get you the volumes assigned to the first DD statement.

First DD Statement: Specify the serial number of the volume that has the index area. This first DD statement is used when the index area resides on a volume of a different device type from the prime area.

Second DD statement: Code the serial numbers in the same order as they were coded on the DD statements used to create the data set. (See "Examples".)

Third DD Statement: Specify the serial number of the volume that has the overflow area. This third DD statement is used when the overflow area resides on a volume of a different device type from the prime area.

Data Attributes

You can omit the DCB parameter if the data set was passed from a previous step. If the data set was kept or cataloged you must code DSORG=IS. Other parameters you may code are shown in Table 62.

Table 62. DCB Subparameters for Retrieving an Indexed Sequential Data Set

DCB Subparameter	Assembler	COBOL E	COBOL F	ANS COBOL [4]	PL/I F
BFALN=	F or D [1]				
BUFNO=	number of buffers [1]	number of buffers (QISAM only) [1,2]	number of buffers (QISAM only) [1,2]	number of buffers (QISAM only) [1,2]	number of buffers (QISAM only) [1,2]
DSORG=	IS	IS	IS	IS	IS
HIARCHY=	0 or 1 [1]				
NCP=	number of channel programs (BISAM only) [1,3]				number of channel programs (BISAM only) [3]

[1] This function can be specified in your program rather than in the DD statement.

[2] This parameter is used only if you are retrieving or updating the data set sequentially.

[3] This parameter is used only if you are retrieving or updating the data set directly.

[4] American National Standard COBOL.

Overwriting an Indexed Sequential Data Set

You can reuse the space previously allocated to an indexed sequential (ISAM) data set and overlay it with a new ISAM data set. The new data set will keep the name (dsname) of the overwritten data set.

In order to overwrite an ISAM data set, you must know the DD statement, or statements that were used to create it. You indicate that you overwriting the data set by making some changes to the original parameters, and recording the remaining parameters without change.

Data Set Information

The DSNAME and DISP parameters are required.

DSNAME Parameter

The DSNAME parameters must be coded just as you coded them in the original DD statements.

DISP Parameter

The format of the DISP parameter is:

```
DISP=(OLD[,KEEP  ])
         [,DELETE]
```

Indexed area: You must specify the same value you specify for the prime area.

Prime area: If there was only one original DD statement that specified a cataloged data set, you can specify

DISP=(OLD,UNCATLG) or DISP=(OLD,PASS)

If you do not wish to uncatalog or pass the data set or if there were more than one original DD statements specify the DISP parameter as shown in the format above.

Overflow Area: You must specify the same value you specify for the prime area.

Location of the Data Set

The UNIT and VOLUME parameter are required unless the original data set was cataloged.

UNIT Parameter

The UNIT parameters must be coded just as you coded them in the original DD statements.

VOLUME Parameter

If you made a specific volume request when you created the data set, the VOLUME parameters must be coded just as you coded them in the original DD statements.

If you requested nonspecific volumes when you created the data set, you must find out the serial numbers and specify them using VOLUME=SER=(serial,...). If the data set was cataloged you can use VOLUME=REF=dsname. If the data set was defined with only one DD statement in a previous step of the same job, you can make a backward reference using VOLUME=REF.

LABEL Parameter

The LABEL parameter is not needed. Any password protection or retention period specified for the original data set will apply to the new data set.

Size of the Data Set

The space allocation specified for the original data set is retained for the new data set. If the SPACE parameter is coded, it is ignored.

Data Attributes

The DCB parameter must be coded on every DD statement. Two subparameters are required:

```
DSORG=IS or DSORG=ISU
         and
MACRF=PM or MACRF=PL
```

All other subparameters you specified for the original data set will be carried over for the new data set and you do not have to recode them. However, if you want to change the values of one or more of those subparameters, you can code the new subparameter in the DCB parameter. (You can change all the subparameters shown in Table 60 except BUFNO.)

When more than one DD statement is used to define the data set, code all the DCB subparameters on the first DD statement. Code DCB=*.ddname on the remaining statement or statements; ddname is the name given to the first DD statemnt.

Examples

1. The following example shows a job with two steps. In the first step you create an indexed sequential data set with three separate areas. All areas reside on the same volume. In the second step you retrieve and delete the data set. Note that you cannot make a backward reference with the DSNAME parameter because the data set was defined with more than one DD statement.

```
//SAMP1 JOB
//ONE   EXEC PGM=A
//IS    DD   DSNAME=&TOM(INDEX),DISP=(,KEEP),UNIT=2311,
//           VOLUME=(PRIVATE,SER=ABCD),SPACE=(CYL,2),
//           DCB=(DSORG=IS,RECFM=F)
//      DD   DSNAME=&TOM(PRIME),DISP=(,KEEP),UNIT=2311,
//           VOLUME=(PRIVATE,SER=ABCD),SPACE=(CYL,20),DCB=*.IS
//      DD   DSNAME=&TOM(OVFLOW),DISP=(,KEEP),UNIT=2311,
//           VOLUME=(PRIVATE,SER=ABCD),SPACE=(CYL,5),DCB=*.IS
.
.
.
//TWO   EXEC PGM=B
//IN    DD   DSNAME=&TOM,DISP=(OLD,DELETE),DCB=DSORG=IS,
//           UNIT=2311,VOLUME=SER=ABCD
```

2. The following example shows how to create an indexed sequential data set with three separate areas. Each area is on separate volumes. The prime area uses two volumes. Note that you must request the same number of devices as there are volumes for the prime area to use the index area is defined by a separate DD statement. The data set is kept.

```
//DATA DD DSNAME=JANE(INDEX),DISP=(,KEEP),UNIT=2314,
//         VOLUME=SER=AB0001,SPACE=(CYL,5),DCB=DSORG=ISU
//      DD DSNAME=JANE(PRIME),DISP=(,KEEP),UNIT=(2413,2),
//         VOLUME=SER=(AB0002,AB0003),SPACE=(CYL,50),
//         DCB=DSORG=ISU
//      DD DSNAME=JANE(OVFLOW),DISP=(,KEEP),UNIT=2314,
//         VOLUME=SER=AB0004,SPACE=(CYL,10),DCB=DSORG=ISU
```

The following DD statement is used to retrieve the data set. Note that the volumes are requested in the same order they were specified. Also, the same number of units as there are volumes are requested through parallel mounting. DSORG=IS must be coded when retrieving the data set. (ISU can only be coded when creating the data set.)

```
//DD4  DD DSNAME=JANE,DISP=OLD,UNIT=(2314,P)
//         VOLUME=SER=(AB0001,AB0002,AB0003,AB0004),DCB=DSORG=IS
```

The following DD statements must be coded to overwrite the data set. Note that the DISP and DCB parameters were changed and that the SPACE parameter was omitted. The data set is kept.

```
//NEWDS DD DSNAME=JANE(INDEX),DISP=OLD,UNIT=2314,
//         VOLUME=SER=AB0001,DCB=(DSORG=ISU,MACRF=PM)
//      DD DSNAME=JANE(PRIME),DISP=OLD,UNIT=(2314,2),
//         VOLUME=SER=(AB0002,AB0003),DCB=*.NEWDS
//      DD DSNAME=JANE(OVFLOW),DISP=OLD,UNIT=2314,
//         VOLUME=SER=AB0004,DCB=*.NEWDS
```

3. In the following example, an indexed sequential data set without an overflow area is defined. The volumes for the index and prime areas use different types of devices.

Nonspecific requests for private volumes are made. The prime area requests two volumes.

```
//DSET1 DD DSNAME=RAY(INDEX),DISP=(,KEEP),UNIT=2301,
//         VOLUME=PRIVATE,SPACE=(CYL,2),
//         DCB=(DSORG=IS,OPTCD=M,NTM=50)
//      DD DSNAME=RAY(PRIME),DISP=(,KEEP),UNIT=(2311,2),
//         VOLUME=(PRIVATE,,,2),SPACE=(CYL,15),
//         DCB=*.DSETI
```

Two DD statements must be used to retrieve the data set because the index and prime areas use different types of device. Assume that the system assigned volume RFS100 to the index area and RFS101 and RFS102 to the prime area.

```
//INPUT DD DSNAME=RAY,DISP=OLD,UNIT=2301,
//         VOLUME=(PRIVATE,SER=RFS100),DCB=DSORG=IS
//      DD DSNAME=RAY,DISP=OLD,UNIT=(2311,2),
//         VOLUME=(PRIVATE,SER=(RFS101,RFS102)),
//         DCB=DSORG=IS
```

4. The following example defines an indexed sequential data set with an overflow area and an index area embedded in the prime area. The overflow and prime areas reside on the same nonspecific volume. Note that the VOLUME parameter is not needed in the first DD statement because you are requesting a public volume, but it must be used in the second DD statement to make sure that both areas are assigned to the same volume.

```
//EX    JOB
//STEP1 EXEC PGM=X
//DD1   DD   DSNAME=TEMP(PRIME),DISP=(,KEEP),UNIT=2311,
//           SPACE=(CYL,(10,,2)),DCB=(DSORG=IS,BUFNO=2)
//      DD   DSNAME=TEMP(OVFLOW),DISP=(,KEEP),
//           VOLUME=REF=*.DD1,SPACE=(CYL,3),DCB=*.DD1
```

This data set cannot be retrieved in the same job because it is defined with more than one DD statement and a nonspecific volume request is made.

5. The following example defines an indexed sequential data set with an overflow area and an index area placed at the end of the prime area. The prime area resides on three volumes. Only one device is used for the three volumes. You can request fewer devices than volumes because there is no separate DD statement for the index area and you are not requesting an embedded index area. The data set will be cataloged by the IEHPROGM program at a later time.

```
//J74 DD DSNAME=AB.AC(PRIME),DISP=(,KEEP),UNIT=2314,
//       VOLUME=SER=(78910,78911,78912,),SPACE=(CYL,20),
//       DCB=(DSORG=IS,RECFM=VB)
//    DD DSNAME=AB.AC(OVFLOW),DISP=(,KEEP),UNIT=2314,
//       VOLUME=SER=78913,SPACE=(CYL,5),,DCB=*.J74
```

The following DD statement is used to retrieve and uncatalog the data set. Note that the same number of units and volumes is requested and that the volumes are requested in the same order they were specified.

```
//M32 DD DSNAME=AB.AC,DISP=(OLD,UNCATLG),UNIT=(2314,4)
//       VOLUME=SER=(78910,78911,78912,78913),
//       DCB=DSORG=IS
```

6. The following example defines an indexed sequential data set with no overflow area and an embedded index area. The data set resides on a nonspecific public volume. The data set can be cataloged because it is defined only on one DD statement.

```
//ISAM DD DSNAME=MCR,DISP=(,CATLG),UNIT=2302,
//        SPACE=(CYL,(25,,5))
//        DCB=(DSORG=IS,KEYLEN=10)
```

The following DD statement is used to retrieve the data set. The UNIT and VOLUME parameters are not needed because the data set is cataloged.

```
 //JAC DD DSNAME=MCR,DISP=OLD,DCB=DSORG=IS
```

The following DD statement must be coded to overwrite the data set. The DISP and DCB parameters were changed. The SPACE parameter was omitted. The UNIT and VOLUME parameters are not needed because the data set is cataloged.

```
 //OVER DD DSNAME=MCR,DISP=OLD,DCB=(DSORG=IS,MACR=PL)
```

7. The following example defines an indexed sequential data set with an overflow area and an embedded index area. The volumes for the prime and overflow areas use different types of devices.

```
//ABD    DD   DSNAME=JOHN(PRIME),DISP=(,KEEP),UNIT=2314
//           VOLUME=SER=ABD1,SPACE=(CYL,(50,,10)),
//           DCB=(DSORG=IS,KEYLEN=5)
//       DD   DSNAME=JOHN(OVFLOW),DISP=(,KEEP),UNIT=2311,
//           VOLUME=SER=7982,SPACE=(CYL,10),
//           DCB=*.ABD
```

Two DD statements must be used to retrieve the data set because the prime and overflow areas use different types of device.

```
//IN     DD   DSNAME=JOHN,DISP=OLD,UNIT=(2314,2),
//           VOLUME=SER=ABD1,DCB=DSORG=IS
//       DD   DSNAME=JOHN,DISP=OLD,UNIT=2311,
//           VOLUME=SER=7982,DCB=DSORG=IS
```

The following DD statements must be coded to overwrite the data set. Note that the DISP and DCB parameters were changed and that the SPACE parameter was omitted. The data set is kept.

```
//NEWABD DD  DSNAME=JOHN(PRIME),DISP=OLD,UNIT=(2314,2),
//           VOLUME=SER=ABD1,DCB=(DSORG=IS,
//           MACRF=PL,KEYLEN=8)
//        DD  DSNAME=JOHN(OVFLOW),DISP=OLD,UNIT=2311,
//           VOLUME=SER=7982,DCB=*.NEWABD
```

Note: Refer to Example 10 in Part II for additional examples of indexed sequential data sets.

Appendix B: Glossary of DCB Subparameters

This glossary lists the keyword subparameters that you can code in the DCB parameter on a DD statement, their definitions, and the values that can be assigned to them. This glossary must be used with Tables 7, 8, 9, 12, 13, 14, 16, 17, 18, 19, 20, 21, 60 and 62. These tables show which subparameters can be used for your data set. Do not use any subparameters or values that do not apply to your data set as shown in the Tables.

There are other DCB subparameters and values of subparameters not shown in this glossary. They apply to devices, such as graphics and telecommunications, and to access techniques, such as "execute channel program" (EXCP), not described in this manual. The complete list of DCB subparameters is shown in IBM System/360 Operating System: Job Control Language Reference.

BFALN=
specifies the boundary alignment of each buffer as follows:

F
each buffer starts on a fullword boundary that is not also a doubleword boundary.

D
each buffer starts on a doubleword boundary.

If not specified, doubleword boundary alignment (D) is assumed.

BFTEK=
specifies the type of buffering to be used by the control program as follows:

S
simple buffering.

E
exchange buffering (track overflow cannot be specified in the RECFM subparameter). Exchange buffering cannot be used with variable -- length blocked or spanned records.

A
record area buffering. In the locate mode with variable-length spanned records, the control program reads and writes entire logical records rather than segments.

R
record buffering. For writing records in the create BDAM mode, this specification allows a logical record to span one or more tracks. For reading a data set, segments without keys are offset in the buffer by the key length. This means that the actual data starts in the same place in the buffer each read.

If the type of buffering is not specified by any source, simple buffering (S) is assumed.

BLKSIZE=
specifies the maximum length, in bytes, of a block. The maximum length that can be specified is 32,760.

- If RECFM=F, then BLKSIZE must be ≥ logical record length.
- If RECFM=FB, then BLKSIZE must be an integral multiple of the logical record length.
- If RECFM=V, then BLKSIZE must be ≥ (maximum block size *4).
- If RECFM=VB, then BLKSIZE must be n times (logical record length +4); where n is the number of logical records in the block.

Note for Indexed Sequential Data Sets (QISAM): The block size that is specified must be at least 10 bytes less than the number of data bytes available on one track of the allocated direct access device. Block size information is required only when creating a data set containing blocked records.

Note for SYSOUT data sets with RECFM=FB: If the BLKSIZE subparameter on a DD statement for a SYSOUT data set (an output data set being routed through the output stream) is not an integral multiple of and larger than the logical record length (LRECL), the block size will be adjusted to the nearest lower multiple of the logical record length (LRECL).

BUFL=
specifies the length, in bytes, of each buffer in the buffer pool. The maximum length is 32,760 bytes. Requirements for supplying 800 buffer length information vary with the different data organizations and access methods as follows:

- Direct data sets
  Required only if dynamic buffering is specified in the MACRF subparameter of the DCB macro instruction (assembler).
- Partitioned and sequential data sets
  If omitted and the control program acquires buffers automatically, the block size and key length information is used to establish buffer length. If card image is specified (MODE=C), BUFL=160 must be specified.
- Indexed sequential data sets
  Not required if the control program acquires buffers automatically or if dynamic buffering is specified. (For BISAM, dynamic buffering is specified in the MACRF subparameter of the DCB macro instruction.)

BUFNO=
specifies the number of buffers to be assigned to the data control block; the maximum number is 255, but the actual number allowed may be less than 255 because of limits established when the system was generated.

CODE=
specifies the paper tape code in which the data set is punched.

A USASCII (8 track).
B Burroughs (7 track).
C National Cash Register (8 track).
F Friden (8 track).
I IBM BCD perforated tape and transmission code (8 track).
N No conversion required.
T Teletype (5 track).

If not specified by any source, I is assumed.

CYLOFL=
specifies the number of tracks on each cylinder to hold the records that overflow from other tracks on that cylinder. The maximum number is 99.

DEN=
specifies the magnetic tape density in number of bits-per-inch used to write a data set, as follows:

DEN=	7 track	9 track
0	200	---
1	556	---
2	800	800
3	---	1600

If not specified by any source, 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 with dual density.

For 7-track tape, all information on the reel must be written in the same density (i.e., labels, data, tapemarks). Do not specify DEN for a SYSOUT data set.

DSORG=
specifies the organization of the data set and whether the data set contains any location-dependent information that would make the data set unmovable (U). The values that can be used are as follows:

DA
: Direct access

DAU
: Direct access unmovable

IS
: Indexed sequential

ISU
: Indexed sequential unmovable

PO
: Partitioned organization

POU
: Partitioned organization unmovable

PS
: Physical sequential

PSU
: Physical sequential unmovable

EROPT=
specifies the option to be executed if an error occurs in writing or reading a record, as follows:

ACC
: accept the block causing the error.

SKP
: skip the block causing the error (implies RELSE).

ABE
: cause abnormal end of task.

If the subparameter is not specified by any source, ABE is assumed.

HIARCHY=
specifies the storage hierarchy in which the buffer pool is to be formed as follows:

0
forms the pool from available space in processor storage.

1
forms the pool from available space in IBM 2361 Core Storage.

If the HIARCHY subparameter is not specified by any source, and if a hierarchy designation is not supplied by the GETPOOL macro instruction, hierarchy 0 is assumed.

The buffer pool is formed within the indicated hierarchy, or in the case of MFT or MVT, in the user partiiton or region in that hierarchy. If space is unavailable within the hierarchy specified, the task is abnormally terminated.

LIMCT=
specifies the number of blocks, if relative block addressing is used, or the number of tracks, if relative track addressing is used, that are to be searched for a block or available space when the extended search option (OPTCD=E) is specified. The number may equal or exceed the number of blocks or tracks in the data set, in which case the entire data set is searched.

If the extended search option is not specified, the LIMCT subparameter is ignored.

Note for direct data sets: If standard labels are used, the key length information can be supplied from the data set label for an existing data set. If a key length is not supplied by any source, no input or output requests that require a key may be issued.

Note for partitioned and sequential data sets: If standard labels are used, the key length information can be supplied from the data set label for an existing data set. If a key length is not supplied by any source before the OPEN macro instruction is issued, a length of zero (no keys) is assumed.

Note for indexed sequential data sets: For an existing data set with standard labels, the key length can only be supplied from the data set label.

LRECL=
specifies the actual or maximum length, in bytes, of a logical record. The record length is required for fixed-length and variable length records; for variable-length records, the maximum record length should be specified. The length cannot exceed the blocksize (BLKSIZE) value except for variable-length spanned records.

- If RECFM=V or VB, then LRECL must be equal to the maximum record length +4.
- If RECFM=F or FB, then LRECL must be equal to the logical record length.
- If RECFM=U, then LRECL should be omitted.

Note for partitioned data sets: The record length is required for fixed-length records only.

Note for sequential data sets: The record length can be omitted from all sources, in which case the block size specification (BLKSIZE) is used. For variable-length spanned records (VS or VBS) processed under QSAM (locate mode) BSAM, if logical record exceeds 32,756, specify LRECL=X.

Note for indexed sequential data sets: For unblocked records, with a relative key position (RKP) of zero, the record length includes only the data portion of the record. The record length can be specified only when creating the data set.

MODE=
specifies the mode of operation to be used with a card reader, a card punch, or a card-read punch, as follows:

C
the card image (column binary) mode.

E
the EBCDIC mode.

If this information is not supplied by any source, E is assumed.

NCP=
specifies the maximum number of READ or WRITE macro instructions issued before a CHECK macro instruction is issued. The maximum number allowed is 99, based on limits established when the system was generated. If chained scheduling is used, NCP must be specified as more than 1.

If not specified by any source, 1 is assumed.

NTM=
specifies the number of tracks to be used for a cylinder index. When the specified number of tracks has been filled, a master index is created. This information is required only when the master index option (OPTCD=M) is selected.

If not specified by any source and OPTCD=M is specified, the master index option is ignored.

OPTCD=
specifies the optional services to be performed by the control program. All optional services must be requested by the same source. The characters may be coded in any order and, when used in combination, no commas are permitted between characters.

A
actual device address are to be presented ("block address" operand) in READ and WRITE macro instructions. For BDAM, P requests the same option as A, and either can be coded.

B
requests that end-of-file recognition be disregarded for tapes.

C
requests that chained scheduling is used.

E
an extended search (more than one track) is to be performed for a block or available space. (The LIMCT subparameter must also be specified; otherwise, this option is ignored.)

F
feedback may be requested in READ and WRITE macro instructions and the device address returned is to be of the form presented to the control program.

I

requests that the control program use the independent overflow areas for overflow records.

L

requests that the control program delete records that have a first byte of all ones; records so marked may be deleted when space is required for new records. Do not specify this option for blocked records if RKP=0.

M

requests that master indexes be created as required according to the information in the NTM subparameter. His option is ignored if the subparameter NTM=number if not specified.

R

for direct data sets actual device addresses are to be presented ("block address" operand) in READ and WRITE macro instructions. For indexed sequential data sets, requests the control program to place reorganization criteria information in the RORG1, RORG2, and RORG3 fields of the data control block. This option is provided whenever the OPTCD subparameter is omitted from all sources.

T

requests user totaling facility.

U

only for 1403 printers with the Universal Character Set feature: Unblocks data checks and allows analysis by an appropriate error analysis (SYNAD) routine. If U is omitted, data checks are blocked (not recognized as errors).

W

requests a validity check for write operations on direct access devices. If the device is a 2321 data cell, validity checking is always performed, whether requested or not.

Y

requests that the control program use the cylinder overflow areas for overflow records.

Z

Only for input on magnetic tape: Requests the control program to shorten its normal error recovery procedure. When Z is specified, a data check is considered permanent after five unsuccessful attempts to read a record. This option is available only if selected at system generation. It should be used only when a tape is know to be faulty and there is no need to process every record. The error analysis (SYNAD) routine should keep a count of the number of permanent errors, and should terminate processing if the number becomes excessive.

PRTSP=

specifies the line spacing on a printer as 0, 1, 2, or 3 lines between printout. This subparameter is valid only if control characters are not present (A or M is not specified in the RECFM subparameter). If not supplied by any source, 1 is assumed.

RECFM=

specifies the format and characteristics of the records in the data set. The format and characteristics must be completely described by one source.

If this subparameter is omitted, an undefined-length record is assumed with no optional features provided, except for indexed sequential data sets where variable-length records are assumed. Both the record format and characteristics are specified using the characters defined below. The allowable combinations of characters are indicated for the associated access methods; the characters must be coded in the order shown.

Character	Definitions
A	The record contains USASI control characters.
B	The records are blocked.
F	The records are of fixed length.
M	The records contain machine code control characters.
S	For fixed-length records, the records are written as standard blocks, i.e., no truncated blocks or unfilled tracks within the data set, with the exception of the last block or track. For variable-length records, a record may space more than one block. Exchange buffering (BFTEK=E) cannot be specified.
T	The records may be written onto overflow tracks if required. Exchange buffering (BFTEK=E) or chained scheduling (OPTCD=C) cannot be used.
U	The records are of undefined length.
V	The records are of variable length.

RKP=
specifies the position of the first byte of the record key, relative to the beginning of each record. (The beginning byte of a record is addressed as 0.) If RKP=0 is specified for blocked fixed-length records, the key begins in the first byte of each record, and the delete option (OPTCD=L) must not be specified. If RKP=0 is specified for unblocked fixed-length records, the key is not written in the data field; the delete option can be specified.

For variable-length records, the relative key position must be 4 or greater, when the delete option (OPTCD=L) is not specified. The relative key position must be 5 or greater if the delete option is specified. If this information is not specified by any source, a relative key position of zero is assumed.

STACK=
specifies the stacker bin to receive the card, and is either 1 of 2. If not specified by any source, 1 is assumed.

TRTCH=
specifies the recording technique for seven-track tape.

C
Data conversion feature is to be used, with odd parity and no translation.

E
Even parity, with no translation and no conversion.

T
Odd parity and no conversion, and BCD to EBCDIC to BCD translation when writing.

ET
Even parity and no conversion, and BCD to EBCDIC translation is required when reading; EBCDIC to BCD translation when writing. If this subparameter is not specified by any source, odd parity and no translation or data conversion is assumed.

Appendix C: Using the Restart Facilities

When a job step abnormally terminates, you may have to resubmit the job for execution. This means lost computer time and a delay in obtaining the desired results. To reduce these effects, you can use the restart facilities.

If a job step abnormally terminates or if a system failure occurs in a system with MFT or MVT, the restart facilities allows you to request that the job step be restarted either at the beginning of the step (step restart) or within the step (checkpoint restart). Furthermore, restart can occur automatically after abnormal termination, or it can be deferred until the job is resubmitted.

Note: This appendix is essentially the same as Appendix B of the publication IBM System/360 Operating System: Job Control Language Reference. It is presented in both publications for the sake of completeness.

Restarts

For automatic step restart to occur, the RD parameter must request it on the JOB statement or on the EXEC statement associated with the step that abnormally terminates. (The RD parameter is described in the sections, "The Job Statement" and "The EXEC Statement". Automatic checkpoint restart can occur only if a CHKPT macro instruction is executed in the processing program prior to abnormal termination. If restart is deferred until the job is resubmitted, the RESTART parameter must be coded on the JOB statement of the resubmitted job. (The RESTART parameter is described in the section "The JOB Statement".) The RESTART parameter identifies the step or the step the checkpoint at which execution is to be resumed. A deferred restart may be initiated regardless of how the resubmitted job was previously terminated (normally or abnormally) and regardless of whether an automatic restart occurred during the original execution.

Automatic Step Restart: If an abnormally terminated step is to be automatically restarted, the RD parameter must be coded as RD=R or RD=RNC. Execution resumes at the beginning of the abnormally terminated step.

Automatic Checkpoint Restart: After an automatic checkpoint restart, execution resumes at the instruction immediately following the last CHKPT macro instruction that was successfully executed in the abnormally terminated step. An automatic checkpoint restart cannot occur if you suppress the action of the CHKPT macro instruction; you do this by coding RD=NC or RD=RNC. Also, an automatic checkpoint restart cannot occur if you code RD=NR; however, RD=NR allows the CHKPT macro instruction to establish a checkpoint.

Deferred Step Restart: To perform a deferred step restart, the RESTART parameter must identify the step at which execution is to be resumed. Steps preceding the restart step are interpreted but not initiated. Since disposition processing occurred during the original execution of

the job, you may have to modify control statements associated with the restart step before you resubmit the job. Modifications may be required in two cases:

1. A data set was defined as NEW during the original execution. If it was created during the original execution, you must change the data set's status to OLD, define a new data set, or delete the data set before resubmitting the job.

2. A data set was passed and was to be received by the restart step or a step following the restart step. If the passed data set is not cataloged, you must supply, in the receiving step, volume serial numbers, device type, data set sequence number, and label type. (Label type cannot be retrieved from the catalog.)

To limit the number of modifications required before you resubmit the job, you can assign conditional dispositions during the original execution. (Data sets assigned a temporary name or no name can only be assigned a conditional disposition of DELETE.) If deferred step restart will be performed, conditional dispositions should be used:

- To delete all new data sets created by the restart step.

- To keep all old data sets used by the restart step, other than those passed to the step. (If a nontemporary data set is defined as DISP=(OLD,DELETE), it is very important that you assign a conditional disposition of KEEP.)

- To catalog all data sets passed from steps preceding the restart step to the restart step or to steps following the restart step.

Additional changes can be made to your control statements before resubmitting the job. For example, you can vary device and volume configurations and request step restart on an alternate system with the same configuration as used originally. You can also make changes to your data.

Deferred Checkpoint Restart: To perform a deferred checkpoint restart, the RESTART parameter must identify the step and the checkpoint at which execution is to be resumed. The SYSCHK DD statement, which defines the checkpoint data set, must also be included. (The SYSCHK DD statement is described in "Special DD Statements" in the section "The DD Statement."

An internal representation of your statements is kept as control information within the system. Some of the control information for the restart step or steps following the restart step may have to be modified before execution can be resumed at a checkpoint. The following modifications for the restart step are automatically made by the system, using information contained in the checkpoint entry:

- The status of data sets used by the step is changed from NEW to OLD. (If a new data set was assigned a nonspecific volume and had not been opened before the checkpoint was established, this change is not made.)

- If nonspecific volumes were requested for a data set used in the restart step, the assigned device type and volume serial numbers are made part of the control information.

- For a multivolume data set, the volume being processed when the checkpoint was established is mounted.

The only modification that you must make to a control statement is to supply certain information about a data set that was being passed by a step preceding the restart step to a step, following a restart step. You must supply, in the receiving step, volume serial numbers, device type, data set sequence number, and label type. You will not have to make these modifications if, during the original execution, you assigned a conditional dispostion of CATLG to such data sets. If the data is cataloged, the system can retrieve this information from the catalog. (Label type cannot be retrieved from the catalog.) You should also use conditional dispositions to keep all data sets used by the restart step. Data sets assigned a temporary name or no name can only be assigned a conditional disposition of DELETE.

Therefore, if you plan a deferred checkpoint restart, you should not define your data sets as temporary. (For any nontemporary data set that may be deleted, it is very important that you assign a conditional disposition of KEEP.)

Before resubmitting the job for checkpoint restart, you can make other modifications to control statements associated with the restart step or steps following the restart step. The following items apply to the step in which restart is to occur.

- The DD statements in the restart step can be altered, but the statements must have the same names as used originally. You can also include additional DD statements.

- If a data set was open at the time a checkpoint was established and restart is to begin at that checkpoint, DD statements in the restart step can define the same data set. If there is no need to process a data set after restart, you can define the data set by coding the DUMMY parameter or DSNAME=NULLFILE on a DD statement provided that: (1) the basic sequential access method (BSAM) or the queued sequential access method (QSAM) was being used to process the data set when the checkpoint was established, (2) the data set is not the checkpoint data set that is being used to restart the job step, and (3) the job step is not restarted from a checkpoint that is within an end-of-volume exit routine for the data set. The name of the DD statement must be the same as the one used for the data set during the original execution of your program.

- If DUMMY is not specified the DD statements must define the same data sets. Also, the data sets must not have been moved on the volume or onto another volume.

- If a data set was not open when the checkpoint was established and is not needed during restart, you can replace the parmaeters used to define the data set with the DUMMY parameter.

- You can alter the data in the restart step. If you omit the data, a delimiter statement (/*) is not required, unless the data was preceded by a DD DATA statement.

Modifications you might want to make to control statements following the restart step are: varying device and volume configurations, altering data, and possibly, requesting checkpoint restart on an alternate system with the same configuration as used originally. If the parameters PGM, COND, SUBALLOC and VOLUME=REF refer to steps preceding the restart step, you must resolve these references before resubmitting the job. (A backward reference of VOLUME=SER=(serial number).)

Examples of Using the Restart Facilities

1. The following control statements illustrate the preparations that would be made for either an automatic step or checkpoint restart before the job is submitted for the first time.

```
//STMRG3   JOB  54321,A.USER,MSGLEVEL=(1,0),RD=R
//STEP1    EXEC PGM=SIMPSORT
//INPUT    DD   DSN=SORTIN,VOL=SER=100468,UNIT=2400,
//              DISP=(OLD,DELETE)
//OUTPUT   DD   DSN=INV(+1),UNIT=2311,VOL=SER=555334,
//              SPACE=(3200,(200,100)),DISP=(NEW,CATLG)
//WORK1    DD   UNIT=2400,DISP=(NEW,DELETE)
//WORK2    DD   UNIT=2400,DISP=(NEW,DELETE)
//CHKPT    DD   UNIT=2400,DISP=(NEW,DELETE)
//STEP2    EXEC PGM=MYMERGE
//MERG1    DD   DSN=INV(+1),DISP=OLD
//MERG2    DD   DSN=M5,VOL=SER=(092501,092502,092503),
//              UNIT=(2400,3),DISP=(OLD,KEEP)
//RESULTS  DD   DSN=M6,UNIT=2400,VOL=SER=(100101,100102,
//              100103),DISP=(NEW,KEEP,DELETE)
```

Here, the RD parameter requests step restart for any abnormally terminated job step. In STEP1, the DD statement CHKPT defines a checkpoint data set. For this step, once a CHKPT macro instruction is executed, only automatic checkpoint restart is performed. An automatic checkpoint restart cannot occur in STEP2 since a checkpoint data set is not defined.

2. The following control statements illustrate the preparations that would be made for either an automatic or deferred step restart before the job is submitted for the first time.

```
//STMRG3   JOB  54321,A.USER,MSGLEVEL=(1,0),RD=R
//STEP1    EXEC PGM=SIMPSORT
//INPUT    DD   DSN=SORTIN,VOL=SER=100468,UNIT=2400,
//              DISP=(OLD,DELETE,KEEP)
//OUTPUT   DD   DSN=INV(+1),UNIT=2311,VOL=SER=555334,
//              SPACE=(3200,(200,100)),DISP=(NEW,CATLG,DELETE)
//WORK1    DD   UNIT=2400,DISP=(NEW,DELETE)
//WORK2    DD   UNIT=2400,DISP=(NEW,DELETE)
//STEP2    EXEC PGM=MYMERGE
//MERG1    DD   DSN=INV(+1),DISP=OLD
//MERG2    DD   DSN=M5,VOL=SER=(092501,092502,092503),
//              UNIT=(2400,3),DISP=(OLD,KEEP)
//RESULTS  DD   DSN=M6,UNIT=2400,VOL=SER=(100101,100102,
//              100103),DISP=(NEW,KEEP,DELETE)
```

If you are resubmitting this resubmitting this job for step restart, you must code the RESTART parameter on the JOB statement and identify the step at which execution is to be resumed. If execution is to be resumed with STEP2, the MERG1 DD statement must be changed to refer to the generation data set by means of its present relative generation number, i.e.,

```
DSN=INV(0)
```

3. The following control statements illustate the preparations that would be made for an automatic step or checkpoint restart or a deferred checkpoint restart before the job is submitted for the first time.

```
//STMRG3  JOB  54321,A.USER,MSGLEVEL=(1,0),RD=R
//STEP1   EXEC PGM=SIMPSORT
//INPUT   DD   DSN=SORTIN,VOL=SER=100468,UNIT=2400,
//             DISP=(OLD,DELETE,KEEP)
//OUTPUT  DD   DSN=INV(+1),UNIT=2311,VOL=SER=555334,
//             SPACE=(3200,(200,100)),DISP=(NEW,CATLG,KEEP)
//WORK1   DD   DSN=A,UNIT=2400,DISP=(NEW,DELETE,CATLG)
//WORK2   DD   DSN=B,UNIT=2400,DISP=(NEW,DELETE,CATLG)
//CHKPT   DD   DSN=C,UNIT=2400,DISP=(NEW,DELETE,CATLG)
//STEP2   EXEC PGM=MYMERGE
//MERG1   DD   DSN=INV(+1),DISP=OLD
//MERG2   DD   DSN=M5,VOL=SER=(092501,092502,092503),
//             UNIT=(2400,3),DISP=(OLD,KEEP)
//RESULTS DD   DSN=M6,UNIT=2400,VOL=SER=(100101,100102,
//             100103),DISP=(NEW,KEEP)
```

Either an automatic automatic checkpoint restart or a deferred checkpoint restart can occur in STEP1 if the step abnormally terminates. To perform a deferred checkpoint restart, the RESTART parameter must be coded on the JOB statement and a SYSCHK DD statement must be included before resubmitting the job. Only automatic step restart can occur in STEP2. The data sets that would normally be defined as temporary data sets so conditional dispositions can be assigned to them.

Appendix D: Summary of the DD Statement

This appendix summarizes the DD statement parameters required to perform the following functions:

For further convenience in using the above Tables, you should also use the fold-out Tables in the section "The DD Statement". Table 5 should be used with Tables 63-66; Table 15, with Tables 67-71; and Table 22, with Tables 72-74.

The DDNAME is not shown in the Tables of this Appendix. You should write the ddname as explained in "Naming the DD Statement" in the section "The DD Statement".

Indexed sequential data sets are described in Appendix A.

Table 63. Creating a Data Set on an Unit Record Device (Card Punch or Printer)

Parameter Type	Parameter	Format
Location of the Data Set	UNIT	UNIT={unit address / 1052 / 1403 / 1442 / 1443 / 2520 / 2540-2 / group name}
Data Attributes	DCB	[DCB=(list of attributes) DCB=({*.ddname / *.stepname.ddname / *.stepname.procstepname.ddname}[,list of attributes])] Notes: See Tables 7 and 8 for attributes
Special Processing Options	UCS	[UCS=(code[,FOLD[,]][,VERIFY])] Notes: Choose one for "code": AN, HN, PCAN, PCHN, PN, QNC, QN, RN, TN, SN, XN, or YN.
	DUMMY	[DUMMY] Note: Positional parameter

Table 64. Creating a Data Set on a System Output Device

Parameter Type	Parameter	Format
Location of the Data Set	SYSOUT	For systems with PCP: SYSOUT=classname For systems with MFT or MVT: SYSOUT=(classname [,program / ,] [,form numbers]) Note: "Classname" is a letter (A-Z) or a number (0-9)
	UNIT	See Table 66.
Size of the Data Set	SPACE	See Table 66.
Data Attributes	DCB	See Tables 7 and 8 for PCP, and Table 12 for MFT or MVT.
Special Processing Option	OUTLIM	OUTLIM=number Note: For systems with MFT or MVT that have the System Management Facilities option.

Table 65. Creating a Data Set on a Magnetic Tape (Part 1 of 2)

Parameter Type	Parameter	Format
Data Set Information	DSNAME and DISP	Temporary - for the duration of the job: DSNAME={&&name / &name},DISP=(NEW,PASS,DELETE) Temporary - for duration of the job step: DSNAME={&&name / &name} ,DISP=(NEW,DELETE,DELETE) Nontemporary - cataloged: DSNAME=dsname,DISP=(NEW,CATLG[,CATLG / ,DELETE / ,KEEP]) Nontemporary - kept: DSNAME=dsname,DISP=(NEW,KEEP[,KEEP / ,DELETE / ,CATLG]) Nontemporary - member generation data group DSNAME=groupname(+number),DISP=(NEW,CATLG[,CATLG / ,DELETE])
Location of the Data Set	UNIT	UNIT=({unit address / 2400 / 2400-1 / 2400-2 / 2400-3 / 2400-4 / group name}[,unitcount / ,P / ,][,DEFER])
	VOLUME	Nonspecific request - one scratch volume (temporary data set): Omit Nonspecific request - more than one scratch volumes (temporary data set): VOLUME=(,,,volcount) Nonspecific request - private volume (temporary data set): VOLUME=(PRIVATE[,,,volcount]) Nonspecific request - private volume (nontemporary data set): VOLUME=([PRIVATE,RETAIN / ,] [,,volcount]) Specific request - private volume (request by serial number): VOLUME=([PRIVATE,RETAIN / ,][,,volcount, / ,]SER=(serial,...))

Table 65. Creating a Data Set on a Magnetic Tape (Part 2 of 2)

Parameter Type	Parameter	Format
Location of the Date Set (cont.)	VOLUME (cont.)	Specific request - private volumes used by other data set): VOLUME=([PRIVATE,RETAIN / ,] [,,volcount, / ,] REF={dsname / *.ddname / *.stepname.ddname / *.stepname.procstepname.ddname}
	LABEL	LABEL=([sequence] [,SL / ,SUL / ,NSL / ,NL / ,BLP / ,] [,PASSWORD / ,] [,IN / ,OUT] [,] [EXPDT=yyddd / RETPD=nnnn])
Data Attributes	DCB	[DCB=(list of attributes) DCB=({*.ddname / *.stepname.ddname / *.stepname.procstepname.ddname}[,list of attributes])] Note: See Table 9 for attributes
Special Processing Options	SEP or AFF	[SEP=(ddname,...) / AFF=ddname]
	DUMMY	[DUMMY] Note: Positional parameter

Table 66. Creating a Data Set on Direct Access Devices (Part 1 of 3)

Parameter Type	Parameter	Format
Data Set Information	DSNAME and DISP	Temporary - for the duration of the job: DSNAME={&&name / &name / &&name(membername) / &name(membername)},DISP=(NEW,PASS,DELETE)) Temporary - for the duration of the job step: DSNAME={&&name / &name / &&name(membername) / &name(membername)},DISP=(NEW,DELETE,DELETE) Nontemporary - cataloged: DSNAME={dsname / dsname(membername) / dsname(membername)},DISP=(NEW,CATLG [,CATLG / ,DELETE / ,KEEP]) Nontemporary - kept: DSNAME={dsname / dsname(membername)},DISP=(NEW,KEEP[,KEEP / ,DELETE / ,CATLG]) Nontemporary - member generation data group: DSNAME=groupname(+number),DISP=(NEW,CATLG[,CATLG / ,DELETE])
Location of the Data Set	UNIT	UNIT=({unit address / 2301 / 2302 / 2302 / 2303 / 2311 / 2314 / 2321 / group name}[,unitcount / ,P / ,][,SEP=(ddname,...)])
	VOLUME	Nonspecific request - one public volume: Omit Nonspecific request - more than one public volumes: VOLUME=(,,,volcount) Nonspecific request - private volume: VOLUME=(PRIVATE[,RETAIN / ,][,,volcount]) Specific request - public volume (request by serial number): VOLUME=([,,,volcount,]SER=(serial,...))

Table 66. Creating a Data Set on Direct Access Devices (Part 2 of 3)

Parameter Type	Parameter	Format
Location of the Data Set (cont.)	VOLUME (cont.)	Specific request - public volume (volumes used by other data set): VOLUME=([,,,volcount,]REF={dsname | *.ddname | *.stepname.ddname | *.stepname.procstepname.ddname}) Specific request - private volume (request by serial number): VOLUME=(PRIVATE[,RETAIN][,,volcount,]SER=(serial,...)) Specific request - private volume (volumes used by other data set): VOLUME=(PRIVATE[,RETAIN][,,volcount,] REF={dsname | *.ddname | *.stepname.ddname | *.stepname.procstepname.ddname})
	LABEL	[LABEL=([,SL | ,SUL][,PASSWORD][,IN | ,OUT][,][EXPDT=yyddd | RETPD=nnnn])
Size of the Data Set (choose one)	SPACE	The system assigns tracks: SPACE=({TRK | CYL | block length},(primary quantity[,secondary quantity][,directory])[,RLSE][,CONTIG | ,MXIG | ,ALX][,ROUND]) Requesting specific tracks: SPACE=(ABSTR,(primary quantity,address[,directory]))
	SPLIT	Units of cylinders: • First data set SPLIT=(n,CYL,(primary quantity [,secondary quantity])) • Subsequent data sets SPLIT=n Units of blocks: • First data set SPLIT=(%,blocklength,(primary quantity[,secondary quantity])) • Subsequent data sets SPLIT=%

Table 66. Creating a Data Set on Direct Access Devices (Part 3 of 3)

Parameter Type	Parameter	Format
Size of the Data Set (choose one) (cont.)	SUBALLOC	Reserved area: SPACE=({TRK / CYL / blocklength},(primary quantity),,CONTIG) Suballocaed data sets: SUBALLOC=({TRK / CYL / block length},(primary quantity [,secondary quantity ,] [,directory]),{ddname / stepname,ddname / stepname.procstepname.ddname})
Data Attributes	DCB	[DCB=(list of attributes) DCB=({dsname / *.ddname / *.stepname.ddname / *stepname.procstepname.ddname} [,list of attributes])] Note: See Tables 12, 13, and 14 for attributes
Special Processing Options	SEP or AFF	[SEP=(ddname,...) / AFF=ddname]
	DUMMY	[DUMMY] Note: Positional parameter

Table 67. Retrieving an Existing Data Set From an Unit Record Device (Card Reader or Paper Tape Reader)

Parameter Type	Parameter	Format
Location of the Data Set	UNIT	UNIT={unit address / 1442 / 2520 / 2540 / 2671 / group name}
Data Attributes	DCB	[DCB=(list of attributes)] DCB=({*.ddname / *.ddname.stepname / *.ddname.procstepname.ddname}[,list of attributes]) Note: See Tables 16 and 17 for attributes
Special Processing Option	DUMMY	[DUMMY] Note: Positional parameter

Appendix

Table 68. Retrieving a Data Set From the Input Stream

Parameter Type	Parameter	Format
Location of the Data Set	* or DATA	{* / DATA}
Data Attributes	DCB	[DCB=([BLKSIZE=number of bytes][,][BUFNO=number of buffers])]

Table 69. Retrieving a Passed Data Set (Magnetic Tape or Direct Access)

Parameter Type	Parameter	Format
Data Set Information	DSNAME	Retrieving by name: DSNAME={dsname | dsname(membername) | &&name | &&name(membername) | &name | &name(membername)} Making backward reference: DSNAME={*.ddname | *.stepname.ddname | *.stepname.procstepname.ddname}
	DISP	Passing the data set to a subsequent step: DISP=(OLD,PASS[,PASS | ,DELETE | ,CATLG | ,KEEP]) Last time a passed temporary data set is used in job: DISP=(OLD,DELETE,DELETE) Last time a passed nontemporary data set is used in job: DISP=(OLD[,KEEP | ,DELETE | ,CATLG | ,][,KEEP | ,DELETE | ,CATLG])
Location of the Data Set	UNIT	[UNIT=(,unitcount)]
	LABEL	[LABEL={,SUL | ,NL | ,NSL | ,BLP}]
Data Attributes	DCB	[DCB=(list of attributes) DCB={*.ddname | *.stepname.ddname | *.stepname.procstepname.ddname}] Note: See Tables 18, 19, 20, and 21 for attributes
Special Processing Option	DUMMY	[DUMMY] Note: Positional parameter

Appendix

Table 70. Retrieving a Cataloged Data Set (Magnetic Tape or Direct Access)

Parameter Type	Parameter	Format
Data Set Information	DSNAME	DSNAME={dsname / dsname(membername) / groupname(number)}
	DISP	DISP=({OLD / SHR}[,UNCATLG / ,DELETE / ,][,UNCATLG / ,DELETE])
Location of the Data Set	UNIT	Requesting units: UNIT=(,[unitcount / P][,DEFER])
		Unit affinity: [UNIT=AFF=ddname]
	VOLUME	VOLUME=([PRIVATE[,RETAIN / ,][,sequence])
	LABEL	[LABEL=(,{SUL / NL / NSL / BLP})]
Data Attributes	DCB	[DCB=(list of attributes) DCB=({dsname / *.ddname / *.stepname,ddname / *.stepname.procstepname.ddname}[,list of attributes])] Note: See Tables 18, 19, 20, and 21 for attributes
Special Processing Options	SEP or AFF	[SEP=(ddname,...) / AFF=ddname]
	DUMMY	[DUMMY] Note: Positional parameter

Table 71. Retrieving a Kept Data Set (Magnetic Tape or Direct Access)

Parameter Type	Parameter	Format
Data Set Information	DSNAME	DSNAME={dsname / dsname(membername)}
	DISP	DISP=({OLD / SHR}[,KEEP / ,CATLG / ,DELETE / ,][,KEEP / ,CATLG / ,DELETE])
Location of the Data Set	UNIT	Requesting units: UNIT=({unitaddress / devicetype / groupname}[,unitcount / ,P / ,][,DEFER])
		Unit affinity: UNIT=AFF=ddname
	VOLUME	VOLUME=([PRIVATE][,RETAIN / ,]SER=(serial,...))
	LABEL	[LABEL=([sequence#][,SUL / ,NL / ,NSL / ,BLP])]
Data Attributes	DCB	[DCB=(list of attributes) DCB=({dsname / *.ddname / *.stepname.ddname / *.stepname.procstepname.ddname}[,list of attributes])]
		Note: See Tables 18, 19, 20, and 21 for attributes
Special Processing Options	SEP or AFF	[SEP=(ddname,...) / AFF=ddname]
	DUMMY	[DUMMY]
		Note: Positional parameter

Table 72. Extending a Passed Data Set (Magnetic Tape or Direct Access)

Parameter Type	Parameter	Format
Data Set Information	DSNAME	Retrieving by name: DSNAME={dsname | dsname(membername) | &&name | &&name(membername) | &name | &name(membername)} Making a backward reference: DSNAME={*.ddname | *.stepname.ddname | *.stepname.procstepname.ddname}
	DISP	Passing the data set to a subsequent step: DISP=(MOD,PASS[,PASS | ,DELETE | ,CATALG | ,KEEP]) last time a passed temporary data set is used in job: DISP=(MOD,DELETE,DELETE last time a passed nontemporary data set is used in job: DISP=(MOD[,KEEP | ,DELETE | ,CATLG | ,][,KEEP | ,DELETE | ,CATLG | ,])
Location of the Data Set	UNIT	[UNIT=([,unitcount][,DEFER]]
	VOLUME	[VOLUME=(,,,volcount)]
	LABEL	[LABEL=({,SUL | ,NL | ,NSL | ,BLP})]
Size of the Data Set	SPACE	[SPACE=({TRK | CYL | blocklength} (1,secondary quantity))]
Data Attributes	DCB	[DCB=(list of attributes) DCB={*.ddname | *.stepname.ddname | *.stepname.procstepname.ddname}[,list of attributes]] Note: See Tables 18, 19, and 21 for attributes
Special Processing Option	DUMMY	[DUMMY] Note: Positional parameter

Table 73. Extending a Cataloged Data Set (Magnetic Tape or Direct Access)

Parameter Type	Parameter	Format
Data Set Information	DSNAME	DSNAME={dsname / dsname(membername)}
	DISP	DISP=(MOD[,UNCATLG / ,DELETE / ,CATLG / ,][,UNCATLG / ,DELETE / ,CATLG])
Location of the Data Set	UNIT	Requesting units: [UNIT=(,[unitcount / P][,DEFER])] Unit affinity: [UNIT=AFF=ddname]
	VOLUME	VOLUME=([PRIVATE[,RETAIN / ,][,sequence# / ,][,volcount])
	LABEL	[LABEL=(,SUL / ,NL / ,NSL / ,BLP)]
Size of the Data Set	SPACE	[SPACE=(TRK / CYL / blocklength,(1,secondary quantity))]
Data Attributes	DCB	[DCB=(list of attributes)] [DCB=(dsname / *.ddname / *.stepname.ddname / *.stepname.procstepname.ddname[,list of attributes])] Note: See Tables 18, 19, and 21 for attributes
Special Processing Options	SEP or AFF	[SEP=(ddname,...) / AFF=ddname]
	DUMMY	[DUMMY] Note: Positional parameter

Appendix

Table 74. Extending a Kept Data Set (Magnetic Tape or Direct Access)

Parameter Type	Parameter	Format
Data Set Information	DSNAME	DSNAME={dsname / dsname(membername)}
	DISP	DISP=(MOD[,KEEP / ,CATLG / ,DELETE / ,][,KEEP / ,CATLG / ,DELETE])
Location of the Data Set	UNIT	Requesting units: UNIT=({unitaddress / devicetype / groupname}[,unitcount / ,P / ,][,DEFER])
		Unit affinity: UNIT=AFF=ddname
	VOLUME	VOLUME=(PRIVATE[,RETAIN / ,][,,volcount,]SER=(serial,...))
	LABEL	[LABEL=([sequence#][,SUL / ,NL / ,NSL / ,BLP])]
Size of the Data Set	SPACE	[SPACE=({TRK / CYL / blocklength},(1,secondary quantity))]
Data Attributes	DCB	[DCB=(list of attributes) DCB=({dsname / *.ddname / *.stepname.ddname / *.stepname.procstepname.ddname}[,list of attributes])] Note: See Tables 18, 19, and 21 for attributes
Special Processing Options	SEP or AFF	[SEP=(ddname,...) / AFF=ddname]
	DUMMY	[DUMMY] Note: Positional parameter

Table 75. Postponing Definition of a Data Set

Parameter Type	Parameter	Format
Special Processing Option	DDNAME	DDNAME=ddname
Data Attributes	DCB	[DCB=([BLKSIZE=number of bytes][,][BUFNO=number of buffers)]

Appendix

Glossary

* parameter: This parameter is coded as the first parameter on a DD statement that precedes data in the input stream.

ACCT parameter: This parameter is used to supply accounting information for a job step to an installation accounting routine and is coded on an EXEC statement.

AFF parameter: This parameter is used to request the same channel separation from certain data sets as was requested earlier in the job step. The AFF parameter is coded on a DD statement.

alias: An alternate name that may be used to refer to a member of a partitioned data set.

allocation: The process of assigning a resource to a job step.

automatic restart: A restart of a job after a job step abnormally terminates. The restart takes place during the current run, that is, without resubmitting the job.

automatic volume recognition (AVR): A feature that allows the operator to mount labeled volumes on available input/output devices before those volumes are required by a job step.

auxiliary storage: Data storage other than main storage; secondary storage.

backward reference: A facility of the job control language that permits you to copy information or refer to DD statements that appear earlier in the job.

catalog:
1. The collection of all data set indexes maintained by data management. Each entry contains a data set name and volume and unit information about the data set.
2. To place an entry for a data set in the catalog. To specify this on a control statement, code DISP=(status,CATLG) on the DD statement that defines the data set you want cataloged. A cataloged data set is easy to retrieve.

cataloged data set: A data set that is represented in an index or hierarchy of indexes in the system catalog, the indexes provide the means for locating the data set.

cataloged procedure: A set of JCL statements that has been assigned a name and placed in a partitioned data set known as the procedure library. To use a cataloged procedure, code the procedure name on an EXEC statement.

checkpoint/restart: A facility of the operating system that can minimize time lost in reprocessing a job step that abnormally terminated. The CHKPT macro instruction, the RESTART parameter on the JOB statement, and the RD parameter on the JOB or EXEC statement are associated with this facility.

checkpoint restart: A restart within a job step. The restart may be automatic (depending on an eligible completion code and the operator's consent) or deferred, where deferred involves resubmitting the job and coding the RESTART parameter on the JOB statement of the resubmitted job.

CLASS parameter: This parameter is used to assign a job class to your statement and is coded on a JOB statement. In multiprogramming systems, jobs within a job class are initiated according to their priority numbers.

command statement: A JCL statement that is used to issue commands to the system through the input stream.

comment statement: A JCL statement used to contain information that may be helpful to yourself or another person that may be running your job or reviewing your output listing.

concatenated data sets: A group of input data sets that are treated as one data set for the duration of a job step.

COND parameter: This parameter is used to test return codes issued by the processing programs; any test that is satisfied causes the job to be terminated or a job step to be bypassed. The COND parameter is coded on a JOB or EXEC statement.

control volume: A volume that contains one or more indexes of the catalog.

data control block (DCB): A control block used to contain certain attributes required by an access method to store or retrieve a

data set. The DCB parameter is one means of supplying attributes.

DATA parameter: This parameter is coded as the first parameter on a DD statement that precedes data in the input stream when the data contains job control statements.

data set: An organized collection of related data in one of several prescribed arrangements. The information required to store and retrieve this data is defined on a DD statement.

data set control block: A data set label for a data set on a direct access volume.

data set label: A collection of information that describes the attributes of a data set. The data set label for a data set is normally on the same volume as the data set it describes.

DCB: See data control block.

DD statement (data definition): A JCL statement that defines a data set that is being created or retrieved in ajob step. DD statements follow an EXEC statement.

ddname (data definition name): A name assigned to a DD statement. This name corresponds to the ddname appearing in a data control block.

DCB parameter: This parameter is used to supply attributes about the data set that are needed to complete the data control block. The DCB parameter is coded on a DD statement.

DDNAME aprameter: This parameter is used to postpone the definition of a data set until later in the same job step and is coded on a DD statement.

deferred restart: A restart that is performed when a job is resubmitted and the RESTART parameter is coded on the JOB statement of the resubmitted job.

delimiter statement: A JCL statement used to mark the end of data. The characters /* appear in columns 1 and 2 of this JCL statement.

device type: A number that corresponds to a type of input/output device. Coding the device type in the UNIT parameter of the DD statement is one way of indicating what input/output device you want allocated to a job step.

direct access device: An auxiliary storage device in which the access time is effectively independent of the location of the data set.

direct data set: A data set whose records are in random order on a direct access volume. Each record is stored or retrieved according to its actual address or its address relative to the beginning of the data set.

directory: A series of 256-byte records at the beginning of a partitioned data set that contains an entry for each member in the data set.

DISP parameter: This parameter is used to describe the status of the data set and indicates what should be done with the data set after termination of the job step that processes it, or at the end of the job. The DISP parameter is coded on a DD statement.

dispatching priority: The number assigned to a task, which in a multitask environment, determines the order in which the tasks may use main storage and CPU resources.

DPRTY parameter: This parameter is used to assign a dispatching priority to a job step and is coded on an EXEC statement.

DSN parameter: This parameter is used to assign a name to a new data set or to identify an existing data set and is coded on a DD statement. Coding DSN is the same as coding DSNAME.

DSNAME parameter: This parameter is used to assign a name to a new data set or to identify an existing data set and is coded on a DD statement. Coding DSNAME is the same as coding DSN.

DUMMY parameter: This parameter is used to tell the system that the processing program should be executed, but no input or output operations should be performed on a particular data set. The DUMMY parameter is coded as the first parameter on a DD statement.

dynamic storage: That portion of main storage that is subdivided into partitions or regions for use by the programs associated with job steps and some system tasks.

EXEC (execute) statement: A JCL statement that marks the beginning of a job step and identifies the program to be executed or the cataloged procedure to be used.

extent: A contiguous area of storage on a direct access volume in which a data set resides. A data set may reside in more than one area of storage on one or more volumes.

F format: A data set format in which the logical records are the same length.

fixed-length record: A record having the same length as all other records with which it is logically or physically associated.

generation data group: A collection of data sets that are kept in chronological order; each data set is called a generation. The DSNAME parameter is used to define the generation you are creating or retrieving.

generation data set: One generation of a generation data group.

group name: A 1- to 8-character name that identifies a device or a collection of devices. Coding a group name in the UNIT parameter is one way of indicating what type of input/output device you want allocated to a job step.

index:
1. A table in the catalog used to locate data sets.
2. A table used to locate the records of an indexed sequential data set.

indexed sequential data set: A data set on a direct access volume whose records contain a key portion and the location of each record depends on the contents of the key portion. The location of each record is computed through the use of an index.

initiation: The process of selecting a job step for execution and allocating input/output devices for the job step.

input job queue: A queue of summary information of JCL statements maintained by the job scheduler, from which it selects the jobs and job steps to be processed.

input stream: The sequence of JCL statements and data submitted to the operating system on an input device especially activated for this purpose by the operator.

in-stream procedure: A set of JCL statements in the form of cards that have been placed in the input stream of a card reader. An in-stream procedure can be executed any number of times during the job in which it appears.

JCL: A high-level programming language used to code JCL statements which describes a job to the operating system and inform the system of how the job is to be processed.

JCL Statement: Any one of the control statements in the input stream that identifies a job or defines its requirements.

job: A total processing application that consists of one or more processing programs required to perform the application. A job is identified by a JOB statement.

JOB statement: A JCL statement that marks the beginning of a job, and when jobs are stacked in the input stream, marks the end of the JCL statements for the preceding job.

job class: An alphabetic character of A through O that characterizes the type of job you are submitting. Each job class is defined by the installation; you indicate the type of job you are submitting in the CLASS parameter on the JOB statement. In multiprogramming systems, jobs within a job class are initiated according to their priority numbers.

job control language: See JCL.

job control statement: See JCL statement.

job library: See private library.

job management: A general term that collectively describes the functions of the job scheduler and master scheduler.

job processing: The reading of JCL statements and data from an input stream, the initiating of job steps defined in these statements, and the writing of system output messages.

job scheduler: A control program function that controls input streams and system output, obtains input/output devices for jobs and job steps, and regulates the use of the computing system by jobs. The job scheduler is made up of the reader/interpreter, initiator/terminator, and output writer.

job step: The unit of work associated with one processing program or one cataloged procedure, and related data. A job consists of one or more job steps.

JOBLIB: A special ddname that when specified on a DD statement indicates to the system that you are defining a private library.

jobname: The name assigned to a JOB statement; it identifies the job to the system.

K: 1024 bytes.

keyword: A symbol that identifies a parameter or subparameter.

keyword parameter: A parameter that consists of a keyword followed by an equal sign, followed by a single value or a list of subparameters. Keyword parameters must follow positional parameters in the operand field of a JCL statement, but the keyword parameters may appear in any order.

LABEL parameter: This parameter is used: (1) to describe the data set label associated with the data set; (2) to describe the sequence number of a data set that does not reside first on a reel; (3) to assign a retention period; (4) to assign password protection; and (5) to override the OPEN macro instruction (BSAM only). The LABEL parameter is coded a DD statement.

library:
1. In general, a collection of information associated with a particular use, and the location of which is identified in a directory of some type. In this context, see link library, private library, system library.
2. Any partitioned data set.

limit priority: A priority associated with every task in an MVT system, representing the highest dispatching priority that the task may assign to itself or to any of its subtasks.

link library: A partitioned data set named SYS1.LINKLIB. Each member is a processing program and is called in the PGM parameter on the EXEC statement or in the ATTACH, LINK, LOAD, and XCTL macro instructions.

logical record: A record that is defined in terms of the information it contains rather than its physical traits. You may have to specify the length of the logical record to complete the data control block; one way to specify this is in the LRECL subparameter of the DCB parameter.

main storage: All addressable storage from which instructions can be executed or from which data can be loaded directly into registers.

main storage hierarchy support: An option that divides main storage into two blocks known as hierarchies; hierarchy 0 is assigned to processor storage and hierarchy 1 to the IBM 2361 Core Storage unit.

master scheduler: The part of the control program that responds to operator commands and returns required information.

member: An independent, sequentially organized data set identified by a unique name in a data set directory. A collection of such members is called a partitioned data set.

MFT (multiprogramming with a fixed number of tasks): A control program that provides priority scheduling of a fixed number of tasks. A priority scheduler is used in MFT.

MSGLASS parameter: This parameter is used to assign an output class to the system messages for your job and is coded on a JOB statement.

MSGLEVEL parameter: This parameter is used to indicate what JCL statements and allocation/termination messages you want displayed as output from your job. This parameter is coded on a JOB statement.

multiprogramming: Executing more than one job step concurrently.

mutually exclusive: The term applied to two parameters that cannot be coded on the same JCL statement.

MVT (multiprogramming with a variable number of tasks): A control program that provides priority scheduling of a variable number of tasks. A priority scheduler is used in MVT.

MVT with Model 65 multiprocessing: An extension to MVT. This control program is used with the Model 65 Multiprocessing System.

M65MP: See MVT with Model 65 multiprocessing.

name: A 1- to 8-character term, beginning with an alphabetic or national (#, @, $) character, that identifies a data set, a JCL statement, a program, or a cataloged procedure.

nonspecific volume request: A request for volumes that allows the system to select suitable volumes. This type of request can only be made when defining an output data set.

nontemporary data set: A new data set that exists after the job that created it terminates.

null statement: A JCL statement used to mark the end of a job's control statements and data.

OUTLIM parameter: This parameter is used to specify the maximum number of logical records you want included for the output data set being routed through the output stream. The OUTLIM parameter is coded on a DD statement that must also contain the SYSOUT parameter.

output class: An alphabetic or numeric character that characterizes the type of output data to be written to a unit record device. Each output class is defined by the installation. For system messages, you indicate the type of output data in the MSGCLASS parameter on a JOB statement; for output data sets, you indicate the type of output data in the SYSOUT parameter on a DD statement.

output listing: A form that is printed at the end of your job that may contain JCL statements used by your job, diagnostic messages about your job, data sets created by your job, or a dump.

output stream: Diagnostic messages and other output data issued by the operating system or the processing program on output devices especially activated for this purpose by the operator.

output writer: A part of the job scheduler that writes output data sets onto a system output device, independently of the programs that produced the data sets.

PARM parameter: This parameter is used to supply a processing program with information it requires at the time the program is executed and is coded on an EXEC statement.

parameter: A variable that is given a constant value for a specific process or purpose.

partition: In systems with MFT, a subdivision of the dynamic area of main storage set aside for a job step or a system task.

partitioned data set: A collection of independent groups of sequential records on a direct access volume, each of which is called a member. Each member has unique name and is listed in a directory at the beginning of the data set.

PCP (primary control program): A control program that provides sequential scheduling of jobs.

PEND statement: A JCL statement used to mark the end of an in-stream procedure.

PGM parameter: This parameter appears as the first parameter on an EXEC statement when you want to execute a particular program.

physical record: A record that is defined in terms of physical qualities rather than by the information it contains (logical record).

positional parameter: A parameter that must precede all keyword parameters in the operand field of a JCL statement. Positional parameters must appear in a specified order.

primary quantity: The initial amount of space on a direct access volume that you request in the SPACE, SPLIT, or SUBALLOC parameter.

priority: A rank assigned to each job step that determines the order in which job steps are selected for execution and requests for resources are satisfied.

priority scheduler: A scheduler used with MFT and MVT. Priority schedulers process complete jobs according to their initiation priority within job classes. Priority schedulers can accept input data from more than one input stream.

private: The term applied to a mounted volume that the system cannot allocate to an output data set for which a nonspecific volume request is made. A private volume is demounted after its last use in a job step.

private library: A partitioned data set whose members are not used often enough to warrant their inclusion in the link library. To execute a program that resides on a private library, you must define that library on a DD statement that has the ddname JOBLIB or STEPLIB.

PROC parameter: This parameter appears as the first parameter on an EXEC statement when you want to call a particular cataloged or in-stream procedure.

PROC statement: A JCL statement used in cataloged or in-stream procedures. It can be used to assign default values for symbolic parameters contained in a procedure. For in-stream procedures, it is used to mark the beginning of the procedure.

procedure step: That unit of work associated with one processing program and related data within a cataloged procedure. A cataloged procedure consists of one or more procedure steps.

processing program: Any program capable of operating in the problem program mode. This includes IBM-distributed language processors, application programs, service and utility programs, and user-written programs.

PRTY parameter: This parameter is used to indicate the job's initiation priority within its job class and is coded on a JOB statement.

public: The term applied to a mounted volume that the system can allocate to an output data set for which a nonspecific volume request is made. A public volume remains mounted until the device on which it is mounted is required by another volume.

qualified name: A data set name that is composed of multiple names separated by periods (e.g., A.B.C.). For a cataloged data set, each name but the last corresponds to an index level in the catalog.

RD parameter: This parameter is used to define the type of restart that can occur and is coded on a JOB or EXEC statement.

reader/interpreter: A job scheduler function that analyzes an input stream of JCL statements.

record: A general term for any unit of data that is distinct from all others.

region: In systems with MVT, a subdivision of the dynamic area of main storage set aside for a job step or a system task. You can specify in the REGION parameter, of the JOB statement or EXEC statement, how large this area of main storage should be.

REGION parameter: This parameter is used to specify how much contiguous main storage is required to execute a job step and can be coded on a JOB or EXEC statement. If main storage hierarchy support is included in the system, the REGION parameter is also used to identify the hierarchy or hierarchies in which the storage is to be allocated.

resource: Any facility of the computing system or operating system required by a job or task and inlcudes main storage, input/output devices, the CPU, data sets, and control and processing programs.

restart: The process of resuming a job after it abnormally terminates. When a restart is performed, processing is continued either at the beginning of a job step that caused the abnormal termination or at a checkpoint within this job step.

RESTART parameter: This parameter is used to identify the step, or the step and the checkpoint within the step at which execution of a job is to be resumed and is coded on the JOB statement of a resubmitted job that is to use the checkpoint/restart facilities.

ROLL parameter: This parameter is used to specify a job step's ability to be rolled out or to cause rollout of another job step and is coded on a JOB or EXEC statement.

rollout/rollin: An optional MVT control program feature that allows the temporary assignment of additional main storage to a job step.

scheduler: See job scheduler.

secondary quantity: The additional amount of space on a direct access volume that you want allocated to a data set if the primary quantity requested in the SPACE, SPLIT, or SUBALLOC parameter is not sufficient.

secondaty storage: See auxiliary storage.

SEP parameter: This parameter is used to request channel separation from specific data sets defined earlier in the job step and can be coded on a DD statement.

sequential data set: A data set whose records are organized on the basis of their successive physical positions, such as they are on magnetic tape.

SPACE parameter: This parameter is used to indicate how much space should be allocated on a direct access volume for a new data set and is coded on a DD statement.

specific volume request: A request for volumes that informs the system of the volume serial numbers.

SPLIT parameter: This parameter is used to allocate space to two or more new data sets that are to share cylinders. The SPLIT parameter is coded on a DD statement.

STEPLIB: A special ddname that when specified on a DD statement indicates to the system that you are defining a private library.

stepname: The name assigned to an EXEC statement; it identifies a job step within a job.

step restart: A restart at the beginning of a job step that abnormally terminates. The restart may be automatic (depending on

an eligible completion and the operator's consent) or deferred, where deferred involves resubmitting the job and coding the RESTART parameter on the JOB statement of the resubmitted job.

SUBALLOC parameter: This parameter is used to place a series of a new data sets in one area of contiguous space on a direct access volume and in a certain sequence. The SUBALLOC parameter is coded on a DD statement.

subparameter: One of the items of variable information that follows a keyword parameter and can be either positional or keyword.

storage volume: The main function of a storage volume is to contain nontemporary data sets for which a nonspecific volume request was made and PRIVATE was not coded in the VOLUME parameter. A direct access volume becomes a storage volume when so indicated in a MOUNT command or in a member of SYS1.PARMLIB named PRESRES.

symbol: In the IBM System/360 Operating System, any group of eight or less alphameric and national characters that begins with an alphabetic or national (#,@,$) character.

symbolic parameter: A symbol preceded by an ampersand that appears in a cataloged procedure. Values are assigned to symbolic parameters when the procedure in which they appear is called.

SYSABEND: A special ddname that when specified on a DD statement tells the system you are defining a data set on which a dump can be written if the step abnormally terminates. The dump provided includes the system nucleus, the processing program storage area, and a trace table.

SYSCTLG: The name of a system data set that contains the name and location of cataloged data sets.

SYSCHK: A special ddname that when specified on a DD statement that precedes the first EXEC statement in the job tells the system you are defining a data set that contains checkpoint entries. This DD statement is included in a job that is being resubmitted for execution and execution is to begin at a particular checkpoint.

SYSIN: A name conventionally used as the data definition name of a data set in the input stream.

SYSOUT parameter: This parameter is used to assign an output class to an output data set and can be coded on a DD statement.

system data sets: The data sets that make up the IBM System/360 Operating System.

system generation: The process of producing an operating system made up of standard and optional components.

system input device. A device specified as a source of an input stream.

system library: A partitioned data set which is one of the collection of system data sets at an installation.

system management facilities: An optional control program feature that provides the means of gathering and recording information that can be used to evaluate system usage.

system messages: Messages issued by the system that pertains to a problem program. These messages appear on an output listing and may include such messages as error messages, disposition messages, and allocation/de-allocation messages.

system output device: An output device, shared by all jobs, onto which specified output data is written.

SYSUDUMP: A special ddname that when specified on a DD statement tells the system you are defining a data set on which a dump can be written if the step abnormally terminates. The dump provided is the processing program storage area.

SYS1.LINKLIB: The name of a partitioned system data set that contains the IBM-supplied processing programs and part of the nonresident portion of the control program. It may also contain user-written programs.

SYS1.PROCLIB: The name of a partitioned system data set that contains cataloged procedures.

SYS1.SYSJOBQE: A system data set that contains information about the input and output streams, and contains the input and output queues.

task: The smallest unit of work that can be performed under the control program.

temporary data set: A new data set that is created and deleted in the same job.

termination: The process of performing disposition processing, as specified in the DISP parameter, de-allocating input/output devices, and supplying control information for writing job output on a system output unit.

TIME parameter: This parameter is used to assign a time limit on how long the job or a particular job step can use the CPU and is coded on a JOB or EXEC statement, or both.

time-slicing: The sharing or the CPU by certain tasks for an equal, predetermined length of time.

TYPRUN parameter: This parameter is used to hold for execution until the operator issues a RELEASE command and is coded as TYPRUN=HOLD on a JOB statement.

UCS parameter: This parameter is used to describe the character set you want to use for printing an output data set on a 1403 printer. The UCS parameter is coded on a DD statement.

unit address: A 3-byte number, made up of the channel, control unit, and unit numbers, that identifies a particular device. Coding a unit address in the UNIT parameter is one way of indicating what input/output device you want allocated to the job step.

UNIT parameter: This parameter is used to describe what device and how many devices you want assigned to a data set. The UNIT parameter can be coded on a DD statement.

V format: A data set format in which logical records are of varying length and include a length indicator; and in which V format logical records may be blocked, and each block containing a block length indicator.

VOL parameter: This parameter is used to identify the volume(s) on which a data set resides or will reside and is coded on a DD statement. Coding VOL is the same as coding VOLUME.

volume: That portion of an auxiliary storage device that is accessible to a single read/write mechanism.

VOLUME parameter: This parameter is used to identify the volume(s) on which a data set resides or will reside and is coded on a DD statement. Coding VOLUME is the same as coding VOL.

volume table of contents (VTOC): A table in a direct access volume thatdescribes each data set on the volume.

Index

Indexes to systems reference library manuals are consolidated in the publication IBM System/360 Operating System: Systems Reference Library Master Index, Order No. GC28-6644. For additional information about any subject listed below, refer to other publications listed for the same subject in the Master Index.

Where more than one page reference is given, the major reference is first.

YOUR COMMENTS, PLEASE . . .

This manual is part of a library that serves as a reference source for systems analysts, programmers and operators of IBM systems. Your answers to the questions on the back of this form, together with your comments, will help us produce better publications for your use. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Note: Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.

Fold Fold

FIRST CLASS
PERMIT NO. 81
POUGHKEEPSIE, N.Y.

BUSINESS REPLY MAIL

NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY ...

IBM Corporation
P.O. Box 390
Poughkeepsie, N.Y. 12602

Attention: Programming Systems Publications
Department D58

Fold Fold

IBM

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
[USA Only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]

Cut Along Line

READER'S COMMENT FORM

IBM System/360 Operating System:
Job Control Language User's Guide

Order No. GC28-6703-0

- Is the material:

	Yes	No
Easy to read?	☐	☐
Well organized?	☐	☐
Complete?	☐	☐
Well illustrated?	☐	☐
Accurate?	☐	☐
Suitable for its intended audience?	☐	☐

- How did you use this publication?
  - ☐ As an introduction to the subject
  - ☐ For additional knowledge

  Other .....

- Please check the items that describe your position:
  - ☐ Customer personnel
  - ☐ IBM personnel
  - ☐ Manager
  - ☐ Systems Analyst
  - ☐ Operator
  - ☐ Programmer
  - ☐ Customer Engineer
  - ☐ Instructor
  - ☐ Sales Representative
  - ☐ Systems Engineer
  - ☐ Trainee

  Other .....

- Please check specific criticism(s), give page number(s), and explain below:
  - ☐ Clarification on page(s) .....
  - ☐ Addition on page(s) .....
  - ☐ Deletion on page(s) .....
  - ☐ Error on page(s) .....

Explanation:

- Thank your for your cooperation. No postage necessary if mailed in the U.S.A.

YOUR COMMENTS, PLEASE . . .

This manual is part of a library that serves as a reference source for systems analysts, programmers and operators of IBM systems. Your answers to the questions on the back of this form, together with your comments, will help us produce better publications for your use. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Note: Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.

Fold Fold

FIRST CLASS
PERMIT NO. 81
POUGHKEEPSIE, N.Y.

BUSINESS REPLY MAIL

NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY ...

IBM Corporation
P.O. Box 390
Poughkeepsie, N.Y. 12602

Attention: Programming Systems Publications
Department D58

Fold Fold

IBM

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
[USA Only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]

Cut Along Line

System/360 OS JCL User's Guide (S360-36) Printed in U.S.A. GC28-6703-0

System/360 OS JCL User's Guide (S360-36) Printed in U.S.A. GC28-6703-0

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
[USA Only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]