File No. S360-48
Form C28-6539-4

OS

IBM Systems Reference Library

IBM System/360 Operating System

Job Control Language

This publication describes the facilities of the Job Control Language, and illustrates how to use these facilities in various applications. Information coded by programmers on job control statements is used by the System/360 Operating System to initiate and control the processing of jobs. Information on MVT contained herein is for planning purposes.

Fifth Edition (March 1967)

This publication is a major revision of Form C28-6539-3 and obsoletes it and prior editions. Significant changes have been made throughout; this edition should be reviewed in its entirety.

Specifications contained herein are subject to change from time to time. Any such change will be reported in subsequent revisions or Technical Newsletters.

This publication was prepared for production using an IBM computer to update the text and to control the page and line format. Page impressions for photo/offset printing were obtained from an IBM 1403 Printer using a special chain.

Copies of this and other IBM publications can be obtained through IBM Branch Offices.

A form for reader's comments appears at the back of this publication. It may be mailed directly to IBM. Address any additional comments concerning this publication to the IBM Corporation, Programming Systems Publications, Department D58, PO Box 390, Poughkeepsie, N. Y. 12602.

The purpose of this publication is to explain how to use the Job Control Language to initiate and control the processing of jobs. To fulfill this purpose, the publication is divided into two sections: "Section 1: Job Control Statements" and "Section 2: A Guide to Using the Job Control Language."

Section 1 describes the language specifications and facilities in detail. It contains a chapter on each of the job control statements, and individual topics on the control statement parameters. This section need only be read by those who desire a thorough knowledge of the Job Control Language and its facilities.

Section 2 defines and illustrates the control statement and parameter requirements for applications of the language. Chapters and topics are organized by application. If you are responsible for coding job control statements, you should be familiar with the text and examples in Section 2. You need not concern yourself with Section 1, unless you want more information on a particular statement or parameter.

Before proceeding to either section, you should read the introduction to familiarize yourself with job management in the operating system and the terminology used in this book. The appendixes contain information gathered from other publications to minimize cross-referencing, and foldout diagrams of control statements and parameters. The list of examples, located at the front of the book, facilitates quick referencing of sample control statements in Section 2.

Before you read this publication, you should understand the concepts and terminology introduced in the prerequisite publications listed below. In addition, the text refers to several other publications for detailed discussions beyond the scope of this book.

PREREQUISITE PUBLICATIONS

IBM System/360 Operating System: Introduction, Form C28-6534

IBM System/360 Operating System: Concepts and Facilities, Form C28-6535

"Section 2: Data Management Services" of the publication IBM System/360 Operating System: Supervisor and Data Management Services, Form C28-6646

PUBLICATIONS TO WHICH THE TEXT REFERS

IBM System/360 Operating System: System Programmer's Guide, Form C28-6550

IBM System/360 Operating System: Utilities, Form C28-6586

IBM System/360 Operating System: Operator's Guide, Form C28-6540

IBM System/360 Operating System: Supervisor and Data Management Macro-Instructions, Form C28-6647

IBM System/360 Operating System: Storage Estimates, Form C28-6551

EXAMPLES

FIGURES

TABLES

Communication with the scheduling components of an operating system is achieved with control cards submitted to the system by a programmer or operator. Control cards provide the operating system user with a means of influencing the scheduling of work, the allocation of system resources, and system performance, in addition to transmitting control information. A medium that is this flexible can itself be considered a high-level programming language. Job scheduling within the System/360 Operating System (hereinafter called the operating system or the system) is controlled by such a medium, the Job Control Language; information coded in the Job Control Language is submitted to the operating system at the time a processing program is executed. Because the information can be submitted on devices other than a card reader, the elements of the language are called "control statements" instead of control cards.

Through the Job Control Language, you can provide job and program information, data characteristics, and device requirements at the time a program is executed rather than when you assembled or compiled it. Other facilities of the language allow you to:

- Copy existing data set names, control statements, and control blocks with a backward-reference facility to reduce recoding.
- Retrieve data sets by name using the system catalog, eliminating the necessity of identifying the unit type and volume serial numbers.
- Optimize use of channels, units, volumes, and direct-access space.
- Pass data sets used by more than one step from one step to another, to reduce mounting and retrieval time.
- Share data sets between two or more job steps operating independently in multiprogramming environments.

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 the job control language or those that are performed on a regular basis can be considerably simplified through the use of cataloged procedures. A cataloged procedure is a set of precoded control statements that can be retrieved by a simple name on one control statement. Any statement in the set can be temporarily modified by other control statements submitted at the time the procedure is used.

This manual includes several operating system terms, some of which are defined rigorously in prerequisite publications. They are listed and defined here for your convenience.

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.

Job: A total processing application comprising one or more related processing programs, such as a weekly payroll, a day's business transactions, or the reduction of a collection of test data.

Job Step: That unit of work associated with one processing program or one cataloged procedure, and related data. A cataloged procedure can comprise many procedure steps.

Input Stream: The sequence of control statements and data submitted to the operating system on an input unit especially activated for this purpose by the operator.

Output Stream: Diagnostic messages and other output data issued by the operating system or the processing program on output units especially activated for this purpose by the operator.

Cataloged: The quality attributed to a data set whose name and location are stored in the system catalog.

Tabulated: The quality attributed to a direct-access data set whose name is stored in the table of contents of the volume on which it resides.

Name: A 1- to 8-character alphameric term that identifies a data set, a control statement, a program, or a cataloged procedure. The first character of the name must be alphabetic.

Qualified Name: A control statement term that comprises one or more names, each qualifying the name that follows it. Levels of qualification are separated by

periods. For example, the term stepname.procstepname represents a procedure step name qualified by a job step name.

Components of Job Management

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

The job scheduler is a set of routines that read input streams, analyze control statements, allocate input/output resources, issue diagnostic messages to the programmer, and schedule job flow through the system.

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

The specific facilities available through the job scheduler and master scheduler depend on the scheduling level your installation selects during system generation. Schedulers are available at two levels -- the sequential scheduler and the more powerful priority scheduler.

Sequential schedulers process job steps one at a time in the order of their appearance in the input stream. Operating systems with a primary control program (PCP) and those that provide multiprogramming with a fixed number of tasks (MFT) use sequential schedulers.

Priority schedulers process complete jobs according to their relative priority, and can accept input data from more than one input stream. Systems that provide multiprogramming with a variable number of tasks (MVT) use priority schedulers. Appendix C lists the components of job management (Table 4) and summarizes the principal features of each scheduler (Table 5).

Communication between the operating system user and the job scheduler is effected through six job control statements (hereinafter called control statements):

1. Job Statement
2. Execute Statement
3. Data Definition Statement
4. Command Statement
5. Delimiter Statement
6. Null Statement

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

Summary of the Control Statements

The job statement (hereinafter called the JOB statement) marks the beginning of a job and, when jobs are stacked in the input stream, marks the end of the control statements for the preceding job. It may contain accounting information for use by your installation's accounting routines, give conditions for early termination of the job, and regulate the display of job scheduler messages. With priority schedulers, you can use additional parameters to assign job priority, to request a specific class for job scheduler messages, and to 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 may also provide job step accounting information, give conditions for bypassing the job step, and pass control information to a processing program. With priority schedulers, additional parameters allow you to assign a time limit for the execution of the job step and 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 input/output resources. DD statement parameters identify the data set, give volume and unit information and disposition, and describe the data set's labels and physical attributes.

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

The delimiter statement and the null statement are markers in an input stream. The delimiter statement is used when data is included in the input stream, to separate the data from subsequent control statements. The null statement can be used to mark the end of the control statements for certain jobs.

Fields in the Control Statements

Control statements contain two identifying characters (// or /*) and four fields (the name, operation, operand, and comments fields). In some of the statements, one or more of the fields are omitted. Figure 1 shows the fields in each statement.

The name field identifies the control statement so that other statements or sys-

Statement	Columns 1 and 2	Fields
Job	//	name JOB operand[1] comments[1]
Execute	//	name[1] EXEC operand comments[1]
Data Definition	//	name[1] DD operand comments[1]
Command	//	operation(command) operand comments[1]
Delimiter	/*	
Null	//	

[1]Optional

Figure 1. Control Statement Fields

tem control blocks can refer to it. It can range from one to eight characters in length, and can contain any alphameric characters. However, the first character of the name must be alphabetic, and must begin in column 3.

The operation field specifies the type of control 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.

The comments field can contain any information deemed 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.

Identifying characters and fields are contained in columns 1 through 71 of the control statement. If the total number of characters exceeds 71, the excess characters can be coded on one or more succeeding statements.

Reference:

- Information on which characters can be coded on control statements and rules for continuing control statements are contained in the introduction to Section 2 of this publication.

Parameters in the Operand Field

The operand field is made up of two types of parameters: one type is characterized by its position in the operand field in relation to other parameters (a positional parameter); the other type 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 assume the form of a list of several items (subparameters) of information.

Positional parameters must be coded first in the operand field in a specific order. 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 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.

Backward References

Many control statement parameters permit you to use a backward-reference facility to fill in information. This 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.

If the DD statement that is the subject of the reference occurs in a cataloged procedure, you must give both the name of the job step that calls the procedure and the name of the procedure step that contains the DD statement, i.e., *.stepname.procstepname.ddname.

JOB STATEMENT

The JOB statement precedes all other statements in the job. It must contain a valid job name 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. To follow the flow of parameters in the JOB statement, turn to Appendix E and fold out Chart 1 while reading this chapter.

Identifying the Job (jobname)

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

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

```
CLOCK       Q
JOBNAMES    T
            A
```

If you find it necessary to use one of these terms as a jobname, you should inform the operator to enclose it in apostrophes if he uses it in a command statement. For example, if you have assigned the jobname CLOCK to a job and the operator wishes to display the status of the job, he must issue a command stating DISPLAY 'CLOCK'. If the apostrophes were omitted, he would get the usual time-of-day display resulting from a DISPLAY CLOCK command.

Supplying Job Accounting Information

For job accounting purposes, the JOB statement can be used to supply information to your installation's accounting procedures. To supply job accounting information, code the positional parameter

```
┌-----------------------------------------┐
|(acct#,additional accounting information)|
└-----------------------------------------┘
```

first in the operand field. 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. As a system generation option with sequential schedulers, the account number can be established as a required subparameter. With priority schedulers, the requirement can be established with a cataloged procedure for the input reader. Otherwise, the account number is considered optional.

Nctes:

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

Reference:

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

Identifying the Programmer

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

Notes:

- The number of characters in the name cannot exceed 20.
- If the name contains special characters other than periods, it must be enclosed in apostrophes. If the special characters include apostrophes, each must be

shown as two consecutive apostrophes, e.g., 'T.O''NEILL'.

- If the job accounting information is not coded, you must indicate its absence by coding a comma preceding the programmer's name.
- If neither job accounting information nor programmer's name is present, you need not code commas to indicate their absence.

Reference:

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

Displaying All Control Statements (MSGLEVEL)

As part of the output for every job, the job scheduler displays the JOB statement, incorrect control statements, and associated diagnostic messages. In addition to this usual output, you can request a display of all the control statements in the job. To receive this additional output, code the keyword parameter

```
MSGLEVEL=1
```

in the operand field of the JOB statement.

If you omit the MSGLEVEL parameter, or code MSGLEVEL=0, only the JOB statement, incorrect control statements, and associated diagnostic messages are displayed.

Note:

- If an error occurs on a control statement that is continued onto one or more cards, only one of the continuation cards is printed with the diagnostics.

Specifying Conditions for Job Termination (COND)

To eliminate unnecessary use of computing time, you might want to base the continuation of a job on the successful completion of one or more of its job steps. At the completion of each job step, the processing program passes a number to the job scheduler as a return code. The COND parameter provides you with the means to test each return code as many as eight times. If any one of the tests is satisfied, subsequent steps are bypassed and the job is terminated.

To specify conditions for job termination, code the keyword parameter

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

in the operand field of the JOB statement. Replace the terms "code" with any decimal number from 0 through through 4095. Replace the terms "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 omit the COND parameter, no return code tests are performed.

Note:

- If you want to make only one return code test, you need not code the outer parentheses, e.g., COND=(8,EQ).

Assigning Job Priority (PRTY)
(Pricrity Schedulers Only)

To assign a priority other than the default job priority (as established in the input reader procedure), you must code the keyword parameter

```
PRTY=n
```

in the operand field of the JOB statement. Replace the letter "n" with a decimal number from 0 through 14 (the highest priority number is 14).

If you omit the PRTY parameter, the default jcb priority is assigned to the job.

Requesting a Message Class (MSGCLASS)
(Priority Schedulers Only)

With the quantity and diversity of data in the output stream, your installation may want to separate different types of output

data into different classes. Each class is directed to an output writer associated with a specific output unit. The MSGCLASS parameter allows you to route all messages issued by the job scheduler to an output class other than the normal message class, A. To choose such a class, code the keyword parameter

```
MSGCLASS=x
```

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

If you omit the MSGCLASS parameter, or code MSGCLASS=A, job scheduler messages are routed to the standard output class, A.

Reference:

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

Specifying Main Storage Requirements for a Job (REGION)
(Priority Schedulers Only)

For jobs that require an unusual amount of main storage, the JOB statement provides you with the REGION parameter. Through this parameter, you can specify the maximum amount of main storage to be allocated to the job. This figure must take into account the system components required by your installation.

To specify a region size, code the keyword parameter

```
REGION=nnnnnK
```

in the operand field of the JOB statement. Replace the term "nnnnn" with the number of 1024-byte areas you want allocated to the job, e.g., REGION=51K. This number can range from one to five digits.

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

Note:

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

Reference:

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

EXEC STATEMENT

The EXEC statement is the first statement in each job step and procedure step. It must contain the word EXEC in its operation field. It is followed in the input stream by DD statements and data that pertain to the step.

The principal function of the EXEC statement is to identify the program to be executed or the cataloged procedure to be used. All other parameters in the operand field are optional. To follow the flow of parameters in the EXEC statement, turn to Appendix E and fold out Chart 1 while reading this chapter.

Identifying the Step (stepname)

The stepname identifies a job step within a job. It must satisfy the positional, length, and content requirements for a name field. You must specify a stepname if later control statements refer to the step, or if the step is going to be part of a cataloged procedure. Each stepname in a job or procedure must be unique.

Identifying the Program (PGM) or Procedure (PROC)

Processing programs are members of special partitioned data sets called libraries. Programs can reside in three types of libraries:

1. Temporary libraries
2. The system library
3. Private libraries

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

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

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

in the first positions of the EXEC statement's operand field. Replace the terms "stepname" and "ddname" with the names of the job step and the DD statement, respectively, where the temporary library is created.

Note:

- If the temporary library is created in a cataloged procedure step, you must include the procedure step name, i.e., PGM=*.stepname.procstepname.ddname.

2. The system library is a partitioned data set named SYS1.LINKLIB that stores frequently used programs. To execute a program that resides in the system library, code

```
PGM=progname
```

in the first positions of the operand field. Replace the term "progname" with the member name or alias associated with the program.

3. Private libraries are partitioned data sets that store groups of programs not used sufficiently to warrant their inclusion in the system library. Private libraries are made available to a job with a special DD statement. To execute a program that resides in a private library, code

```
PGM=progname
```

in the first positions of the operand field. Replace the term "progname" with the member name or alias associated with the program.

Reference:

- To make a private library available to a job, see the chapter titled "Additional DD Statement Facilities" in Section 2 of this publication.

Note:

- 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.

Instead of executing a particular program, a job step may use a cataloged procedure. A cataloged procedure can contain control statements for several steps, each of which executes a particular program. Cataloged procedures are members of a library named SYS1.PROCLIB. To request a cataloged procedure, code

```
┌──────────────────────────────────────┐
│            procedure name            │
└──────────────────────────────────────┘
```

first in the EXEC statement's operand field. Replace the term "procedure name" with the member name associated with the cataloged procedure.

Notes:

- If you wish to note that you are executing a cataloged procedure, you can obtain the same result by coding PROC=procedure name, instead of just the procedure name.

- 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.

Reference:

- For detailed information on using and modifying cataloged procedures, see the chapter "Using Cataloged Procedures" in Section 2 of this publication.

Specifying Conditions for Bypassing the Job Step (COND)

The execution of certain job steps is often based on the success or failure of preceding steps. The COND parameter provides you with the means to make as many as eight tests on return codes issued by preceding job steps or cataloged procedure steps. If any one of the tests is satisfied, the job step is bypassed.

To specify conditions for bypassing a job step, code the keyword parameter

```
┌──────────────────────────────────────┐
│COND=((code,operator,stepname),...,   │
│                (code,operator,stepname))│
└──────────────────────────────────────┘
```

in the operand field of the EXEC statement. 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 coded COND=((20,GT,STEP1),(60,EQ,STEP2)), it would read "If 20 is greater than the return code issued by STEP1, or if STEP2 issues a return code of 60, I want this job step bypassed."

If you omit the COND parameter, no return code tests are made.

Notes:

- If you want only one test made, you need not code the outer parentheses, e.g., COND=(12,EQ,STEPX).

- If you want each return code test made on all preceding steps, you need not code the terms "stepname", e.g., COND=((20,GT),(60,EQ)).

- If the step issuing the return code is part of a cataloged procedure, you must include the procedure step name, i.e., COND=((code,operator,stepname.procstepname),...)

- When the job step uses a cataloged procedure, you can establish return code tests for a procedure step by including, as part of the keyword COND, the procedure step name, e.g., 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.

- To establish one set of return code tests for all the 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.

Passing Information to the Processing Program (PARM)

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

```
PARM=value
```

in the operand field. Replace the term "value" with up to 40 characters of data.

If you omit the PARM parameter, no control information is passed to the processing program.

Notes:

- If the value contains special characters, the information must be enclosed in single apostrophes (5-8 punches), e.g., PARM='CONTROL INFORMATION'. (The enclosing apostrophes are not considered part of the information.) If the special characters include apostrophes, each must be shown as two consecutive apostrophes, e.g., PARM='CONTROL INFORM''N'.
- If the only special character in the value is the comma, the job scheduler will accept the value enclosed in parentheses instead of apostrophes, e.g., PARM=(123,456,789).
- 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.
- 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.

Reference:

- The exact location and format of control information passed to a processing program are described under the topic titled "Program Management" in Section 1 of the publication IBM System/360 Operating System: Supervisor and Data Management Services.

Specifying Job Step Accounting Information (ACCT)

When executing a multistep job, or a job that uses cataloged procedures, you might want to charge individual job steps to separate accounting areas. To specify items of accounting information pertaining to a job step, code the keyword parameter

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

in the operand field of the EXEC statement. Replace the term "accounting information" with one or more subparameters separated by commas. The items of information are made available to your installation's job step accounting routines.

Notes:

- 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 of information, you need not enclose it in parentheses, e.g., ACCT=12345.
- If any subparameter of information contains special characters (except hyphens), you must enclose the subparameter in apostrophes (5-8 punches). The apostrophes are not considered part of the information.
- When the job step uses a cataloged procedure, you can furnish accounting information pertaining to a single procedure step by including, as part of the keyword ACCT, the procedure step name, e.g., 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.
- To furnish 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.

Reference:

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

Setting Job Step Time Limits (TIME) (Priority Schedulers Only)

To limit the computing time used by a single job step or cataloged procedure step, you might want to assign a maximum time for its completion. Such an assignment is useful in a multiprogramming environment where more than one job has access to the computing system.

To assign a time limit to a job step, code the keyword parameter

```
┌──────────────────────────────────────┐
│        TIME=(minutes,seconds)        │
└──────────────────────────────────────┘
```

in the operand field of the EXEC statement. Replace the terms "minutes" and "seconds" with the maximum number of minutes and seconds allotted for execution of the job step. The number of minutes cannot exceed 1439; the number of seconds cannot exceed 59. If the job step is not completed in this time, the entire job is terminated.

If you omit the TIME parameter, the default job step time limit (as established in the cataloged procedure for the input reader) is assumed. If the job step execution time may exceed 1439 minutes (24 hours), code TIME=1440 to eliminate job step timing.

Notes:

- If the time limit is given in minutes only, you need not code the parentheses, e.g., TIME=5.
- If the time limit is given in seconds only, you must code a comma to indicate the absence of minutes, e.g., TIME=(,45).
- When the job step uses a cataloged procedure, you can set a 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 paramters of this form as there are steps in the cataloged procedure.
- To set a 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.

Specifying Main Storage Requirements for a Job Step (REGION) (Priority Schedulers Only)

For job steps that require an unusual amount of main storage, the EXEC statement provides you with the REGION parameter. Through this parameter you can specify the maximum amount of main storage to be allocated to the associated job step. This size must take into account the system components required by your installation.

To specify a region size, code the keyword parameter

```
┌──────────────────────────────────────┐
│            REGION=nnnnnK             │
└──────────────────────────────────────┘
```

in the operand field of the EXEC statement. Replace the term "nnnnn" with the number of 1024-byte areas you want allocated to the job step, e.g., REGION=51K. This number can range from one to five digits.

If you omit the REGION parameter, the default region size (as established in the cataloged procedure for the input reader) is assumed.

Notes:

- If you have specified a REGION parameter in the JOB statement, REGION parameters in the job's EXEC statements are ignored.
- 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.
- 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.

Reference:

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

DD STATEMENT

Data sets used by processing programs must be represented by DD statements in the input stream. The DD statements pertaining to a particular job step follow the EXEC statement associated with the step. 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.

The DD statement is the final source of information that is needed to retrieve and store data. Figure 2 illustrates the sources of information and the means by which each source refers to the next.

An input/output macro-instruction, coded as part of the processing program, issues an input or output request (OPEN, CLOSE, GET, PUT, READ, WRITE). This request uses a dcbname to refer to a data control block created earlier by a DCB macro-instruction. The data control block contains information about a data set that is gathered from several sources, one of which is a DD statement whose ddname matches the ddname given in the data control block. The DD statement, the final source of information, is associated with a specific data set. It refers to the data set with a data set name.

Because of the DD statement's position in this sequence of information sources, you can specify such characteristics as buffer size, record length, and device type at the time the job step is executed, rather than when you code the processing program.

To follow the flow of parameters in the DD statement, turn to Appendix E and fold out Chart 2 while reading this chapter. Individual parameters are shown in detail in a series of figures on Chart 3 of Appendix E.

Identifying the DD Statement (ddname)

The ddname identifies the DD statement so that subsequent control statements and the data control block can refer to it. It must satisfy the position, length, and content requirements for a name field. Each ddname within a job step should be unique. If duplicate ddnames exist, all references are directed to the first such DD statement in the job step, and the second is ignored.

Note:

- Omit the ddname if the data set is concatenated with the data set defined by the preceding DD statement, or the DD statement is one of a group of DD statements that define an indexed sequential data set.

If the 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 is valid only for the duration of the job step; it does not change the procedure permanently.

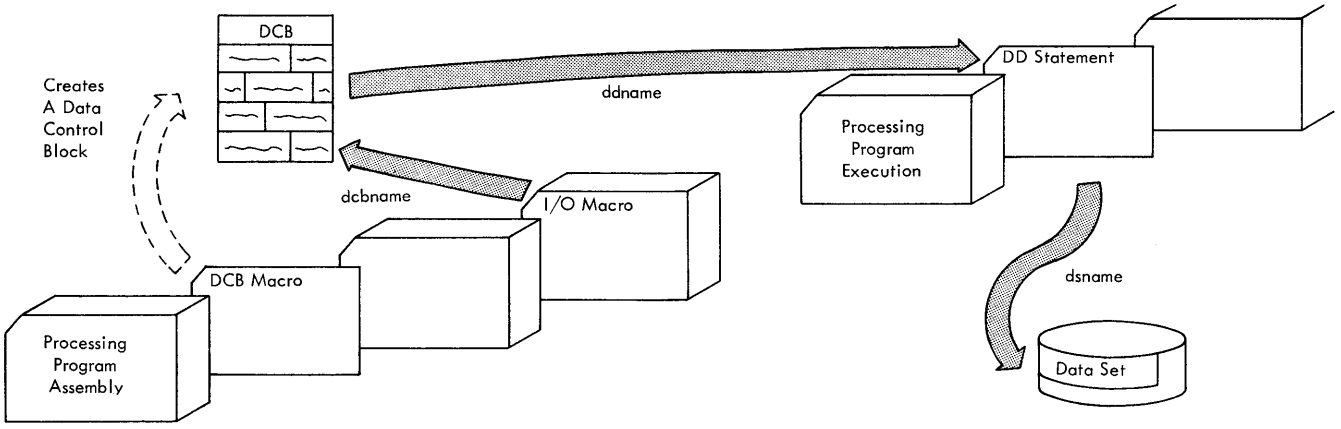

Figure 2. Data Set Information Sources

References:

- For more information on how to code the ddname when creating indexed sequential data sets, see Appendix D of this publication.
- Instructions for concatenating data sets and modifying cataloged procedures are contained in the chapters "Additional DD Statement Facilities" and "Using Cataloged Procedures" in Section 2 of this publication.

Defining Data in an Input Stream (DD * or DD DATA)

One of the ways a data set can be introduced to the system is by its inclusion in an input stream in the form of 80-byte records. A data set in an input stream is bounded by a DD statement that marks its beginning and a delimiter statement that marks its end.

To mark the beginning of the data set, code an asterisk (*) in the operand field of the DD statement that precedes it. If the data contains job control statements (statements with // in columns 1 and 2), as would be the case if the data set were a procedure being cataloged, code the word DATA in the operand field instead of the *. In both cases, do not code any other parameters.

To mark the end of the data set, insert a delimiter statement (/* in columns 1 and 2) after the last data card.

Notes:

- When using a sequential scheduler, you can include only one data set in the input stream for each job step or procedure step. This data set must be defined by the last DD statement in the step. With systems providing multiprogramming with a fixed number of tasks, the processing program that uses data in the input stream must be in the lowest priority partition.
- When using a priority scheduler, you can include more than one data set in the input stream. If you leave out the DD * statement for the first such data set, the system assigns the name SYSIN to it. The delimiter statement is not required following data sets preceded by DD *.
- Data sets in an input stream cannot contain statements having the characters /* in columns 1 and 2.

Bypassing I/O Operations on the Data Set (DUMMY)

The DUMMY parameter, a DD statement positional parameter, offers you the facility to bypass I/O operations, space allocation, and disposition of data sets referred to by the basic or queued sequential access methods. This facility can be used to suppress the writing of certain output data sets, such as assembler listings, and to update new master files with a dummy detail file. Bypassing operations on noncritical data sets also results in a saving of time when you are testing a program. To use this facility, code the word DUMMY as the first parameter in the operand field.

An attempt to read a "dummy" data set in your processing program will result in an immediate end-of-data-set exit. If you attempt to write on a dummy data set, the write request is recognized, but no data is transmitted. In addition, no device allocation, external storage allocation, or data set disposition takes place. Another way of establishing a dummy data set is by assigning it the name NULLFILE.

Postponing the Definition of a Data Set (DDNAME)

A DD statement in a cataloged procedure and, in certain cases, in an input stream, need not contain descriptive parameters. Instead, it can point to a subsequent DD statement that contains a complete description of a data set. The original data set does not assume real characteristics until the DD statement that contains the complete description is encountered. To postpone the definition of a data set in this manner, code

```
DDNAME=ddname
```

in the operand field of the DD statement. Replace the term "ddname" with the name of the DD statement that contains complete information. This feature is particularly useful when a cataloged procedure uses data in an input stream.

Reference:

- For more information and examples of usage of the DDNAME parameter, see the chapter "Using Cataloged Procedures" in Section 2 of this publication.

Routing a Data Set Through the Output Stream (SYSOUT)

Output data sets can be routed through the system output stream and handled much the same as system messages. When using a sequential scheduler, you can route a data set through the output stream by coding the keyword parameter

```
SYSOUT=A
```

in the operand field of the DD statement. Your processing program writes the data set on the system output device. A unit record or labeled tape device becomes the system output device when the operator activates it as such with a START WTR command. With systems providing multiprogramming with a fixed number of tasks, the processing program that writes the data must be in the lowest priority partition.

When you use a priority scheduler, such operations are not performed during execution of the job step. Instead, the processing program writes the data set on an intermediate direct-access device. Later the data set is routed through an output stream to a system output device. To schedule such an operation, code the keyword parameter

```
SYSOUT=x
```

in the operand field. Replace the letter "x" with an alphabetic (A-Z) or numeric (0-9) character. The letter represents one of the system output classes. Output writers route data from the output classes to system output devices. The DD statement for this data set can also include a unit specification describing the intermediate direct-access device, and an estimate of the space required. If you omit these parameters, the job scheduler provides default values as the job is read and interpreted.

When using a priority scheduler, you have two additional options with the SYSOUT parameter. If you have a special program to handle output operations, code

```
SYSOUT=(x,progname)
```

Replace the term "progname" with the member name associated with your program. The program must reside in the system library. If you want the data set printed or punched on a specific type of output form, code

```
SYSOUT=(x,,form#)
```

Replace the term "form#" with the 4-digit form number to be used. This form number is used to instruct the operator in a message issued at the time the data set is to be printed.

Note:

- If you wish to specify both an output program and a form number, code SYSOUT=(x,progname,form#).

Identifying the Data Set (DSNAME)

Data sets used by a processing program are identified with the DSNAME parameter. You need not code this parameter if the data set is temporary; the system automatically assigns a name if the DSNAME parameter is omitted. You can specify this parameter in one of three ways:

1. By giving the name by which a data set is or will be cataloged or tabulated.
2. By referring to an earlier DD statement that gives its cataloged or tabulated name.
3. By giving a temporary name, if it is a temporary data set.

To supplement this discussion of the DSNAME parameter pictorially, turn to Appendix E and fold out Chart 3.

1. Data sets that will be identified in later jobs by name, or that were assigned a name in an earlier job are referred to by a cataloged or tabulated name. To name or retrieve a data set of this type, code the keyword parameter

   ```
   DSNAME=dsname
   ```

   in the operand field of the DD statement. Replace the term "dsname" with the data set's cataloged or tabulated name. If the catalog makes use of index levels, you must give a fully qualified name, e.g., A.B.LINKFILE.

   Notes:

   - The data set is assigned a dummy status if you code DSNAME=NULLFILE.
   - If the DD statement refers to a particular generation of a generation data group, you must code the

generation number in parentheses, i.e., DSNAME=dsname(number). A generation number can be 0 or a signed integer.

- If the DD statement refers to a member of a partitioned data set, you must code the member name in parentheses after dsname, i.e., DSNAME=dsname(membername).

- If the DD statement is one of a group of DD statements required to define an indexed sequential data set, you must code one of the terms INDEX, PRIME, or OVFLOW in parentheses after dsname, i.e., DSNAME=dsname(PRIME). These terms identify the DD statements within the group.

Reference:

- For additional information on generation data groups, see the chapter "Additional DD Statement Facilities" in Section 2 of this publication. For instructions on creating and retrieving indexed sequential data sets, see Appendix D.

2. The name of a data set that is used several times in a job, whether specified in a DSNAME parameter or assigned by the system, can be copied after its first use with a simple reference to the DD statement that first identifies it. This allows you to change a name easily and eliminates your having to assign names to temporary data sets. To obtain the data set name from an earlier DD statement, code the keyword parameter

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

in the operand field of the DD statement. Replace the terms "stepname" and "ddname" with the job step name and DD statement name, respectively, where the data set was first defined.

Note:

- If the earlier DD statement is contained in a cataloged procedure step, you must include the procedure step name, i.e., DSNAME=*.stepname.procstepname.ddname.

3. A data set that exists only within the boundaries of a job can be assigned any temporary name. To assign a temporary name, code the keyword parameter

```
DSNAME=&name
```

Replace the term "name" with any 1- to 8-character name not used by another temporary data set in the job. The system replaces the &name with a name of the form name.jobname. (Thus, a DSNAME parameter specifying &WORK1 in a job named PAYROLL would result in a data set name of WORK1.PAYROLL.) You can retrieve this data set later in the job by coding DSNAME=&name in a DD statement, using the same name, or DSNAME=*.stepname.ddname.

Notes:

- If the DD statement refers to a member of a temporary partitioned data set, you must code the membername in parentheses, i.e., DSNAME=&name(membername).

- If the DD statement is one of a group of DD statements required to define an indexed sequential data set, you must code one of the terms INDEX, PRIME, or OVFLOW in parentheses after &name, i.e., DSNAME=&name(PRIME). These terms identify the DD statements within the group.

- DD statements defining temporary data sets should either use this form of the DSNAME parameter or have DSNAME omitted altogether. Temporary data sets should not be assigned a permanent name, i.e., DSNAME=dsname.

Requesting a Unit (UNIT)

The UNIT parameter of the DD statement allows you to specify information about the input or output unit(s) used by a data set. Unit information is not required if the data set:

- Is cataloged.

- Is passed from a previous step.

- Shares space or cylinders with an earlier data set, i.e., SUBALLOC or SPLIT is specified.

- Is assigned volumes used by an earlier data set in the same job, i.e., VOLUME=REF is specified.

In these cases, unit information is taken from other parameters and sources. You can

specify unit information in one of two ways:

1. By indicating a specific unit or group of units.

2. By requesting the same unit or units used by another data set in the same job step.

Other options of the UNIT parameter allow you to specify the number of units you need, defer mounting of volumes until the data set is opened, and request that the data set not be retrieved or stored with access mechanisms used by certain other data sets. To supplement this discussion of the UNIT parameter pictorially, turn to Appendix E and fold out Chart 3.

1. A specific unit or group of units can be identified by (a) its address, (b) its type number, or (c) its group name.

   (a) To identify a unit by its address, code the keyword parameter

   ```
   UNIT=address
   ```

   in the operand field of the DD statement. Replace the word "address" with the 3-byte address of the unit, as set by the operator on the LOAD-UNIT switches of the console, e.g., UNIT=180 for channel 1, control unit 8, unit 0.

   (b) Unit type numbers correspond to model numbers of input/output devices configured into your system. Type numbers are provided by the system automatically. Unit type numbers provide you with a certain degree of device independence in that your request may be filled by any of a number of devices of the same type. To identify a unit by its type number, code the keyword parameter

   ```
   UNIT=type
   ```

   in the operand field. Replace the word "type" with a valid unit type number, e.g., UNIT=2400-2. Unit type numbers for all units are listed in Appendix A.

   (c) Unit groups are established by your installation at system generation. They allow you to designate names for individual units or collections of units, and to classify collections of magnetic tape and direct-access units under the same name. To identify a group of units, code the keyword parameter

   ```
   UNIT=group
   ```

   in the operand field. Replace the word "group" with a valid unit group name, e.g., UNIT=TAPE.

Note:

- When you are using a priority scheduler, you should not use the unit address technique of identification unless absolutely necessary. This technique limits unit assignment and may result in delay of the job if the unit is being used by another job.

If a data set exceeds more than one volume, you can assign each volume to a separate unit. This eliminates your having to wait for demounting and mounting operations and allows you to process a data set that is split between two or more direct-access volumes. Code the number of units you desire after the unit type or group name, e.g., UNIT=(2400,3). If only one unit is required, you need not code the number. (If you code 0, the system assumes 1.)

If the data set is cataloged or the DD statement implies the number of volumes required, you can have the volumes mounted in parallel by coding "P" instead of a number. The system assigns one unit of the specified type for each volume on which the data set resides.

In certain instances, you may find it unneccessary or undesirable to have all data sets in place before their use in a job step. To defer the mounting of volumes until a data set is actually opened, code the word DEFER following the number of units, e.g., UNIT=(TAPE,2,DEFER). If the data set requires only one unit, insert an extra comma to indicate the absence of the number of units, e.g., UNIT=(TAPE,,DEFER).

Note:

- You cannot defer the mounting of a direct-access volume that is to

contain a new data set. DD statements required by operating system utility programs are an exception to this note, in that they require the specification of DEFER.

To optimize the flow of input and output data in a job step, you might want to assign separate access mechanisms to direct-access data sets. To request a separate access mechanism, code the keyword subparameter SEP=(ddname,...,ddname) in the last positions of the UNIT parameter. Replace the terms "ddname" with the names of other DD statements in the job step, e.g., UNIT=(,P,SEP=(INPUT1, INPUT2)). The system will attempt to process this data set with different access mechanisms than the ones used to process the data sets defined by the named DD statements.

Notes:

- When you use a sequential scheduler, the operating system ignores a SEP request if it conflicts with another request, or if sufficient access mechanisms are not available.
- When you use a priority scheduler, the operator is notified if the request cannot be satisfied. He must decide whether to continue the job or to withdraw separation requests.
- Unit separation requests are ignored when the automatic volume recognition feature is used.
- If the list of ddnames includes only one ddname, you need not code the inner parentheses, e.g., UNIT=(190,SEP=INPUT).

2. To conserve the number of units used in a job step, you can request that a data set be assigned the same unit(s) as assigned to an earlier data set in the same step. This technique, known as unit affinity, indicates that you want certain data sets and their associated volumes to use a single unit in sequential order. (Thus, deferred mounting is implied for data sets requesting unit affinity.) To request unit affinity, code the keyword parameter

```
UNIT=AFF=ddname
```

in the operand field. Replace the term "ddname" with the name of an earlier DD statement in the job step. This data set will be assigned the same unit or units as the data set defined by the named DD statement.

Note:

- Unit affinity requests are ignored when the automatic volume recognition feature is used.

Specifying Volume Information (VOLUME)

Information about the volume or volumes on which an input data set resides, or on which an output data set will reside is given in the VOLUME parameter. Volumes can be used most efficiently if you are familiar with the states a volume can assume. Volume states involve two criteria: the type of data set you are defining and the manner in which you request a volume.

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

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

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

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

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

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

The state of a volume determines when the volume will be demounted and what kinds of data sets can be assigned to it. The system determines the precise state of a volume by combining two characteristics, the "mountability" of the volume, and its availability for allocation. Some volumes are assigned a permanent state at system generation; others can assume different states, through MOUNT commands and VOLUME parameters in DD statements. The remainder of this discussion on volume states is divided into two parts, the first dealing with direct-access volumes and the second with magnetic tape volumes.

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

Mount Characteristics	Allocation Characteristics
Permanently Resident	Public
Reserved	Private
Removable	Storage

All combinations of characteristics are valid except removable/storage. Table 1 explains how direct-access volumes are assigned their mount and allocation characteristics. Actions 4 through 8 are induced by control statements in the input stream.

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

- All physically nondemountable volumes, such as 2301 Drum Storage.
- The volume from which the system is loaded (the IPL volume).
- The volume containing the system data sets SYS1.LINKLIB, SYS1.PROCLIB, and SYS1.SYSJOBQE.
- Volumes used by the system for SYSIN and SYSOUT. (Priority schedulers only.)

Any other direct-access volume can be designated as permanetly resident in a special member of SYS1.PROCLIB named PRESRES. The reserved characteristic applies to volumes that remain mounted until the operator issues an UNLOAD command. They can be reserved by either a MOUNT command referring to the unit on which they are mounted, or an entry in PRESRES. The removable characteristic applies to all volumes that are neither permanently resident nor reserved. Removable volumes do not have an allocation characteristic when they are not mounted. A reserved volume becomes removable after an UNLOAD command is issued for the unit on which it resides.

The allocation characteristics -- public, private, and storage -- deal with a volume's availability to be assigned by the system to temporary data sets, and, if the volume is removable, when it is to be demounted. A public volume is used primarily for temporary data sets and, if it is permanently resident, for frequently used data sets. It must be requested by a specific volume reference if a data set is to be kept or cataloged on it. If a public volume is removable, it is demounted only when it's unit is required by another volume. You can change a removable/public volume to private by specifying VOLUME=PRIVATE. A private volume must be requested by a specific volume reference, and, therefore, cannot contain temporary data sets having nonspecific volume requests. If it is reserved, it remains mounted until the operator issues an UNLOAD command for the unit on which it resides. If it is removable, it will be demounted after it is used unless you specifically requested that it be retained

Table 1. Direct-Access Volume States

Mount Characteristic	Allocation Characteristic		
	Public	Private	Storage
Permanently resident	1 PRESRES entry	2 PRESRES entry	3 PRESRES entry
Reserved	4 PRESRES entry -or- MOUNT command	5 PRESRES entry -or- MOUNT command	6 PRESRES entry -or- MOUNT command
Removable	7 VOLUME=PRIVATE not specified in the DD statement	8 VOLUME=PRIVATE specified in the DD statement	9 Invalid state

(VOLUME=RETAIN) or passed (DISP=PASS). Once a removable volume has been made private, it will ultimately be demounted. To use it as a public volume, you must have it remounted. A storage volume is always permanently resident or reserved; it is effectively an extension of main storage. Its principal use is to keep or catalog a data set having a non-specific volume reference.

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

Mount Characteristics	Allocation Characteristics
Reserved	Private
Removable	Scratch

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

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

- It is requested with a specific volume reference, except when DSNAME refers to a passed data set.
- It is requested for allocation to a nontemporary data set.
- The VOLUME parameter requests a private volume.

A removable/private volume is demounted after its last use in the job step, unless you request that it be retained.

All other tape volumes are assigned the removable/scratch state. They remain mounted until their unit is required by another volume.

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

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

These facilities are all optional. You can omit the VOLUME parameter when defining a new data set, in which case the system assigns a suitable public or scratch volume. To supplement this discussion of the VOLUME parameter pictorially, turn to Appendix E and fold out Chart 3.

To request that the data set be read from or written on a private volume, code

```
VOLUME=PRIVATE
```

in the operand field of the DD statement. Later data sets having nonspecific volume requests will not be assigned to this volume. In addition, the volume will be demounted after its last use in the jobstep.

Notes:

- If you specify VOLUME=PRIVATE, but do not request a specific volume, the system requests the mounting of a volume and assigns it the private characteristic.
- If you specify VOLUME=PRIVATE, and request a permanently resident volume, the data set is assigned to the volume, but the volume retains its public state.

If you have requested a private volume, you may want the volume to remain mounted after its last use in the job step. To bypass the demounting operation, code

```
VOLUME=(PRIVATE,RETAIN)
```

The volume remains mounted until after it is used in a subsequent step, or the end job, whichever occurs first.

Notes:

- If the data set resides on more than one volume, and the volumes are mounted in sequential order, only the last volume is retained.
- Tape volumes that are retained are rewound at the end of the step, and are not repositioned when they are used again.
- If the system finds it necessary to remove a retained volume, it ensures through messages to the operator that the volume is remounted before its next use.

When you are reading or writing a multivolume cataloged data set, a facility of the VOLUME parameter allows you to begin processing with a selected volume. Code

```
VOLUME=(,,seq#)
```

in the operand field. Replace the term "seq#" with a 1- to 4-digit volume sequence number. All volumes whose sequence numbers precede this number are omitted from processing.

Note:

- You can also include the parameters PRIVATE and RETAIN with the sequence number, e.g., VOLUME=(PRIVATE,,3) or VOLUME=(PRIVATE,RETAIN,3).

When you are creating a data set that exceeds one volume, you must either identify the volumes specifically or give a volume count, in which case suitable volumes are assigned. To make a nonspecific request, code

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

Replace the term "volct" with the number of volumes the data set requires.

Note:

- You can also include the parameters PRIVATE and RETAIN with the volume count, e.g., VOLUME=(PRIVATE,,,4) or VOLUME=(PRIVATE,RETAIN,,4).

To make a specific volume request, you can either identify the volumes by their serial numbers or use the same volume(s) used by an earlier data set in the job. Code

```
VOLUME=SER=(ser#,...,ser#)
```

to identify the volumes by serial numbers. Replace the terms "ser#" with the 1- to 6-character volume serial numbers associated with the volumes. This form of the VOLUME parameter must be used when retrieving noncataloged data sets.

Notes:

- If only one volume is involved, you need not code the parentheses, i.e., VOLUME=SER=ser#.
- When you use a sequential scheduler, the volume containing the input stream has the serial SYSIN. This serial must not be used for other volumes. With a priority scheduler, other serials can be used for volumes containing input streams.
- Each volume in your installation should have a different serial number.
- Labeled tape reels contain the volume serial number in their labels. However, all tape reels should be made easily identifiable, such as having the serial number posted on the reel.
- You can also include the parameters PRIVATE and RETAIN when you use SER, i.e., VOLUME=(PRIVATE,SER=(ser#s)) or VOLUME=(PRIVATE,RETAIN,SER=(ser#s)).

The alternative means of making a specific volume request is by using the same volumes as assigned to an earlier data set. If the data set is cataloged or passed, you refer to it by coding

```
VOLUME=REF=dsname
```

Replace the term "dsname" with the data set's cataloged name. If it is not cataloged or passed, you can refer to the DD statement that defined it by coding

```
VOLUME=REF=*.stepname.ddname
```

Replace the terms "stepname" and "ddname" with the name of the job step and DD statement where the earlier data set is defined.

Notes:

- If the earlier data set is part of the same job step, you need not code the stepname i.e., VOLUME=REF=*.ddname.
- If the earlier data set is part of a cataloged procedure, you must include the procedure step name, i.e., VOLUME=REF=*.stepname.procstepname.ddname.
- If the earlier data set resides on more than one tape volume, only the last volume is assigned when VOLUME=REF is specified.

Describing the Attributes of the Data Set (DCB)

Specifying data set attributes at program execution time represents one of the DD statement's primary advantages. Through the use of the DCB parameter, you can specify data set label information, buffer requirements, and other pertinent information at the time a processing program is executed rather than when it is compiled.

A data control block associated with each data set is originally constructed in the processing program by a DCB macro-instruction. Here you can specify descriptive information that is not subject to change. When the processing program is executed, the remaining DCB information is supplied by the data set label (provided the data set already exists) and the DD statement. The DCB parameter must be coded in the DD statement unless the data control block is completed by the processing program, default options assumed in the OPEN macro-instruction, or the data set label.

To save time in coding the DCB parameter, you may be able to copy the data set label information associated with a similar data set. To copy the data set label information associated with a cataloged direct-access data set, code the keyword parameter

```
DCB=dsname
```

in the operand field of the DD statement. Replace the word "dsname" with the data set's cataloged name. 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.

If such a data set does not exist, you still might be able to copy the DCB parameter of an earlier DD statement in the job. To refer to this DD statement, code the keyword parameter

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

in the operand field. Replace the terms "stepname" and "ddname" with the job step name and DD statement name, respectively.

Notes:

- If the earlier DD statement is contained in the same job step, you need not code the stepname, i.e., DCB=*.ddname.
- If the earlier DD statement is contained in a cataloged procedure step, you must include the procedure step name, i.e., DCB=*.stepname.procstepname.ddname.

If you wish to modify the information that is copied from another data set label or DCB parameter, code

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

Replace the term "reference" with dsname or *.stepname.ddname. The attributes in the list override the corresponding copied attributes. Data set attributes are coded in the form of keyword subparameters separated by commas, e.g., BLKSIZE=810 for a block size of 810 bytes. These subparameters correspond to operands in the DCB macro-instruction and are coded using the same keywords and values. A glossary of valid DCB subparameters is given in Appendix B of this publication.

If you cannot copy another data set label or DCB parameter, you must supply all DCB attributes that are not specified in the processing program (either directly or by default) or data set label. Code the keyword parameter

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

in the operand. Again, the attributes are coded as keyword subparameters separated by commas, e.g., DCB=(RECFM=FB,LRECL=80,...).

References:

- DCB macro-instructions and operands are described in detail in the publication IBM System/360 Operating System: Supervisor and Data Management Macro-Instructions.
- DCB macro-instructions and operands associated with the graphic access methods are described in the publications IBM System/360 Operating System: Graphic Programming Services for the IBM 2250 Display Unit, Form C27-6909, and Graphic Programming Services for IBM 2260 Display Station (Local Attachment), Form C27-6912.

Describing the Data Set Label (LABEL)

Data sets residing on magnetic tape volumes usually have data set labels. Direct-access volumes and volumes mounted through the automatic volume recognition feature must have labels conforming to standard label specifications. The LABEL parameter indicates the label type, the data set's relative position on tape, its retention period, and whether a password is required to read or write on it. To supplement this discussion of the LABEL parameter pictorially, turn to Appendix E and fold out Chart 3.

Magnetic tape volumes can contain volume labels and data set header and trailer labels that do not conform to the system standard label specifications. To create or retrieve a data set residing on such a tape volume, you must include the LABEL parameter. To specify the label type, code

```
LABEL=(,type)
```

in the operand field. Replace the word "type" with:

SL - if the data set has standard labels.
NL - if the data set has no labels.
NSI - if the data set has nonstandard labels.
SUL - if the data set has both standard and user labels.
BLP - to bypass label processing.

If you specify SUL, SL, or omit the label type (in which case standard labels are assumed), the operating system will

ensure that the correct volumes are mounted. If you specify NSL, your installation must have incorporated label processing routines into the operating system. If you specify NL, the data set must have no labels.

The feature that allows you to bypass label processing is a system generation option (OPTIONS=BYLABEL). If this option was not requested at system generation and you have coded BLP, the system assumes NL.

Note:

- When BLP is specified, you should ensure that the operator mounts the correct tape volume before processing it.

If the data set is not first in sequence on the reel, the LABEL parameter serves to position the tape properly through a data set sequence number. Code

```
LABEL=seq#
```

in the operand field. Replace the term "seq#" with the 1- to 4-digit sequence number assigned to the data set when it was created.

The sequence number describes the data set's position with respect to other data sets on the volume or group of volumes.

Notes:

- If you are retrieving a data set that resides on an unlabeled tape volume, you must give its sequence number with respect to other data sets on that single volume, beginning with 1.
- If 0 appears as the data set sequence number, the system assumes 1.

Both magnetic tape and direct-access data sets can be assigned a retention period and password protection when they are created. If you wish the data set to remain intact for some period of time, you can specify either the length of time in days or the exact date you want it to expire. Otherwise, a retention period of zero days is assumed. After expiration, the data set can be deleted, or opened for any type of output. To specify a retention period, code

```
LABEL=RETPD=nnnn
```

in the operand field. Replace the term "nnnn" with the number of days you want the data retained. To specify, instead, an expiration date, code

```
LABEL=EXPDT=yyddd
```

in the operand field. Replace the term "yyddd" with the 2-digit year number and 3-digit day number after which the data set can be considered expired.

If you wish the data set to be accessible only through the use of a password, code

```
LABEL=(,,PASSWORD)
```

in the operand field. The operating system assigns the data set security protection. To retrieve it, the operator must respond to a message by issuing the correct password.

Note:

- Subparameters in the LABEL parameter can be coded in various combinations. The terms seq#, type, and PASSWORD are all positional subparameters.

Specifying Data Set Status and Disposition (DISP)

The DISP parameter describes the status of a data set and indicates what is to be done with it after it is processed. You can omit this parameter if a data set is created and deleted during a single job step.

The first term in the DISP parameter reflects the data set's status with respect to the job step. If the data set existed before the job step, it can be used either as an input data set to be read or as a partially completed output data set. To specify the status of an existing data set used as input to a processing program, code

```
DISP=OLD
```

in the operand field of the DD statement. If the data set resides on a direct-access volume and is part of a job whose operations do not prevent simultaneous use of the data set by another job, code

```
DISP=SHR
```

in the operand field. This parameter has meaning only in a multiprogramming environment for existing data sets. If SHR is not coded in a multiprogramming environment, the data set is considered unusable by concurrently operating jobs. If SHR is coded in other than a multiprogramming environment, the system assumes the data set's status is OLD.

If a data set is sequentially organized, and is used for additional output by the processing program, code

```
DISP=MOD
```

in the operand field. When the data set is opened, the read/write mechanism is automatically positioned after the last record in the data set. If no volume information is available for the data set, the system assumes it does not yet exist and changes the MOD specification to NEW. (Volume information is considered available if it is coded in the DD statement, passed with the data set from a previous step, or contained in the catalog.)

A data set created in a job step is used by the processing program for output data. You can indicate a new status by coding

```
DISP=NEW
```

in the operand field, by coding DISP=MOD and including parameters usually required by new data sets, or by omitting a status specification altogether.

The second term in the DISP parameter tells how you want the data set handled by the job scheduler at the end of the job step that processes the data set. If the job scheduler can determine that the data set was not opened, the requested disposition is not performed.

Notes:

- If the job step is bypassed because of an error that occurs before the step is executed (e.g., an incorrect control statement is read, the system is not able to allocate units, etc.), requested dispositions are not performed. Subsequent job steps are also bypassed.
- If the job step is bypassed because of a return code test, requested dispositions are performed only for data sets that have been passed from a previous step.
- If the job step is bypassed because of an error that occurs during execution of the step (e.g., an incorrect volume label is encountered), requested dispositions are performed.

If you want the data set to assume the same status it had before the job step, you need not code a disposition. Existing data sets (OLD, MOD, and SHR) will continue to exist and newly created data sets (NEW) will be deleted. Special dispositions allow you to:

1. Uncatalog a data set.
2. Catalog a data set.
3. Delete an existing data set.
4. Keep a data set.
5. Pass a data set to a later job step.

These dispositions are discussed in the following numbered paragraphs.

1. To uncatalog an input data set, code the keyword parameter

   ```
   DISP=(OLD,UNCATLG)
   ```

   in the operand field. The catalog entry that points to the data set is removed from the index structure. If the data set resides on a direct-access volume, it remains tabulated in the volume table of contents.

2. To catalog a data set, code

```
DISP=(status,CATLG)
```

in the operand field. The term "status" reflects the data set's status, as discussed in earlier paragraphs. When you request cataloging, an index entry pointing to the data set is placed in the system catalog. The index structure required to catalog the data set must be defined before the cataloging operation can take place. DD statements in subsequent jobs can then refer to this data set simply by giving its cataloged name.

Notes:

- A cataloged data set whose status is MOD might be expanded to additional volumes during the job step. To update the catalog to reflect these additional volumes, code DISP=(MOD,CATLG).
- If the status of the data set is NEW, and you want to catalog it, you can omit the term NEW. However, you must indicate its absence with a comma, e.g., DISP=(,CATLG).

3. If you have no further need for a data set after its use and want to release its space, code

```
DISP=(status,DELETE)
```

in the operand field. The data set is automatically uncataloged if you have used the catalog to locate it. In addition, the system removes the volume table of contents entry associated with the data set, if it resides on a direct-access device.

Notes:

- If the status of the data set is NEW and you want to delete it after its use, i.e., the data set is temporary, you can omit the DISP parameter altogether.
- If you specify DISP=(SHR,DELETE), the system assumes OLD instead of SHR.

4. For data sets that are used in a later job but are not of sufficient importance to warrant their being cataloged, code

```
DISP=(status,KEEP)
```

in the operand field. The data set is kept intact until a DELETE request is encountered. If the volume containing the data set is demounted, the system advises the operator of the data set's KEEP disposition. If the data set resides on a direct-access volume, it remains tabulated in the volume table of contents.

Note:

- If the status of a data set is NEW and you want to keep it until a later time, you can omit the term NEW. However, you must indicate its absence with a comma, e.g., DISP=(,KEEP).

5. When a data set is used by two or more job steps in the same job, you can eliminate retrieval and disposal operations by passing it from step to step. Each step can use the data set one time. You do not indicate the final disposition of the data set until its last use in the job. To pass a data set to a succeeding step, code

```
DISP=(status,PASS)
```

in the operand field. Subsequent DD statements referring to the passed data set must identify it with the DSNAME parameter, must provide either no unit information, or unit information consistent with that in the original data set, and must issue another disposition. Between steps, the volume that contains the passed data set remains mounted; thus, you need not code RETAIN in the VOLUME parameter of a DD statement that specifies a disposition of PASS.

Notes:

- If the status of a data set is NEW and you want to pass it, you can omit the term NEW. However, you must indicate its absence with a comma, e.g., DISP=(,PASS).
- If the system finds it necessary to remove the volume containing a passed data set, it ensures through messages to the operator that the volume is remounted before its next use.

Allocating Direct-Access Space (SPACE, SPLIT and SUBALLOC)

When writing a new data set on a direct-access volume, you allocate space on the volume with a DD statement parameter. You can allocate direct-access space:

1. By requesting the quantity of space and letting the system assign specific tracks.
2. By requesting specific tracks.
3. By splitting cylinders with other data sets.
4. By suballocating space from an earlier data set.

To supplement this discussion of space allocation pictorially, turn to Appendix E and fold out Chart 3.

1. The most frequently used technique of space allocation requires that you specify the amount of space you desire and let the system assign specific tracks. This technique is recommended for allocating direct-access space when the job is operating in a multiprogramming environment. Other options permit you to request the manner in which the space is to be arranged, to release unused space, and to request that the space begin and end on cylinder boundaries.

   The quantity of space you desire can be given in units of tracks, cylinders, or blocks, whichever is most convenient. In the latter case the system will compute the number of tracks or cylinders required. To allocate space using this technique, code the keyword parameter

```
SPACE=(units,quantity)
```

   in the operand field. Replace the term "units" with:

   TRK - If you want space in tracks.
   CYL - If you want space in cylinders.
   average block length in bytes - If you want space in terms of blocks.

   Replace the term "quantity" with the amount of space you desire in the units you have chosen, e.g., SPACE=(TRK,200) for 200 tracks, SPACE=(CYL,10) for 10 cylinders, and SPACE=(400,100) for 100 blocks with an average length of 400 bytes.

   Notes:

   - For most efficient performance, request space in units of cylinders (CYL).
   - The average record length cannot exceed 65,535 bytes.
   - If you request space in units of blocks, and the blocks have keys, you must give the key length in the DCB parameter, i.e., KEYLEN=n.

   If the possibility exists that the data set might at some time exceed the amount of space you requested, you can ensure that extra space will be made available by denoting an incremental quantity. Code

```
SPACE=(units,(quantity,increment))
```

   in the operand field. Replace the term "increment" with a decimal number. Each time the data set exhausts its space, additional space will be allocated on the same volume in the amount of the increment. For example, if you coded SPACE=(TRK,(200,10)), the data set would be initially allocated 200 tracks. If it later exceeded 200 tracks, 10 additional tracks would be made available. This incrementing by 10 tracks would take place each time (up to a maximum of 15 times) the data set exhausted its total space.

   If the data set for which you are allocating space has a partitioned organization (i.e., a BPAM data set), you must indicate the size of its directory in the SPACE parameter. Code

```
SPACE=(units,(quantity,,directory))
```

   in the operand field. Replace the term "directory" with the number of 256-byte blocks in the directory.

   Note:

   - If you wish to give an incremental quantity, code SPACE=(units,(quantity,increment,directory)).

   If the data set has an indexed sequential organization (i.e., a BISAM or QISAM data set), you must indicate the size of its index in the SPACE parameter. Code

```
SPACE=(units,(quantity,,index))
```

in the operand field. Replace the term "index" with the size of the index, in cylinders.

Notes:

- If you wish to give an incremental quantity, code SPACE=(units, (quantity,increment,index)).
- The operating system differentiates between the terms "directory" and "index" by examining the DCB parameter. The DCB parameter for an indexed sequential data set must contain one of the attributes DSORG=IS or DSORG=ISU.

For simplicity, the terms "(quantity,increment,directory)" and "(quantity,increment,index)" will be represented by "(quantities)" in the remainder of this discussion.

An optional facility of the SPACE parameter allows you to release unused space when you are finished writing the data set. To use this facility, code

```
SPACE=(units,(quantities),RLSE)
```

in the operand field. All unused space is released to the system when the data set is closed.

Note:

- RLSE is a positional subparameter. Thus, its absence must be indicated by a comma if subsequent subparameters are coded.

When you request space using this technique, the operating system attempts to allocate space in contiguous tracks or cylinders. If contiguous space is not available, the system satisfies the request with up to five noncontiguous blocks of storage. You can override these system actions by coding one of three subparameters:

(a) CONTIG (contiguous)
(b) MXIG (maximum contiguous)
(c) ALX (all extents)

The subparameter you choose must follow either RLSE or the comma showing the absence of RLSE.

a. To ensure that space is allocated in contiguous tracks or cylinders, code

```
SPACE=(units,(quantities),
                    RLSE,CONTIG)
```

in the operand field. The system allocates space in one area of space, with no intervening tracks or cylinders.

b. If the data set is likely to exceed the amount of space you requested, you can request a larger amount of space by coding

```
SPACE=(units,(quantities),
                      RLSE,MXIG)
```

in the operand field. The system allocates the largest area of contiguous auxiliary storage that is at least as large as the quantity you requested.

c. If you desire an unusually large amount of space, code

```
SPACE=(units,(quantities),
                      RLSE,ALX)
```

in the operand field. The system allocates five areas of contiguous storage, each at least as large as the quantity you requested. If this request cannot be fully satisfied, the system allocates as many blocks as are available.

If you allocate space in units of blocks, you can further request that the space be rounded to an integral number of cylinders to increase performance. Code

```
SPACE=(units,(quantities),,,ROUND)
```

in the operand field. The system computes the number of tracks or cylinders required 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.

Note:

- The commas may be replaced by RLSE, and CONTIG, MXIG, or ALX, as you desire.

2. A data set can be placed in a specific position on a direct-access volume by requesting space in terms of a quantity and a track number. This allocation technique is recommended only for location-dependent data sets. To allocate tracks beginning at a specific address, code the keyword parameter

```
SPACE=(ABSTR,(quantity,address))
```

in the operand field of the DD statement. Replace the term "quantity" with the number of tracks you desire, and "address" with the relative track address of the beginning track. (The relative track address of the first track on the volume is 0. This track cannot be allocated.) If tracks you request have been allocated to another data set, the job is terminated.

Note:

- If the new data set is partitioned, you must indicate its directory size in the SPACE parameter by coding SPACE=(ABSTR,(quantity,address,directory)). Replace the term "directory" with the number of 256-byte blocks in the directory.

3. When a job step involves one or more 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 a percentage of the tracks on every cylinder allocated.

   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.

   To request the total amount of space in units of cylinders, code the keyword parameter

```
SPLIT=(n,CYL,quantity)
```

   in the operand field of the first DD statement in the sequence. Replace the letter "n" with the number of tracks per cylinder you wish allocated to the first data set. Replace the term "quantity" with the total number of cylinders to be allocated for all the data sets. Each succeeding DD statement in the group must contain the parameter SPLIT=n, where n is the number of tracks per cylinder to be allotted to the associated data set.

   To request the total amount of space in units of blocks, code the keyword parameter

```
SPLIT=(%,blksize,quantity)
```

   in the operand field of the first DD statement in the sequence. Replace the character "%" with the percentage of tracks per cylinder you wish allocated to the first data set. Replace the terms "blksize" and "quantity" with the average length of the blocks in the data sets and the total number of blocks, respectively. The system computes the total number of cylinders from these figures. Each succeeding DD statement in the group must contain the parameter SPLIT=% in the operand field, where % is the percentage of tracks per cylinder to be allotted to the associated data set.

   Notes:

   - The SPLIT parameter cannot be used to allocate space for direct (BDAM) data sets or data sets residing on drum storage volumes.
   - The average block length cannot exceed 65,535 bytes.
   - If space is requested in units of blocks, and the blocks have keys, you must give the key length in the DCB parameter, i.e., KEYLEN=n.

   As in the SPACE parameter, you can ensure that increments of extra space will be automatically allocated by coding

```
SPLIT=(n,CYL,(quantity,increment))
```

   for increments in cylinders, or

```
SPLIT=(%,blksize,
              (quantity,increment))
```

   for increments in blocks. If any of the data sets in the group exceeds its allotted space, additional space is

allocated in the amount of the increment. (If space is requested in blocks, the system increases the increment to an integral number of cylinders.) The additional space applies only to the data set that exhausted its allotted space, and is not split with the other data sets in the group.

4. The fourth method of obtaining direct-access space is through the technique of suballocation. Suballocation allows you to place a number of data sets in contiguous order on a direct-access device. To use this technique, you simply request part of the space assigned to an earlier data set. Code

```
SUBALLOC=(units,(quantities),
                    stepname.ddname)
```

in the operand field of the DD statement. Replace the term "units" as in the SPACE parameter:

TRK - for a space request in tracks.
CYL - for a space request in cylinders.
average block length in bytes - for a space request in blocks.

The term "(quantities)" represents "(quantity,increment,directory)." The incremental quantity is optional and the number of directory blocks is required only when the DD statement defines a partitioned data set. If you indicate an incremental quantity, increments of space are allocated from available space on the volume, not from the space in the original data set. Replace the terms "stepname" and "ddname" with the names of the job step and the DD statement where an earlier data set is defined; the system suballocates the amount of space you request from this earlier data set.

Notes:

- Space obtained through suballocation must be contiguous, and cannot be further suballocated.
- The original data set must be used only for suballocation. Suballocated space is removed from the front of it.
- If the suballocation request refers to a DD statement in the same job step, you need not code the job step name, i.e., code the stepname, i.e., SUBALLOC=(units,(quantities),ddname).
- If the suballocation request refers to a DD statement in a cataloged procedure, you must include the name of the procedure step in which it appears, i.e., SUBALLOC=(units,(quantities),stepname.procstepname.ddname).

Optimizing Channel Usage (SEP and AFF)

A job step that requires several input and output operations might be performed more efficiently by balancing the channel requirements of its data sets. To obtain optimum channel usage, you can request that a data set be assigned a separate channel from the ones assigned to earlier data sets. A later DD statement can express the same separation requirements by requesting affinity.

To request channel separation from as many as eight other data sets in the job step, code the keyword parameter

```
SEP=(ddname,...,ddname)
```

in the operand field of the DD statement. Replace the terms "ddname" with the names of up to eight earlier DD statements in the job step.

To extend the channel separation facility, you can request affinity with an earlier data set that requested channel separation by coding the keyword parameter

```
AFF=ddname
```

in the operand field of a later DD statement. Replace the term "ddname" with the name of the earlier DD statement. The data set that requests affinity is also separated channelwise from those identified in the SEP parameter of the earlier statement. This feature eliminates your having to write identical SEP parameters more than once.

Note:

- Channel separation and affinity requests are ignored if the automatic volume recognition feature is used.

THE COMMAND STATEMENT

Command statements are inserted in the input stream by the operator. They must appear immediately before a JOB statement, an EXEC statement, a null statement, or another command statement, and cannot be interspersed with data in the input stream. With a sequential scheduler, all commands except SET, START, and UNLOAD are accepted as they are issued. If you use a priority scheduler, all commands are accepted when issued.

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

Note:

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

Summary of Available Commands

Different sets of commands are available at the three system levels. With systems having a primary control program, the following commands can be used in the input stream:

DISPLAY	STOP
MOUNT	UNLOAD
SET	VARY
START	

With systems that provide multiprogramming with a fixed number of tasks, you can use the SHIFT command in addition to the above commands.

With systems that provide multiprogramming with a variable number of tasks (a priority scheduler), the set of valid commands comprises:

DISPLAY	RESET
HALT	SET
HOLD	START
LOG	STOP
MODIFY	UNLOAD
MOUNT	VARY
RELEASE	WRITELOG

A complete discussion of commands and operands is presented in the publication IBM System/360 Operating System: Operator's Guide.

THE DELIMITER STATEMENT

The delimiter statement marks the end of a data set in the input stream. Code the identifying characters /* in columns 1 and 2, with other fields blank. You can code comments at your discretion.

Note:

- When using a priority scheduler, you need not mark the end of a data set in the input stream that is defined by a DD * statement.

THE NULL STATEMENT

The null statement is used to mark the end of certain jobs in an input stream. If the last DD statement in a job defines data in an input stream, the null statement should be used to mark the end of the job so that the card reader is effectively closed. Code the identifying characters // in columns 1 and 2, and leave all remaining columns blank.

The Job Control Language is a comprehensive medium for controlling the performance of many different types of jobs. Because of its power and flexibility, it appears somewhat complex in its entirety. However, most individual applications use only subsets of the language. Succeeding chapters in Section 2 define and illustrate control statement and parameter requirements for selected applications of the language.

Job control statements are initially processed by a job scheduler component, the interpreter. Thus, the statements must conform to certain coding conventions. These conventions comprise (1) the use of special characters, (2) the spacing of fields, and (3) rules for continuing statements onto additional cards or card images.

Coding Special Characters

Special characters are used in the Job Control Language to delimit parameters and fields, and to perform other syntactical functions. With the exception of the cases listed below, variable information in a parameter must be coded in alphameric and national characters. Table 2 defines the alphameric, national, and special character sets.

Table 2. Character Sets

Character Set	Contents	
Alphameric	Alphabetic	A through Z
	Numeric	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	

Four 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 in the JOB statement.
2. The programmer's name in the JOB statement.
3. The ACCT parameter in the EXEC statement.
4. The PARM parameter in the EXEC statement.

Because these parameters are passed directly to installation routines or processing programs, you can include as part of the information any of the special characters. However, you must notify the interpreter of this by enclosing the item that contains the special characters in apostrophes (5-8 punch), e.g., PARM='123,456'. If one of the special characters is an apostrophe, you must identify it by coding two consecutive apostrophes (5-8 punches) in its place, e.g., 'O''NEILL'.

Notes:

- The programmer's name in the job statement can contain periods without being enclosed in apostrophes, e.g., T.JONES.
- The unit type number in the UNIT parameter of the DD statement can contain a hyphen without being enclosed in apostrophes, e.g., UNIT=2400-2.
- The account number and items of accounting information in the JOB statement and the ACCT parameter of the EXEC statement can contain hyphens without being enclosed in apostrophes.

Spacing Control Statement Fields

Except for the identifying characters in columns 1 and 2, 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 can not code blanks between parameters.

Continuing Control Statements

Control statements are contained in columns 1 through 71 of cards or card images. If the total length of a statement exceeds 71 columns, or if 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, including the comma that follows it, at or before column 71. (See note below.)
2. Follow the interrupted field with at least one blank. You may leave blanks or write comments in the remaining space, through column 71. (If the comma is coded in column 71, do not leave a blank after it.)
3. Code any nonblank character in column 72.
4. Code the identifying characters // in columns 1 and 2 of the following card or card image.
5. Continue the interrupted operand beginning in column 16. (Columns 3 through 15 must be blank.)

Note:

- The accounting information in the JOB statement, the ACCT parameter in the EXEC statement, and the DCB parameter in the DD statement can also be interrupted after a complete subparameter.

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.

2. Code a nonblank character in column 72 if the interruption occurs before this column.

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

4. Continue the comments fields beginning in any column after 15 (columns 3 through 15 must be blank).

DEFINING A JOB

Information related to the performance of a job is presented to the job scheduler in the JOB statement. Most of the information is optional. JOB statements also serve to define the boundaries of a job.

Coding Required Information

To make a job acceptable to the scheduler, you must include identifying characters, a job name, and the operation "JOB" in the JOB statement:

```
//PAYROLL JOB
```

If your installation has established special accounting routines to handle the charging of jobs, you may also be required to code an account number and your name:

```
//PAYROLL JOB 5048321,A.USER
```

These items of information are passed directly to your installation's accounting routines.

If the accounting routines require additional information, you can add additional items of information to the account number:

```
//PAYROLL JOB (5048321,013,14-01),A.USER
```

The total number of characters in the account number and accounting information, plus intervening commas, cannot exceed 142.

Coding Optional Information

The remaining information in the JOB statement is optional and may be coded in any order and combination you choose. Some parameters apply only to priority schedulers and are ignored by sequential schedulers.

If you wish to see a printout of all the job control statements in the job, in addition to the normal printout, of incorrect statements and associated diagnostic messages, code the MSGLEVEL parameter:

```
//PAYROLL JOB 5048321,A.USER,MSGLEVEL=1,...
```

To test the return codes issued by each step in the job, code the COND parameter:

```
//PAYROLL JOB 5048321,A.USER,COND=((12,LE),(8,EQ)),...
```

The system tests each step in the job using this criteria. If any test is satisfied, the job is terminated. In this case, if 12 is less than

or equal to a return code, or if any return code is 8, the job is terminated. In addition to the operators LE and EQ, you can use LT (less than), NE (not equal to), GT (greater than), and GE (greater than or equal to).

With a priority scheduler, you assign job priority with the PRTY parameter:

```
//PAYROLL JOB 5048321,A.USER,PRTY=14,...
```

The scheduler arranges and selects jobs for execution according to their priority numbers. In this case, the job PAYROLL is assigned top priority. Priority numbers range from 0 to 14.

Messages issued by the job scheduler are directed to the standard output class, A. If you are using a priority scheduler, you can direct messages to a different class by coding the MSGCLASS parameter. Message classes can be represented by any alphameric character:

```
//PAYROLL JOB 5048321,A.USER,MSGCLASS=F,...
```

If you wish to indicate the main storage requirements of a job operating under a priority scheduler, code the REGION parameter:

```
//PAYROLL JOB 5048321,A.USER,REGION=51K,...
```

REGION parameters can also be coded in EXEC statements, but are superseded by a REGION parameter coded in the JOB statement.

Defining Job Boundaries

Jobs are ordinarily bounded by JOB statements. Each JOB statement marks the beginning of one job and the end of the control statements for the preceding job. Figure 3 shows a group of jobs and their boundaries.

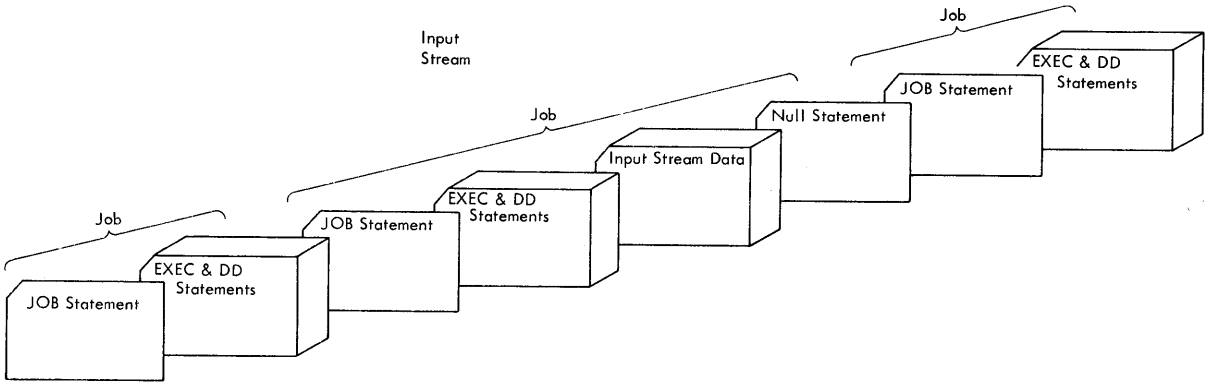

Figure 3. Defining Job Boundaries

DEFINING A JOB STEP

A job step is the unit of work associated with the execution of one program or one cataloged procedure. The program or cataloged procedure is identified in the EXEC statement. The EXEC statement also provides information related to the execution of a job step and serves to define job step boundaries.

Identifying a Program

To execute a program, you must include identifying characters (//), the operation "EXEC", and the PGM parameter in the EXEC statement. If later control statements refer to the job step in any way, you must also code a job step name. A program that resides in a temporary library can be identified by referring to the DD statement that defines the temporary library:

```
//STEP2 EXEC PGM=*.STEP1.TEMPLIB
```

Here, the system retrieves the program from the temporary library defined in an earlier DD statement named TEMPLIB, which occurs in a job step named STEP1.

If the program resides in the system library or a private library, you identify it by its member name:

```
//STEP3 EXEC PGM=CONVERT
```

The system searches the appropriate library for the member named CONVERT.

Identifying a Cataloged Procedure

Cataloged procedures reside as members in the system procedure library, SYS1.PROCLIB. A cataloged procedure may comprise any number of EXEC statements and DD statements. You identify a cataloged procedure by its 1- to 8-character name:

```
//STEP4 EXEC ANALYSIS
```

Here, the system searches the procedure library for the member named ANALYSIS. If you wish to note the fact that ANALYSIS is a procedure, code PROC= before the procedure name:

```
//STEP4 EXEC PROC=ANALYSIS
```

Coding Optional Information

Remaining parameters in the EXEC statement provide optional information related to the execution of the job step. They can be coded in any order and combination you choose.

If the valid execution of the job step depends on the results of earlier steps, you can test the return codes issued by these steps:

```
//STEP3 EXEC PGM=CONVERT,COND=((4,EQ,STEP1),(8,EQ,STEP2)),...
```

Here, the system tests the return codes issued by earlier steps named STEP1 and STEP2. If STEP1 issues a return code of 4, or STEP2 issues an 8, STEP3 is bypassed. You can compose up to 8 different return code tests.

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

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

Here, the cataloged procedure step named REDUCE is bypassed if the procedure step named LOOKUP issues a return code of 16. You can code as many COND parameters of this type as there are steps in the procedure.

Special control information can be passed to a processing program using the PARM parameter:

```
//STEP3 EXEC PGM=CONVERT,PARM='3-14-67,3.1416',...
```

The system picks up the information 3-14-67,3.1416 and passes it to the processing program. Language processors require specific PARM information, as outlined in the programmer's guide associated with each processor.

If the EXEC statement calls a cataloged procedure, you can provide control information to a procedure step by coding the PARM parameter followed by the name of the procedure step to which it applies:

```
//STEP4 EXEC ANALYSIS,PARM.REDUCE=10,...
```

Since the information 10 contains no special characters, enclosing apostrophes are omitted. You can code as many PARM parameters of this type as there are steps in the procedure. If you wish to pass information to the first procedure step and nullify all other PARM parameters in the procedure, code the PARM parameter without a procedure step name.

Accounting information pertinent to a job step or procedure step is coded in the ACCT parameter.

```
//STEP3 EXEC PGM=CONVERT,ACCT=5052321,...
```

The system passes the accounting information 5052321 to your installation's job step accounting routines.

If the EXEC statement calls a cataloged procedure, you can provide accounting information for a procedure step by coding the ACCT parameter followed by the name of the procedure step to which it applies:

```
//STEP4 EXEC ANALYSIS,ACCT.DISPLAY=(013,15-01),...
```

You can code as many ACCT parameters of this type as there are steps in the procedure.

When using a priority scheduler, you can establish a time limit for the execution of a job step or procedure step with the TIME parameter:

```
//STEP3 EXEC PGM=CONVERT,TIME=(1,30),...
```

Here, the job step is automatically terminated if its execution time exceeds one minute 30 seconds.

If the EXEC statement calls a cataloged procedure, you can time one step of the procedure by coding the TIME parameter followed by the name of the procedure step:

```
//STEP4 EXEC ANALYSIS,TIME.LOOKUP=(,45),...
```

The procedure step named LOOKUP is terminated if its execution time exceeds 45 seconds. The comma preceding 45 indicates the omission of minutes. You can code as many TIME parameters of this type as there are steps in the procedure.

If you wish to indicate the main storage requirements of a job step operating under a priority scheduler, code the REGION parameter:

```
//STEP3 EXEC PGM=CONVERT,REGION=48K,...
```

This parameter should be coded if the job step requires a region size different from the default region size, and REGION is not coded in the JOB statement.

If the EXEC statement calls a cataloged procedure, you can indicate the region size for a procedure step by coding the REGION parameter followed by the name of the procedure step to which it applies:

```
//STEP4 EXEC ANALYSIS,REGION.LOOKUP=48K,...
```

Here the procedure step LOOKUP is assigned a region size of (48 X 1024) bytes. You can code as many REGION parameters of this type as there are steps in the procedure.

Defining Job Step Boundaries

Job step boundaries are ordinarily established by EXEC statements. Each EXEC statement marks the beginning of a new job step and the completion of control statements for the previous job step. The completion of the last step in a job is marked with a JOB statement associated with the succeeding job, or a null statement. Figure 4 shows a group of job steps and their boundaries.

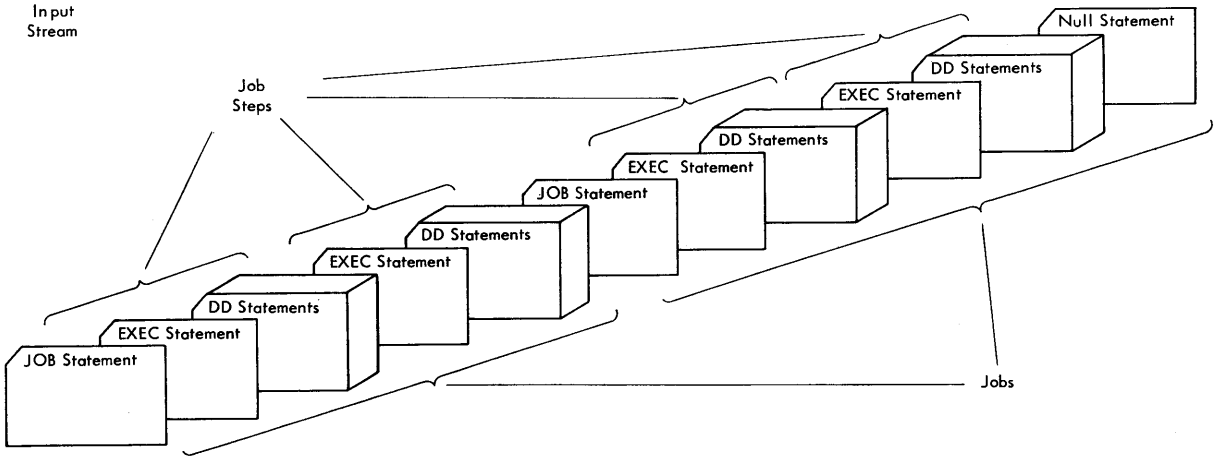

Figure 4. Defining Job Step Boundaries

CREATING NEW OUTPUT DATA SETS

Output data sets in a job step that do not exist before the step is executed are created by using subsets of the DD statement parameters. The contents of each subset depend primarily on what type of device the data set resides on.

Creating Unit Record Data Sets

Data sets whose destination is a printer or card punch are created with the DD statement parameters UNIT and DCB.

UNIT: Required. Code unit information using the 3-digit address (e.g., UNIT=00E), the type (e.g., UNIT=1403), or the system-generated group name (e.g., UNIT=PRINTER).

DCB: Required only if the data control block is not completed in the processing program. Valid DCB subparameters are listed in Appendix B.

Examples 1 and 2 illustrate valid DD statements for creating data sets on the printer or card punch.

```
//OUTPUT DD UNIT=1403
```

Example 1. Creating a Data Set on the Printer

```
//OUTPUT DD UNIT=2520,DCB=(STACK=2,MODE=E)
```

Example 2. Creating a Data Set on a Card Punch

Creating Data Sets on Magnetic Tape

Tape data sets are created using combinations of the DD statement parameters UNIT, LABEL, DSNAME, DCB, VOLUME, and DISP.

UNIT: Required, except when volumes are requested using VOLUME=REF. You can assign a unit by specifying its address, type, or group name, or by requesting unit affinity with an earlier data set. You can also request multiple output units and defer volume mounting with this parameter.

LABEL: Required when the tape does not have standard labels, and when the data set does not reside first on the reel. It is also used to assign a retention period and password protection.

DSNAME: Required for data sets that are to be cataloged or used by a later job.

DCB: Required only when data control block information is not completely specified in the processing program. Usually, such attributes as the logical record length (LRECL) and buffering technique (BFTEK) will have been specified in the processing program. Other attributes, such as the maximum block size (BLKSIZE) and the tape recording technique (TRTCH), are more appropriately specified in the DD statement. Valid DCB subparameters are listed in Appendix B.

VOLUME: Optional, you can use this parameter to request specific volumes. If you use VOLUME=REF, and want to save existing data sets on the specified volume(s), you must indicate the data set sequence number in the LABEL parameter.

DISP: Required for data sets that are to be cataloged, passed, or kept.

Examples 3 through 8 illustrate valid DD statements for creating data sets on magnetic tape.

```
//OUTPUT2  DD  UNIT=2400-2
```

Example 3. Creating a Temporary Data Set on Labeled Tape

```
//OUTPUT2  DD  UNIT=2400,LABEL=(,NL)
```

Example 4. Creating a Temporary Data Set on Unlabeled Tape

```
//OUTPUT2  DD  UNIT=2400-2,DSNAME=A.B.C,DISP=(,CATLG),                X
//               LABEL=RETPD=0090
```

Example 5. Creating and Cataloging a Tape Data Set

```
//OUTPUT2  DD  DSNAME=&WORK,VOLUME=REF=*.STEP1.OUTPUT,                X
//               DISP=(,PASS),LABEL=(,NL)
```

Example 6. Creating a Temporary Data Set on Unlabeled Tape, Using VOLUME=REF

```
//OUTPUT   DD  DSNAME=ALPHA,UNIT=2400,DISP=(,KEEP),LABEL=2,           X
//               VOLUME=SER=T2
```

Example 7. Creating and Keeping a Data Set Second in Sequence on a Labeled Tape

```
//OUTPUT2  DD  UNIT=2400-2,DCB=(TRTCH=C,LRECL=256,DEN=1)
```

Example 8. Creating a Temporary Data Set Having an Incomplete Data Control Block

Creating Sequential (BSAM or QSAM) Data Sets on Direct-Access Devices

Sequential data sets are created using combinations of the DD statement parameters UNIT, DSNAME, VOLUME, LABEL, DISP, DCB, and one c the space allocation parameters SPACE, SPLIT, or SUBALLOC.

UNIT: Required, except when volumes are requested using VOLUME=REF or space is allocated using SPLIT or SUBALLOC. You can assign a unit by specifying its address, type, or group name, or by requesting unit affinity.

DSNAME: Required for all but temporary data sets.

LABEL: Required if you want to assign a retention period or password protection.

DCB: Required only when data control block information is not completely specified in the processing program. Usually, such attributes as the logical record length (LRECL) and buffering technique (BFTEK) will have been specified in the processing program. Other attributes, such as the maximum block size (BLKSIZE) and the number of buffers (BUFNO) are more appropriately specified in the DD statement. Valid DCB subparameters are listed in Appendix B.

VOLUME: Optional. You can use this parameter to request specific volumes, multiple nonspecific volumes, and to specify PRIVATE and RETAIN.

DISP: Required for data sets that are to be cataloged, passed, or kept.

SPACE, SPLIT, SUBALLOC: One of these is required for all new direct-access data sets.

Examples 9 through 14 illustrate valid DD statements for creating sequential data sets on direct-access devices.

```
//OUTPUT3  DD  UNIT=2311,SPACE=(TRK,(10,2))
```

Example 9. Creating a Temporary Data Set on Disk

```
//OUTPUT3  DD  DSNAME=&BUF,DISP=(,PASS),SPLIT=10
```

Example 10. Creating a Temporary Disk Data Set That Shares Cylinder Space With the Preceding Data Set

```
//OUTPUT3  DD  UNIT=2301,DCB=(BLKSIZE=1026,RECFM=FB),                X
//             SPACE=(1026,(30,3),,CONTIG,ROUND)
```

Example 11. Creating a Temporary Drum Data Set, With Space Allocation in Blocks

```
//OUTPUT3  DD  SUBALLOC=(TRK,(5,1),STEP1.OUTPUT)
```

Example 12. Creating a Temporary Disk Data Set, Using Suballocation Technique

```
//OUTPUT3  DD  DSNAME=ALPHA,UNIT=2311,DISP=(,KEEP),                  X
//             SPACE=(1024,(200,10)),VOLUME=(PRIVATE,RETAIN,SER=D02)
```

Example 13. Creating and Keeping a Data Set on a Private Disk Pack

```
//OUTPUT3  DD  DSNAME=X.Y.Z,VOLUME=REF=*.STEP1.OUTPUT,DISP=(,CATLG), X
//             SPACE=(CYL,(2,1)),LABEL=EXPDT=67365
```

Example 14. Creating and Cataloging a Disk Data Set, Using VOLUME=REF

Creating Direct (BDAM) Data Sets

Direct (BDAM) data sets are created using the same subset of DD statement parameters as sequential data sets, with the exception of the SPLIT parameter. To distinguish a BDAM data set, you must code a DCB parameter containing one of the attributes DSORG=DA or DSORG=DAU. Valid DCB subparameters for BDAM data sets are listed in Appendix B.

Creating Partitioned (BPAM) Data Sets

Partitioned (BPAM) data sets are created using the same subset of DD statement parameters as sequential data sets, except that BPAM data sets cannot occupy a split cylinder. To distinguish a BPAM data set, you must code the number of directory blocks in the space allocation parameter. Valid DCB subparameters for BPAM data sets are listed in Appendix B.

Example 15 illustrates a DD statement for creating a BPAM data set with a 12-block directory.

```
//OUTPUT3  DD  DSNAME=X.Y.Z,VOLUME=REF=*.STEP1.OUTPUT,                X
//            DISP=(,CATLG),SPACE=(CYL,(20,,12))
```

Example 15. Creating and Cataloging a Partitioned Data Set, Using VOLUME=REF

Creating Indexed Sequential (BISAM and QISAM) Data Sets

Indexed sequential (ISAM) data sets are created using combinations of the DD statement parameters UNIT, DSNAME, VOLUME, LABEL, DISP, DCB, and SPACE. ISAM data sets occupy three areas of space: an index area that contains master and cylinder indexes: a prime area that contains the data records and track indexes; and an optional overflow area to hold additional records when the prime area is exhausted. Detailed information on creating and retrieving indexed sequential data sets is presented in Appendix D.

Creating Data Sets in the Output Stream

New data sets can be written on a system output device, much the same as messages. When using a sequential scheduler, you direct a data set to the output stream with the SYSOUT and DCB parameters.

SYSOUT: Required. Specify the standard output class, A.

DCB: Required only if complete data control block information has not been specified in the processing program.

Example 16 illustrates a DD statement for routing a data set through the output stream using a sequential scheduler.

```
//OUTPUT5  DD  SYSOUT=A
```

Example 16. Creating a SYSOUT Data Set (Sequential Scheduler)

When you are using a priority scheduler, data sets are not routed directly to a system output device. They are stored by the processing program on an intermediate direct-access device and later written on a system output device. In addition to the SYSOUT and DCB parameters, DD statements defining a data set of this type can also contain UNIT and SPACE parameters. All other parameters must be absent.

SYSOUT: Required. Specify the output class through which you want the data set routed. Output classes are identified by a single alphameric character.

DCB: Required only if complete data control block information has not been specified in the processing program. Data control block information is used when the data set is written on an intermediate direct-access volume and read by the output writer. However, the output writer's own DCB attributes are used when the data set is written on the system output device. Valid DCB parameters are listed in Appendix B.

UNIT: Optional. Assign an intermediate direct-access device. A default device is assigned if you omit this parameter.

SPACE: Optional. Estimate the amount of direct-access space required. A default estimate is assumed if you omit this parameter.

Example 17 illustrates a DD statement for routing a data set through an output stream using a priority scheduler.

```
//OUTPUT5  DD  SYSOUT=F,UNIT=2311,SPACE=(TRK,(10,2))
```

Example 17. Creating a SYSOUT Data Set (Priority Scheduler)

RETRIEVING EXISTING DATA SETS

Data sets that exist before the job step that uses them are retrieved using subsets of the DD statement parameters. The contents of each subset depend on the disposition assigned to the data set when it was created. Existing data sets can be used for both input and output purposes. In the latter case, the existing data set is extended with additional output instead of being read as input.

Retrieving Cataloged Data Sets

Input data sets that were assigned a disposition of CATLG, or were cataloged by the IEHPROGM utility program, are retrieved using the DD statement parameters DSNAME, DISP, LABEL, and DCB. The device type, volume serial number, and data set sequence number (if tape) are stored in the catalog.

DSNAME: Required. Identify the data set by its cataloged name. If the catalog contains more than one index level, the data set name must be fully qualified.

DISP: Required. Give the data set's status, OLD or SHR, and indicate how you want it treated after its use, unless you want it to remain cataloged.

LABEL: Required only if the data set does not have a standard label.

DCB: Required only if complete data control block information is not specified by the processing program and the data set label. To save recoding time, you can copy DCB attributes from an existing DCB parameter and modify them if necessary. Valid DCB subparameters are listed in Appendix B.

Examples 18 and 19 illustrate valid DD statements for retrieving cataloged data sets.

```
//INPUT2   DD  DSNAME=A.B.C,DISP=(OLD,UNCATLG)
```

Example 18. Retrieving and Uncataloging a Data Set

```
//INPUT2   DD  DSNAME=X.Y.Z,DISP=SHR
```

Example 19. Retrieving a Disk Data Set, Which Can Be Shared by Another Job

Note: In addition to the disposition UNCATLG, you can pass a cataloged data set to a later step (PASS), or delete it (DELETE).

Retrieving Noncataloged (Kept) Data Sets

Input data sets that were assigned a disposition of KEEP are retrieved by their tabulated name and location, using the DD statement parameters DSNAME, UNIT, VOLUME, DISP, LABEL, and DCB.

DSNAME: Required. Identify the data set by the name assigned to it when it was created.

UNIT: Required, unless VOLUME=REF is used. Identify the unit by its address, type, or group name. If the data set requires more than one unit, give the number of units. You can also request deferred volume mounting and unit separation with this parameter.

VOLUME: Required. Identify the volume or volumes with serial numbers or, if the data set was retrieved earlier in the same job, with VOLUME=REF. If you want the volume to be PRIVATE, specify PRIVATE. If you want a private volume to remain mounted until a later job step uses it, specify RETAIN.

DISP: Required. Give the data set's status, OLD or SHR, and indicate how you want it treated after its use.

LABEL: Required if the data set does not have a standard label. If the data set resides with others on tape, you must give its sequence number.

DCB: Required for all indexed sequential data sets. Otherwise, required only if complete data control block information is not supplied by the processing program and the data set label. To save recoding time, you can copy DCB attributes from an existing DCB parameter, and modify them if necessary. Valid DCB subparameters are listed in Appendix B.

Examples 20 through 22 illustrate valid DD statements for retrieving noncataloged data sets.

```
//INPUT3   DD  DSNAME=ALPHA,UNIT=2311,DISP=SHR,                    X
//             VOLUME=SER=(P12,P14)
```

Example 20. Retrieving a Noncataloged Data Set, Which Can Be Shared by Another Job

```
//INPUT3   DD  DSNAME=BETA,UNIT=2400,LABEL=(2,BLP),                X
//             DISP=(OLD,DELETE),VOLUME=(PRIVATE,RETAIN,SER=T3),   X
//             DCB=(*.STEP1.OUTPUT,DEN=2)
```

Example 21. Retrieving and Deleting a Noncataloged Data Set, With Bypass Label Processing

```
//INPUT    DD  DSNAME=MHB,DCB=DSORG=IS,UNIT=(2311,3),              X
//             DISP=(OLD,KEEP),VOLUME=SER=(334,335,336)
```

Example 22. Retrieving an Indexed Sequential Data Set on Three Disks

Retrieving Passed Data Sets

Input data sets used in a previous job step and passed are retrieved using the DD statement parameters DSNAME, DISP, and UNIT. The data set's unit type, volume location, and label information remain available to the system from the original DD statement.

DSNAME: Required. Identify the original data set by either its name or the DD statement reference term *.stepname.ddname. If the original DD statement occurs in a cataloged procedure, you must include the procedure step name in the reference term.

DISP: Required. Identify the data set as OLD, and indicate how you want it treated after its use.

UNIT: Required only if you want more than 1 unit allocated to the data set.

Example 23 illustrates a valid DD statement for retrieving a passed data set.

```
//INPUT4   DD  DSNAME=*.STEP1.OUTPUT,DISP=(OLD,DELETE)
```

Example 23. Retrieving a Passed Data Set

Extending Data Sets With Additional Output

A processing program can extend an existing data set by adding records to it instead of reading it as input. Such a data set is retrieved using the same subsets of DD statement parameters described under the preceding three topics, depending on whether it was cataloged, kept, or passed when created. In each case, however, the DISP parameter must indicate a status of MOD. When MOD is specified, the system positions the appropriate read/write head after the last record in the data set. If you indicate a disposition of CATLG for an extended data set that is already cataloged, the system updates the catalog to reflect any new volumes caused by the extension.

Examples 24 through 26 illustrate valid DD statements for retrieving data sets to be extended by the processing program.

```
//INPUT5   DD  DSNAME=A.B.C,DISP=(MOD,CATLG)
```

Example 24. Extending and Recataloging a Data Set

```
//INPUT5   DD  DSNAME=ALPHA,UNIT=2311,VOLUME=SER=P12,                X
//            DISP=(MOD,KEEP)
```

Example 25. Extending and Keeping a Noncataloged Data Set

```
//INPUT5   DD  DSNAME=*.STEP1.OUTPUT,DISP=(MOD,PASS)
```

Example 26. Extending and Passing a Passed Data Set

Retrieving Data Through an Input Stream

Data sets in the form of decks of cards or groups of card images can be introduced to the system through an input stream by interspersing them with control statements. To define a data set in the input stream, mark the beginning of the data set with a DD statement and the end with a delimiter statement. The DD statement must contain one of the parameters * or DATA. Use DATA if the data set contains job control statements.

Examples 27 and 28 illustrate valid DD statements for defining a data set in the input stream.

```
//INPUT6   DD  *
```

Example 27. Retrieving a Data Set Through the Input Stream

```
//INPUT6   DD  DATA
```

Example 28. Retrieving a Data Set That Contains Control Statements Through the Input Stream

Notes:

When you use a sequential scheduler:

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

When you use a priority scheduler:

- Each job step and procedure step can be associated with several data sets in an input stream. All such data sets except the first in the job must be preceded by DD * or DD DATA statements.
- The characters in the records must be coded in BCD or EBCDIC.

ADDITIONAL DD STATEMENT FACILITIES

Parameters and fields of the DD statement can be coded in special ways to perform functions other than simply creating and retrieving data sets. Variations of the name field allow you to:

- Concatenate two or more input data sets.
- Use a private library.
- Define data sets used for ABEND dumps.

The DUMMY parameter coded by itself or with other parameters can be used to bypass input/output operations on data sets. The DSNAME parameter, when coded in a special way, can be used in combination with other DD statement parameters to create and retrieve generation data groups. The AFF and SEP parameters allow you to make efficient use of channels in certain situations.

Concatenating Data Sets

Several input data sets, each of which may reside on a different volume, can be read as if they were a single data set through the technique of concatenation. This technique makes it possible for a processing program to get its input from several different types of devices. Concatenated data sets are read in the order of appearance of their DD statements in the input stream.

To concatenate data sets, simply omit the ddnames from all DD statements except the first in sequence. Example 29 illustrates a group of DD statements defining concatenated data sets, including a data set in the input stream.

```
//INPUT    DD  DSNAME=A.B.C,DISP=(OLD,DELETE)
//         DD  DSNAME=X.Y.Z,DISP=OLD,LABEL=(,NL)
//         DD  DSNAME=ALPHA,UNIT=2311,VOLUME=SER=P12,                X
//             DISP=(OLD,DELETE)
//         DD  *

        ---Data Cards---
/*
```

Example 29. Concatenating Data Sets

Using a Private Library

Processing programs that are used most frequently 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, you must first make the library available to a job. The simplest way to do this is by placing a special DD statement at the beginning of the job. When the operating

system encounters this statement, it effectively concatenates the private library with the system library, for the duration of the job. As the job progresses, the system searches for each program, first in the private library, and then in the system library.

The DD statement must contain the special ddname JOBLIB, and must appear immediately after the JOB statement of the job to which it pertains. The operand field, at minimum, must contain the DSNAME and DISP parameters. The DISP parameter must be coded DISP=(OLD,PASS) or DISP=(SHR,PASS), so that the library remains available throughout the job. (The system assumes DISP=(OLD,PASS) if you code DISP=OLD.) Other parameters should be coded according to requirements for retrieving data sets, as discussed in an earlier chapter.

Example 30 illustrates a valid sequence of control statements for making a private library available to a job.

```
//PAYROLL  JOB [JOB statement parameters]
//JOBLIB   DD  DSNAME=PRIVATE.LIB1,DISP=(OLD,PASS)
//STEP1    EXEC [EXEC statement parameters]
```

Example 30. Retrieving a Cataloged Private Library

As with ordinary DD statements, you can arrange a sequence of JOBLIB DD statements so that the private libraries they define are effectively read as one. The libraries are searched in the order in which the DD statements appear, with the system library searched last. To concatenate private libraries, omit the ddname from all the DD statements except the first. The first statement must specify a ddname of JOBLIB. The entire group must appear immediately after the JOB statement, and before the first EXEC statement.

Defining Data Sets Used 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 basic sequential (BSAM) data set. The processing program must not make a reference to such a data set. If more than one special ddname is included in a job step, all but the first 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. The SYSUDUMP DD statement provides only a dump of the problem program storage area.

If you choose to have the dump routed through the output stream and written on a system output device, you must include the SYSOUT parameter. If you are using a priority scheduler, you can also include the UNIT and SPACE parameters to define the intermediate direct-access device. Appropriate parameters for this type of output are discussed in an earlier topic titled "Routing a Data Set through the Output Stream."

If you choose, instead, to store the dump and write it at a later time, the SYSABEND DD statement must identify the data set, provide unit and volume information, and give a disposition of KEEP or CATLG. If the unit is direct-access, you must also include one of the space allocation parameters. Example 31 illustrates a sample set of job steps that makes use of the abnormal termination dumping facilities.

```
//STEP1  EXEC  PGM=PROGRAM1
//SYSABEND DD  DSNAME=DUMP,UNIT=2311,DISP=(,KEEP),                    X
//            VOLUME=SER=1234,SPACE=(TRK,(110,10))
       .
     Other DD Statements
       .
//STEP2  EXEC  PGM=PROGRAM2
//SYSABEND DD  DSNAME=*.STEP1.SYSABEND,DISP=(MOD,KEEP),               X
//            VOLUME=REF=*.STEP1.SYSABEND
       .
     Other DD Statements
       .
//STEP3  EXEC  PGM=PHONY,COND=((5,LT,STEP1),(5,LT,STEP2))
//SYSABEND DD  DSNAME=*.STEP1.SYSABEND,DISP=(OLD,DELETE),             X
//            VOLUME=REF=*.STEP1.SYSABEND
```

Example 31. Using SYSABEND DD Statements

The SYSABEND DD statement in STEP2 takes advantage of the direct-access space acquired in STEP1 by indicating MOD. Note that the space request in STEP1 is large (110 tracks) so that the dumping operation is not inhibited due to insufficient space.

The purpose of STEP3 is to delete the data set and free the space acquired for dumping if all steps were executed successfully. Each step that uses the dumping facility should issue a return code that indicates its results. In Example 31, steps preceding STEP3 issue a return code less than or equal to 5 if the step was completed successfully, and greater than 5 if at least one task was terminated abnormally. Thus, STEP3 is executed only if STEP1 and STEP2 were successfully completed (i.e., all return codes were less than 5).

The program named PHONY can be simply a load module in the system library consisting of a single RETURN macro-instruction.

Bypassing I/O Operations on a Data Set

The DUMMY parameter, a DD statement positional parameter, offers you the facility to bypass input/output operations, space allocation, and disposition of data sets referred to by the basic or queued sequential access methods. This facility can be used to suppress the writing of certain output data sets, such as assembler listings, and to update new master files with a dummy detail file. Bypassing operations on noncritical data sets also results in a saving of time when you are testing a program. To use this facility, code the word DUMMY first in the operand field. Later you can remove this word and the DD statement will assume its ordinary functions of allocation and disposition, and I/O operations will be performed.

A read operation performed on a dummy data set results in an end of data set exit. A write request is recognized, but no data is transferred. Example 32 illustrates the use of the DUMMY parameter to establish a dummy data set.

```
//OUTPUT3  DD  DUMMY,DSNAME=X.Y.Z,DISP=(,CATLG),                      X
//            UNIT=2311,SPACE=(TRK,(10,2))
```

Example 32. Bypassing I/O Operations on a Data Set

The data set defined in this example will not be cataloged after its use or allocated a device or direct-access space until the word DUMMY is removed.

Creating and Retrieving Generation Data Sets

A cataloged data set that is periodically processed, such as a weekly payroll, can be grouped with its earlier generations to form a named generation group. It is convenient to form such a group in that each member can be addressed by a simple generation number. The entire group of data sets is associated with a generation group name. You identify individual generations by coding the group name followed by a number enclosed in parentheses, i.e., DSNAME=groupname(number). The current generation, that is, the most up-to-date generation before the start of a job, is addressed throughout the job by the group name followed by (0). You address the generation immediately preceding the current one by coding DSNAME=groupname(-1), and the generation immediately succeeding the current one by coding DSNAME=groupname(+1). (New generations must always be assigned a disposition of CATLG.) Other past or future generations can be addressed by decreasing or increasing the generation number in increments of 1. You refer to a generation by the same number throughout a job, even though one or more new generations are created. Generation numbers are updated to their new values at the end of the job.

Example 33 shows valid DD statements for retrieving the current generation of a group named WEEKLY.PAYROLL, and creating a new generation.

```
          :
//INPUT   DD  DSNAME=WEEKLY.PAYROLL(0),DISP=OLD
          :
//OUTPUT  DD  DSNAME=WEEKLY.PAYROLL(+1),DISP=(,CATLG)
          :
```

Example 33. Retrieving and Creating Generation Data Sets

Subsequent references to either of these data sets in the same job must use the same generation numbers. If other generations are created within the same job, they must be numbered (+2), (+3), etc. At the end of the job the data defined by INPUT becomes (-1) and the data set defined by OUTPUT becomes (0), provided other new generations were not created.

Notes:

- You can concatenate the entire group of data sets by specifying the group name without a generation number, e.g., DSNAME=WEEKLY.PAYROLL.
- You cannot specify DISP=SHR when retrieving a generation data set.

Reference:

- For details on creating a generation data group, see the chapter "Modifying System Control Data" of the publication IBM System/360 Operating System: Utilities.

Optimizing Channel Usage

Optimum channel usage can be obtained in some job steps by requesting that certain data sets be assigned separate channels from others. This facility should be used only when a significant saving of time can be realized by using separate channels. It considerably restricts device allocation and may result in unnecessary dismounting of volumes.

The system treats channel separation and affinity requests on an "if available" basis, that is, they will be recognized only if enough channels are available. If one request cannot be satisfied, all such requests in the step may be ignored. In addition, requests are ignored if the automatic volume recognition feature is used. Example 34 illustrates a job step containing channel requests.

```
//STEP1    EXEC  PGM=CONVERT
//INPUT1   DD    DSNAME=A.B.C,DISP=OLD
//INPUT2   DD    DSNAME=X.Y.Z,DISP=OLD
//BUF      DD    UNIT=2400,SEP=(INPUT1,INPUT2)
//OUTPUT   DD    DSNAME=ALPHA,UNIT=TAPE,DISP=(,KEEP),AFF=BUF
```

Example 34. Requesting Channel Separation and Affinity

If enough channels are available, the temporary data sets defined by the DD statements BUF and OUTPUT are assigned a channel other than the ones used by data sets A.B.C and X.Y.Z. Note that the data sets defined by BUF and OUTPUT may or may not use the same channel.

USING CATALOGED PROCEDURES

Applications that require many control statements and are used on a regular basis can be considerably simplified through the use of cataloged procedures. A cataloged procedure is a set of job control statements that has been placed in a partitioned data set known as the procedure library. (The procedure library is a system data set named SYS1.PROCLIB.) You retrieve a cataloged procedure from the library by using its member name in an EXEC statement in the input stream. Other control statements in the input stream can be used to temporarily override or add to the control statements in a procedure.

Establishing Cataloged Procedures

Cataloged procedures contain one or more EXEC statements, each followed by associated DD statements. Each EXEC statement and its associated DD statements represent a procedure step. EXEC statements identify programs to be executed. Cataloged procedures cannot contain:

- EXEC statements referring to other cataloged procedures.
- JOB, command, delimiter, or null statements.
- DD statements with the ddname JOBLIB.
- DD statements with * or DATA coded in the operand field.

You add cataloged procedures to the procedure library by using the IEBUPDTE utility program. You can also use this program to permanently modify existing procedures. A description of this program and examples of its use are contained in the publication IBM System/360 Operating System: Utilities.

Overriding EXEC Statements in Cataloged Procedures

To override or add to the parameters on an EXEC statement in a cataloged procedure, you must include the procedure step name as part of the keywords on the EXEC statement that calls the procedure, i.e., TIME.procstepname. Each such keyword can appear as many times as there are steps in the procedure. However, you must code all overriding parameters for one procedure step before those of the next step.

If you wish to override all EXEC statement parameters in a cataloged procedure with a single set of parameter values, code the overriding keywords by themselves. Overriding parameters of this type modify as follows:

- PARM -- applies to the first step in the procedure and nullifies all other PARM parameters.
- COND, ACCT -- apply to every step in the procedure.
- TIME, REGION -- override all TIME and REGION parameters and apply to the entire procedure.

Overriding and Adding DD Statements

To override the DD statement parameters in a cataloged procedure, or to add data sets to the procedure, you must include some DD statements in the input stream. These DD statements must have a name field of the form procstepname.ddname, e.g., STEP1.OUTPUT. The terms "procstepname" and "ddname" identify the procedure step and DD statement that you are overriding. If you are adding a data set to the procedure, the term "ddname" must be different from other ddnames in the procedure step.

There are a few rules you should keep in mind when overriding or adding to a cataloged procedure step:

1. Overriding DD statements must be in the same order in the input stream as the corresponding DD statements in the cataloged procedure.

2. DD statements to be added must follow overriding DD statements for the step.

3. If you are using a sequential scheduler and one of the overriding or additional DD statements has an * in its operand field, it must be last.

To override a parameter in a procedure DD statement, either (1) recode the entire parameter on the overriding statement, modifying it as you wish, or (2) code a suitable replacement for the parameter, such as SPLIT for SPACE. The DCB parameter is an exception to these rules; you need only recode the DCB attributes you wish to modify. Parameters that you do not wish to override need not be recoded.

To nullify a keyword parameter, code the keyword followed by an equal sign in the overriding DD statement, e.g., UNIT=. To nullify a DUMMY parameter, code the DSNAME parameter in the overriding DD statement, but do not use the data set name NULLFILE.

Example 35 shows an existing cataloged procedure named ANALYSIS. This procedure is used and modified by the statements in Example 36 to produce the temporary result as shown in Example 37.

```
//LOOKUP   EXEC  PGM=SEARCH
//IN1      DD    DSNAME=A.B.C,DISP=OLD
//OUT1     DD    UNIT=2311,SPACE=(TRK,(10,2)),DISP=(,PASS)
//REDUCE   EXEC  PGM=TRUNCATE
//IN2      DD    DSNAME=*.LOOKUP.OUT1,DISP=(OLD,DELETE)
//WORK     DD    UNIT=TAPE
//OUT2     DD    UNIT=2311,SPACE=(TRK,(5,1)),DISP=(,PASS)
//DISPLAY  EXEC  PGM=PRINT
//IN3      DD    DSNAME=*.REDUCE.OUT2,DISP=(OLD,DELETE)
//OUT3     DD    SYSOUT=A
```

Example 35. Modifying a Cataloged Procedure -- The Procedure

The procedure in Example 35 comprises three steps, each of which receives input data from the catalog or the previous step, processes the data, and places it in an output data set. The second step makes use of a temporary work file on tape.

```
//STEP1         EXEC   ANALYSIS
//LOOKUP.OUT1   DD     UNIT=2400,DISP=
//REDUCE.WORK   DD     UNIT=180
//REDUCE.XTRA   DD     UNIT=181
//DISPLAY.IN3   DD     DISP=(OLD,KEEP)
```

Example 36. Modifying a Cataloged Procedure -- The Input Stream

The job step in Example 36 uses the procedure and modifies it by:

- Changing a disk to a tape and nullifying a disposition (implying a disposition of DELETE).
- Specifying a specific unit for the work file.
- Adding an additional work file.
- Changing a disposition.

As a result of these modifications, the procedure temporarily appears as shown in Example 37. As a result of the unit change in the DD statement OUT1, the SPACE parameter is ignored.

```
//LOOKUP    EXEC  PGM=SEARCH
//IN1       DD    DSNAME=A.B.C,DISP=CLD
//OUT1      DD    UNIT=2400,SPACE=(TRK,(10,2))
//REDUCE    EXEC  PGM=TRUNCATE
//IN2       DD    DSNAME=*.LOOKUP.OUT1,DISP=(OLD,PASS)
//WORK      DD    UNIT=180
//XTRA      DD    UNIT=181
//OUT2      DD    UNIT=2311,SPACE=(TRK,(5,1)),DISP=(,PASS)
//DISPLAY   EXEC  PGM=PRINT
//IN3       DD    DSNAME=*.REDUCE.OUT2,DISP=(OLD,KEEP)
//OUT3      DD    SYSOUT=A
```

Example 37. Modifying a Cataloged Procedure -- The Result

Using the DDNAME Parameter

The DDNAME parameter, used primarily in cataloged procedures and job steps that call procedures, allows you to define a dummy data set that can, at a later point in the same job step, assume real data set characteristics. This facility is primarily used when a cataloged procedure receives data from an input stream.

To use this facility, code DDNAME=ddname in the operand field of a DD statement. This DD statement represents a data set whose definition is postponed until a DD statement with a matching ddname in its name field is encountered. At this point, all of the data set characteristics defined in the matching statement are applied to the original statement.

There are a few rules you should keep in mind as you use the DDNAME parameter:

1. You cannot make a backward reference (i.e., *.ddname) to a DD statement whose ddname matches a previous DDNAME parameter. Also you cannot place unnamed DD statements after such a statement (i.e., concatenate).

2. You can make a backward reference to the original DD statement containing the DDNAME parameter. If such a reference occurs before a matching ddname is found, it refers to a dummy data set. You may also place unnamed DD statements after the original statement containing the DDNAME parameter.

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 DD statement that defines a private library (i.e., a DD statement named JOBLIB).

Example 38 shows a cataloged procedure named PROC1 that uses the DDNAME facility. This procedure is used and modified by the statements in Example 39.

```
//STEP1     EXEC     PGM=PROGRAM1
//DD1       DD       DDNAME=INPUT
//DD2       DD       [DD statement parameters]
//DD3       DD       [DD statement parameters]
//STEP2     EXEC     PGM=PROGRAM2
```

Example 38. Using the DDNAME Parameter in a Cataloged Procedure -- The Procedure

```
              .
              .
//STEPA           EXEC     PROC1
//STEP1.INPUT     DD       *

     --  Data Cards  --
              .
              .
```

Example 39. Using the DDNAME Parameter in a Cataloged Procedure -- The Input Stream

The procedure data set defined by DD1 assumes the characteristics of a data set in the input stream when the matching ddname INPUT is encountered.

Examples 40 and 41 illustrate another common use of the DDNAME parameter. The cataloged procedure (PROC2) shown in Example 40 is called and modified by the job step shown in Example 41.

```
//STEP1     EXEC     PGM=PROGRAM1
//DD1        DD      [DD statement parameters]
//DD2        DD      [DD statement parameters]
//STEP2     EXEC     PGM=PROGRAM2
              .
              .
```

Example 40. Using the DDNAME Parameter in an Input Stream -- The Procedure

```
                .
                .
//STEPA         EXEC      PROC2
//STEP1.DD1      DD       DDNAME=SKIP
//STEP1.DD2      DD       [Overriding parameters]
                .
                .
//SKIP           DD       *
            --  Data Cards  --
                .
                .
```

Example 41. Using the DDNAME Parameter in an Input Stream -- The Input Stream

According to the rules of overriding cataloged procedures, DD statements in the input stream must appear in the same order as their counterparts in the procedure, and yet an overriding DD * statement must appear last. The DDNAME parameter shown in Example 41 solves this problem by postponing the data in the input stream until last while maintaining the correct overriding order.

APPENDIX A: UNIT TYPES

The UNIT parameter of the DD statement can identify an input or output unit by its actual address, its type number, or its group name. Type numbers, automatically established at system generation, correspond to units configured into your system. Type numbers and corresponding units are listed here for your convenience.

Tape Units

Unit Type	Unit
2400	any 2400 Nine-Track Magnetic Tape Drive
2400-1	any 2400 Magnetic Tape Drive with Seven-Track Compatibility
2400-2	2400 Magnetic Tape Drive with Seven-Track Compatibility and Data Conversion
2400-4,-5,-6	any 2400 Nine-Track Magnetic Tape Drive with a density of 1600 bytes per inch. Optional feature allows for a density of 800 bytes per inch. Model number denotes rate of data transmission.

Direct Access Units

Unit Type	Unit
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

Unit Record Equipment

Unit Type	Unit
1052	1052 Printer-Keyboard
1403	1403 Printer or 1404 Printer (continuous form only)
1442	1442 Card Read Punch
1443	any 1443 Printer
2501	2501 Card Reader
2520	2520 Card Read Punch
2540	2540 Card Read Punch (read feed)
2540-2	2540 Card Read Punch (punch feed)
2671	2671 Paper Tape Reader

Graphic Units

Unit Type	Unit
1053	1053 Printer
2250	2250 Display Unit, Model 1
2250-2	2250 Display Unit, Model 2
2250-3	2250 Display Unit, Model 3
2260	2260 Display Station (local attachment)
2280	2280 Film Recorder
2282	2282 Film Recorder/Scanner

APPENDIX B: DCB SUBPARAMETERS

The data control block associated with a data set is filled by a number of sources, one of which is the DD statement. The DCB parameter supplies missing or overriding attributes in the form of a list of subparameters. The glossary below lists the keywords that you can code in the DCB parameter, their definitions, and the values they may assume.

Table 3 supplies valid keywords and values that you can use with the indexed sequential, partitioned, direct, and sequential access methods. Subparameters that apply to the graphics access method (GNCP,GDSORG) and the teleprocessing access methods (BUFRQ,CPRI,INTVL,SOWA) do not appear in the table. Further information on DCB subparameters appears in the publication IBM System/360 Operating System: Supervisor and Data Management Macro-Instructions.

Glossary of DCB Subparameters

BFALN — Fullword (F) or doubleword (D) boundary alignment of each buffer.

BFTEK — Type of buffering (simple or exchange) to be supplied by the control program (S or E).

BLKSIZE — Maximum block size in bytes (a number).

BUFL — Length, in bytes, of each buffer to be obtained for a buffer pool (a number).

BUFNO — Number of buffers to be assigned to the data control block.

BUFRQ — Number of buffers to be read in advance from the direct-access device queue. (For use with teleprocessing access methods.)

CODE — Paper tape code in which the data is punched.

- I - IBM BCD perforated tape and transmission code (8 tracks)
- F - Friden (7 tracks)
- B - Burroughs (7 tracks)
- C - National Cash Register (8 tracks)
- A - ASCII (8 tracks)
- T - Teletype (5 tracks)
- N - No conversion

CPRI — Relative priority to be given to sending and receiving operations. (For use with teleprocessing access methods.)

CYLOFL — Number of tracks to be reserved on each cylinder to hold records that overflow from other tracks on that cylinder.

DEN — Tape recording density.

- 0 - 200 bits/inch (7-track only)
- 1 - 556 bits/inch (7-track only)
- 2 - 800 bits/inch

For 7-track tapes, 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 — Organization of the data set.

- PS - Physical sequential
- PSU - Physical sequential unmovable
- PO - Partitioned organization
- POU - Partitioned organization unmovable
- IS - Indexed sequential
- ISU - Indexed sequential unmovable
- DA - Direct-access
- DAU - Direct-access unmovable

EROPT — Option to be executed if an error occurs.

- ACC - Accept
- SKP - Skip
- ABE - Abnormal end of task

GDSORG — Organization of a graphic data set. (For use with the graphics access method.)

GNCP — Maximum number of input/output macro-instructions that will be issued before a WAIT macro-instruction. (For use with the graphics access method.)

INTVL — Number of seconds of intentional delay between passes through a polling list. (For use with teleprocessing access methods.)

KEYLEN — Length, in bytes, of a key (a number).

LIMCT — Search limits, in tracks or blocks, if the extended search option (OPTCD=E) is chosen (a number).

LRECL — Actual or maximum length, in bytes, of a logical record (a number).

MODE — Mode of operation (column binary or EBCDIC) for a card reader or card punch. (C or E.)

NCP — Maximum number of READ or WRITE macro-instructions issued before a CHECK macro-instruction (number of channel programs).

NTM — Number of tracks to be contained in the master index of an indexed sequential data set; required when a master index (OPTCD=M) is requested. Through this master index facility, extensive serial search through a large cylinder index can be avoided.

OPTCD — Optional services to be provided by the control program.

- W - Perform a WRITE validity check.
- F - Feedback will be requested in READ or WRITE macro-instructions in the program.
- E - Use extended search.
- R - Relative block addresses are used.
- A - Actual addresses are used.
- C - Use chained scheduling method.
- WC - Perform a WRITE validity check and use chained scheduling method.
- M - Create master indexes as required.
- Y - Use cylinder overflow areas.
- I - Delete marked records when new records are added to the data set.
- U - Universal character set.

PRTSP — Line spacing on a printer (0, 1, 2, or 3).

RECFM — Characteristics of the records in a data set. These can be used in combination, as required.

- U - Undefined-length records.
- V - Variable-length records.
- F - Fixed-length records.
- B - Blocked records.
- T - Track overflow used.
- S - Standard blocks - no truncated blocks or unfilled tracks within the data set.
- A - ASA control character.
- M - Machine code control character.

RKP — Relative position of the first byte of the record key within each logical record (a number).

SOWA — Size, in bytes, of the user-provided work areas. (For use with teleprocessing access methods.)

STACK — Stacker bin on a card reader or card punch (1 or 2).

TRTCH — Tape recording technique for seven-track tape. These values can be used in combination.

- C - Data conversion feature
- E - Even parity
- T - BCD to EBCDIC translation

Table 3. Valid DCB Subparameters

Keyword	BISAM	QISAM	BPAM	BDAM	BSAM	QSAM
BFALN=	F or D	F or D	F or D	F or D	F or D	F or D
BFTEK=						S or E
BLKSIZE=		number of bytes	number of bytes	number of bytes	number of bytes	number of bytes
BUFL=			number of bytes	number of bytes	number of bytes	number of bytes
BUFNO=	number of buffers	number of buffers	number of buffers	number of buffers	number of buffers	number of buffers
CODE=					see glossary	see glossary
CYLOFL=		number of tracks				
DEN=					0, 1, or 2 (Tape)	0, 1, or 2 (Tape)
DSORG=	IS or ISU	IS or ISU		DA or DAU	PS or PSU (Disk)	PS or PSU (Disk)
EROPT=						ACC,SKP or ABE
KEYLEN=		number of bytes	number of bytes	number of bytes	number of bytes	
LIMCT=			number of trks or blks			
LRECL=		number of bytes	number of bytes		number of bytes	number of bytes
MODE=					C or E (Rdr/Punch)	C or E (Rdr/Punch)
NCP=	number of chan'l pgms.		number of chan'l pgms.		number of chan'l pgms.	
NTM=		number of tracks				
OPTCD=		see glossary	see glossary	see glossary	see glossary	see glossary
PRTSP=					0,1,2,or 3 (Printer)	0,1,2,or 3 (Printer)
RECFM=		see glossary	see glossary	see glossary	see glossary	see glossary
RKP=		byte number				
STACK=					1 or 2 (Rdr/Punch)	1 or 2 (Rdr/Punch)
TRTCH=					C, E, or T (Tape)	C, E, or T (Tape)

The components and features available through job management depend on the scheduler your installation selects during system generation. A sequential scheduler is used with operating systems having a primary control program (PCP) and systems that provide multiprogramming with a fixed number of tasks (MFT). A priority scheduler is used with systems that provide multiprogramming with a variable number of tasks (MVT).

Table 4 shows the job management components available with each scheduler. Table 5 compares and contrasts the job management features supported by the two schedulers.

Table 4. Components of Job Management

Components		Scheduler	
		Sequential	Priority
Job Scheduler	Reader/Interpreter	1	1 (or more)
	Initiator	1	1 (or more)
	Queue Manager	-	1
	System Output Writer	-	1 (or more)
Master Scheduler		1	1

Table 5. Comparison of Job Management Features

Feature	Scheduler	
	Sequential	Priority
Maximum Number of Input Streams	One.	One for each reader/interpreter.
Data Sets in the Input Stream	The processing program reads any data present in the input stream. You can have one such data set for each job step.	Each reader/interpreter, operating as a separate task, places the data on a direct-access device for subsequent high-speed retrieval by the processing program. You can have more than one such data set for each job step.
Console Configuration Allowed	Alternate Console or Composite Console, but not both, can be specified.	Alternate Console, Composite Console, or a combination of the two, can be specified.
Operator Commands That can be Entered Through the Console	CANCEL DISPLAY MOUNT REPLY REQ SET SHIFT[1] START STOP UNLOAD VARY [1]Systems with MFT only.	All commands valid with a sequential scheduler except REQ and SHIFT plus the following: HALT HOLD LOG MODIFY RELEASE RESET WRITELOG

(Continued)

Table 5. Comparison of Job Management Features (Continued)

Feature		Scheduler	
		Sequential	Priority
Operator Command Processing		All commands except SET, START RDR, and START WTR are accepted as they are issued. SET, START RDR, and START WTR are accepted between job steps and are issued after first issuing a REQ command.	All commands are accepted as they are issued.
System Output	To the operator	All messages to the operator appear on the operator's console.	
	To the programmer (Job scheduler messages)	Job scheduler messages are automatically routed to the standard output class. They are issued between job steps and appear on the device currently associated with the standard output class.	An output writer, operating as a separate task, processes job scheduler messages. You can use the MSGCLASS parameter of the JOB statement to control the output class to which these messages are assigned: • If you specify the standard output class (A) or omit the MSGCLASS parameter, job scheduler messages are routed to a device associated with the standard ouput class. • If you specify another class, job scheduler messages are routed to a device associated with that class.
Processing Program Output	To the operator	The WTO (write to operator) and the WTOR (write to operator with reply) macro-instructions can be used for processing program communication with the operator.	
	To the programmer (SYSOUT data sets)	You can elect to have the processing program write its own output directly, or specify the SYSOUT parameter in the DD statement. Note: The processing program transfers SYSOUT data directly onto a system output device.	You can elect to have the processing program write its own output directly, or specify the SYSOUT parameter in the DD statement. Notes: The processing program places a data set to be processed by SYSOUT routines on a direct-access device. The DD statement for this data set must contain the SYSOUT parameter, and can optionally contain UNIT and SPACE parameters. (Default values for UNIT and SPACE are assigned if these parameters are absent.) System output writer programs transfer the data from the direct-access device to an output device associated with the class named in the SYSOUT parameter. Data sets are made available for SYSOUT processing at the end of the job, and are selected for processing according to the priority of their jobs.

APPENDIX D: CREATING AND RETRIEVING INDEXED SEQUENTIAL DATA SETS

Indexed sequential (ISAM) data sets are created and retrieved using special subsets of DD statement parameters and subparameters. They can occupy up to three different areas of space:

- Prime area -- This area contains data records and related track indexes. It exists for all ISAM data sets.
- Overflow area -- This area contains overflow from the prime area when new data records are added. It is optional.
- Index area -- This area contains master and cylinder indexes associated with the data set. It exists for any ISAM data set that has a prime area occupying more than one cylinder.

ISAM data sets must reside on direct-access volumes. Because an ISAM data set can be associated with more than one type of unit, it is not usually cataloged.

Creating an ISAM Data Set

ISAM data sets are created with from one to three DD statements. One of the statements must define the prime area. DD statements must define the areas in a specific order:

1. Index area.
2. Prime area.
3. Overflow area.

This order must be maintained if one of the statements is absent. The first or only DD statement defining the data set can contain a name field. Other statements must have a blank name field.

The subset of DD statement parameters used to create an ISAM data set excludes *, DATA, DUMMY, DDNAME, SYSOUT, SUBALLOC, and SPLIT. The remaining DD statement parameters -- DSNAME, UNIT, VOLUME, LABEL, DCB, DISP, SPACE, and SEP and AFF -- are all valid. However, you must follow certain restrictions in using these parameters.

DSNAME: Required. In addition to giving the data set name, the DSNAME parameter identifies the area you are defining, i.e., DSNAME=name(INDEX), DSNAME=name(PRIME), and DSNAME=name(OVFLOW).

Notes:

- If the data set is temporary, replace name with &name.
- If you are using only one DD statement to define the entire data set, use DSNAME=name(PRIME) or DSNAME=name.

UNIT: Required, unless VOLUME=REF is used. The first subparameter identifies a direct-access unit. If you include separate statements for the prime and index areas, you must request the same number of units for the prime area as there are volumes. You cannot specify DEFER on any of the statements. Another way of requesting units is by using the unit affinity subparameter, AFF.

Notes:

- DD statements for prime and overflow areas must indicate the same type of unit.
- The DD statement for the index area can indicate a unit type different than the others.

VOLUME: Optional. Can be used to request private volumes (PRIVATE), retain private volumes (RETAIN), or to make specific volume references (SER or REF).

LABEL: Optional. Can be used to indicate the use of both standard and user labels (SUL) and to specify a retention period (RETPD or EXPDT) and password protection (PASSWORD).

DCB: Required. Can be used to complete the data control block if it has not been completed by the processing program. You must include in the list of attributes DSORG=IS or DSORG=ISU, even though this attribute was provided in the processing program. If more than one DD statement is used to define the data set, the DCB parameters in the statements must not contain conflicting attributes.

DISP: Optional. Must be coded if you want to keep the data set (KEEP), catalog it (CATLG), or pass it to a later job step (PASS). An ISAM data set can be cataloged using CATLG only if all three areas are defined by the same DD statement.

Note:

- You can catalog ISAM data sets defined by more than one DD statement by using the system utility program IEHPROGM, provided all volumes reside on the same type of unit. The utility program IEHPROGM is described in the publication IBM System/360 Operating System: Utilities.

SPACE: Required. You must request space using either the recommended nonspecific allocation technique or the more restricted absolute track (ABSTR) technique. If more than one DD statement is used to define the data set, all must request space using the same technique.

If you use the nonspecific space allocation technique, space must be requested in units of cylinders (CYL). The quantity of space you request is assigned to the area identified in the DSNAME parameter. If you requested more than one unit, this quantity of space is allocated on each volume used by the data set. You cannot request incremental space for ISAM data sets. If you are using one DD statement to define both the index and prime areas, you can indicate the size of the index in the SPACE parameter of the DD statement that defines the prime area. The subparameters RLSE, MXIG, ALX, and ROUND cannot be used. You can, however, request contiguous space on each of the volumes occupied by the data set with the subparameter CONTIG. If CONTIG is coded on one of the statements, it must be coded on all of them.

If you use the absolute track technique of allocating space, the number of tracks must be equivalent to an integral number of cylinders. The address of the beginning track must correspond with the first track of a cylinder other than the first cylinder on a volume. If you requested more than one unit, space is allocated beginning at the specified address and continuing through the volume and onto the next volume until the request has been satisfied. If you are using one DD statement to define both the index and prime areas, you can indicate the size of the index (in tracks) in the SPACE parameter of the DD statement defining the prime area. This number must also be equivalent to an integral number of cylinders.

Notes:

- The first volume to be allocated for the prime area of an ISAM data set cannot be the volume from which the system is loaded (the IPL volume).

- Space can be requested on more than one volume only on the DD statement that defines the prime area.

SEP and AFF: Optional. You can request channel separation from earlier data sets on any of the DD statements in the group. If you wish to have areas of an ISAM data set written using separate channels, you must request units by their actual address, e.g., UNIT=190.

Example 42 illustrates a valid set of DD statements for creating an ISAM data set. In this example, each area is defined by a separate DD statement.

The manner in which the areas of an ISAM data set are arranged is based primarily on two criteria:

1. The number of DD statements used to define the data set.

2. The types of DD statements used (as reflected in the DSNAME parameter).

An additional criterion arises when you do not include a DD statement for the index area:

3. Is an index size coded in the SPACE parameter of the DD statement defining the prime area?

Table 6 illustrates the arrangements resulting from various permutations of the above criteria. In addition, it points out restrictions on the number and type of units that can be requested for each permutation.

```
//OUTPUT4  DD  DSNAME=MHB(INDEX),UNIT=2301,DCB=DSORG=IS,                    X
//             SPACE=(CYL,10,,CONTIG),DISP=(,KEEP)
//
//           DD  DSNAME=MHB(PRIME),DCB=DSORG=IS,UNIT=(2311,2),             X
//               VOLUME=SER=(334,335),DISP=(,KEEP),                        X
//               SPACE=(CYL,25,,CONTIG)
//
//           DD  DSNAME=MHB(OVFLOW),DCB=DSORG=IS,UNIT=2311,                X
//               VOLUME=SER=336,SPACE=(CYL,25,,CONTIG),DISP=(,KEEP)
```

Example 42. Creating an Indexed Sequential Data Set

Table 6. Area Arrangement for ISAM Data Sets

CRITERIA			RESTRICTIONS ON UNIT TYPES AND NUMBER OF UNITS REQUESTED.	RESULTING ARRANGEMENT OF AREAS
1. Number of DD statements	2. Types of DD statements	3. Index size coded?		
3	INDEX PRIME OVFLOW	-	PRIME and OVFLOW must specify the same unit type.	Separate index, prime, and overflow areas.
2	INDEX PRIME	-	None	Separate index and prime areas.
2	PRIME OVFLOW	No	Both statements must specify the same type of unit.	Prime area and overflow area with an index at its end.
2	PRIME OVFLOW	Yes	Both statements must specify the same unit type. The statement defining the prime area cannot request more than one unit.	Prime area with embedded index, and overflow area.
1	PRIME	No	None	Prime area with index at its end. Additional area, if any, used for overflow.
1	PRIME	Yes	Cannot request more than one unit.	Prime area with embedded index area.

Retrieving An ISAM Data Set

ISAM data sets are retrieved with the DD statement parameters DSNAME, UNIT, VOLUME, DCB, and DISP. Channel separation requests can be made using the SEP and AFF parameters. If all areas of the data set reside on the same type of unit, you can retrieve the entire data set with one DD statement. If the index resides on a different type of unit, you must use two DD statements.

DSNAME: Required. Identify the data set by its name. If it was passed from a previous step, identify it by a backward reference or its temporary name. Do not include the terms INDEX, PRIME, or OVFLOW.

UNIT: Required, unless the data set was passed on one volume. Identify the unit type. If the data set resides on more than one volume and all units are the same type, request the total number of units required by all areas. If the index area resides on a different type of unit, you must use two DD statements, each indicating the number of units of the specified type required.

VOLUME: Required, unless the data set was passed on one volume. Identify the volumes by their serial numbers (SER), listed in the same order as they were when the data set was created.

DCB: Required, unless the data set was passed. You can use this parameter to complete the data control block if it was not completed in the program. You must include either DSORG=IS or DSORG=ISU.

DISP: Required. Identify the data set as OLD or MOD and give its new disposition, if you wish to change its disposition.

Example 43 shows how to retrieve the ISAM data set created in Example 42.

```
//INPUT    DD  DSNAME=MHB,DCB=DSORG=IS,UNIT=2301,DISP=OLD
//           DD  DSNAME=MHB,DCB=DSORG=IS,UNIT=(2311,3),DISP=OLD,                X
//               VOLUME=SER=(334,335,336)
```

Example 43. Retrieving an ISAM Data Set

The JOB Statement

//Name	Operation	Operand	P/K	Comments
//jobname	JOB	[([acct#][,acctg information])]	P	Can be made mandatory
		[programmer's name]	P	Can be made mandatory
		[MSGLEVEL=1]	K	
		[COND=((condition),...)]	K	Maximum of 8 conditions
		[PRTY=n]	K	Priority scheduler only
		[MSGCLASS=x]	K	Priority scheduler only
		[REGION=nnnnnK]	K	Priority scheduler only

The EXEC Statement

//Name	Operation	Operand	P/K	Comments
//[stepname]	EXEC	{PGM=program name PGM=*.stepname.ddname [PROC=]procedure name}	P	
		[COND=((condition),...) COND.procstepname=((condition),...)]	K	Maximum of 8 conditions
		[PARM=value PARM.procstepname=value]	K	
		[ACCT=(acctg information) ACCT.procstepname=(acctg info)]	K	
		[TIME=(minutes,seconds) TIME.procstepname=(min,sec)]	K	Priority scheduler only
		[REGION=nnnnnK REGION.procstepname=nnnnnK]	K	Priority scheduler only

Legend:

P Positional parameter.
K Keyword parameter.
{} Choose one.
[] Optional; if more than one line is enclosed, choose one or none.

The DD Statement

//Name	Operation	Operand	Comments
//[ddname / procstepname.ddname]	DD	[DSNAME=identification]	See Chart 3 for subparameters
		[UNIT=(unit information)]	See Chart 3 for subparameters
		[VOLUME=(volume information)]	See Chart 3 for subparameters
		[DCB=([dsname][,attributes])] / [DCB=([*.stepname.ddname] [,attributes])]	
		[LABEL=(label information)]	See Chart 3 for subparameters
		[DISP=([status][,disposition])]	See Chart 3 for subparameters
		[SPACE=(direct-access space)] / [SPLIT=(direct-access space)] / [SUBALLOC=(direct-access space)]	See Chart 3 for subparameters
		[SEP=(ddnames)] / [AFF=ddname]	
		{* / DATA}	To define a data set in the input stream
		DUMMY,...	To bypass operations on a data set
		DDNAME=ddname	To postpone the definition of a data set
		SYSOUT=A,...	To route a data set through the output stream (sequential scheduler)
		SYSOUT=(x[,progname,form#] / [,progname] / [,,form#]),...	To route a data set through the output stream (priority scheduler)

Legend:

{} Choose one.
[] Optional; if more than one line is enclosed, choose one or none.

The DSNAME Parameter		The DISP Parameter	
DSNAME=	{name name(area name) name(membername) name(generation #) &name(membername) &name(area name) *.stepname.ddname}	DISP=	([SHR|NEW|OLD|MOD][,DELETE|,KEEP|,PASS|,CATLG|,UNCATLG])
The Unit Parameter			
UNIT=	([address|type|group][,P|,n|,][,DEFER][,SEP=(list of ddnames)])		
The VOLUME Parameter			
VOLUME=	([PRIVATE][,RETAIN|,][,volseq#|,][,volcount][,SER=(list of serial #s)|,REF=dsname|,REF=*.stepname.ddname])		
The LABEL Parameter			
LABEL=	([data set seq#][,SL|,SUL|,NSL|,NL|,BLP][,PASSWORD][,][EXPDT=yyddd|RETPD=nnnn])		
Space Allocation Parameters			
SPACE=	(ABSTR,(quantity,address[,directory]))		
SPACE=	({TRK|CYL|blocksize},(quantity[,increment][,directory|,index])[,RLSE][,CONTIG|,MXIG|,ALX|,][,ROUND])		
SPLIT=	{n (n,CYL,(quantity[,increment])) % (%,blocksize,(quantity[,increment]))}		
SUBALLOC=	({TRK|CYL|blocksize},(quantity[,increment][directory]),stepname.ddname)		

Legend:

{} Choose one.
[] Optional; if more than one line is enclcsed, choose one or none.

IBM Technical Newsletter

File Number S360-48

Re: Form No. C28-6539-4

This Newsletter No. N28-2214

Date February 27, 1967

Previous Newsletter Nos. None

IBM SYSTEM/360 OPERATING SYSTEM
JOB CONTROL LANGUAGE

This technical newsletter amends the publication, IBM System/360 Operating System: Job Control Language, Form C28-6539-4. Corrections and additions to the text are noted by vertical bars at the left of the change.

Pages to Be Inserted	Pages to Be Removed
19,20	19,20
29,30	29,30
47-50	47-50
53,54	53,54
57,58	57,58
67,68	67,68
73,74	73,74
81,82	81,82

Summary of Amendments

Documentation of the SYSUDUMP DD statement, bypass label processing, and the password protection feature. Additional unit types are included in Appendix A.

Note: Please file this cover letter at the back of the publication. Cover letters provide a quick reference to changes and a means of checking receipt of all amendments.

International Business Machines Corp., Product Publications Dept., Poughkeepsie, N. Y.

PRINTED IN U.S.A.

IBM Technical Newsletter

File Number	S360-48
Re: Form No.	C28-6539-4
This Newsletter No.	N28-2226
Date	April 10, 1967
Previous Newsletter Nos.	N28-2214

IBM SYSTEM/360 OPERATING SYSTEM
JOB CONTROL LANGUAGE

This technical newsletter amends the publication IBM System/360 Operating System: Job Control Language, Form C28-6539-4. The attached pages replace pages in the publication. Corrections and additions to the text are noted by vertical bars at the left of the affected text.

Pages to be Inserted	Pages to be Removed
13,14	13,14
19-22	19-22
25,26,26A	25,26
29-32	29-32
35,36	35,36
51,52	51,52
61-64	61-64
67,68	67,68

Summary of Amendments

This technical newsletter includes descriptions of:

- Storage volumes, and the resulting volume states.
- The effect of using the data set name NULLFILE.
- Treatment of channel and unit separation and affinity requests when the automatic volume recognition feature is used.
- New graphic units (Appendix A).

In addition, it clarifies or expands the descriptions of the MSGLEVEL parameter, the operand field of the DD statement, the SPLIT parameter, and the data set sequence number subparameter in the LABEL parameter.

Note: Please file this cover letter at the back of the publication. Cover letters provide a quick reference to changes, and a means of checking receipt of all amendments.

IBM Corporation, Programming Systems Publications, P.O. Box 390, Poughkeepsie, N.Y. 12602

PRINTED IN U.S.A.

READER'S COMMENTS

Title: IBM System/360 Operating System
Job Control Language

Form: C28-6539-4

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 _____

fold

Please check the items that describe your position:
___ Customer personnel ___ Operator ___ Sales Representative
___ IBM personnel ___ Programmer ___ Systems Engineer
___ Manager ___ Customer Engineer ___ Trainee
___ Systems Analyst ___ Instructor 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:

CUT ALONG LINE

fold

FOLD ON TWO LINES, STAPLE AND MAIL
No Postage Necessary if Mailed in U.S.A.

C28-6539-4

staple staple

fold fold

FIRST CLASS
PERMIT NO. 81

POUGHKEEPSIE, N.Y.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

POSTAGE WILL BE PAID BY

IBM CORPORATION
P.O. BOX 390
POUGHKEEPSIE, N. Y. 12602

ATTN: PROGRAMMING SYSTEMS PUBLICATIONS
DEPT. D58

fold fold

Printed in U.S.A. C28-6539-4

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]

C28-6539-4

Printed in U.S.A. C28-6539-4

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]