GC28-0617-2
File No. S370-36

Systems

OS/VS JCL Services

VS1 Release 2
VS2 Release 1

IBM

Preface

OS/VS JCL Services describes services provided by the operating system that an applications programmer can request by coding parameters of the job control language (JCL). This book is written primarily for applications programmers who code JCL statements for their jobs. It is divided into two parts:

The introduction describes the nine JCL statements and the organization of services used in this book. This publication does not include descriptions of all JCL services: a list of JCL services, noting the publication in which each is described and the parameter, subparameter, or statement used to request the service, is included in the introduction.

The descriptions of JCL services are grouped into five sections: Running Your Job; Defining and Describing Data Sets; Special Data Sets; Obtaining Output; Cataloged and In-stream Procedures. Each chapter describes when or why you would want to request the service discussed, and how to request or control the service.

The book assumes the reader has a basic knowledge of computer operating systems and some familiarity with job control language. Background information on VS1 and VS2 is included in the **IBM System Summary**, GA22-7001.

Parameters of the job control language are discussed only in the context of requesting services. Complete JCL parameter descriptions are included in the **OS/VS JCL Reference**, GC28-0618.

Publications to which the text refers:

OS/VS JCL Reference, GC28-0618

OS/VS Data Management Services Guide, GC26-3783

OS/VS Data Management Macro Instructions, GC26-3793

OS/VS Supervisor Services and Macro Instructions, GC27-6979

OS/VS Checkpoint/Restart, GC26-3784

OS/VS Utilities, GC35-0005

Introduction to Virtual Storage in System/370, GR20-4260

Third Edition (December, 1972)

This is a major revision of, and obsoletes, GC28-0617-1. See Summary of Amendments following the Contents. Changes or additions to the text and illustrations are indicated by a vertical line to the left of the change.

This edition applies to release 2 of OS/VS1 and release 1 of OS/VS2 and to all subsequent releases until otherwise indicated in new editions or Technical Newsletters. Changes are continually made to the information herein; before using this publication in connection with the operation of IBM systems, consult the latest **IBM System/360 and System/370 Bibliography**, Order No. GA22-6822, and the current **SRL Newsletter**, Order No. GN20-0360, for the editions that are applicable and current.

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

A form for reader's comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Programming Publications, Dept. G60, PO Box 6, Endicott, New York 13760. Comments become the property of IBM.

Contents

Figures

Summary of Amendments for GC28-0617-2 VS1 Release 2 VS2 Release 1

Changes Relating Only to the VS1 User:

I/O load balancing is an optional feature in VS1 release 2; if it is included in the system, unit separation requests are unnecessary, as described under "Unit Separation" in the chapter "Requesting Units and Volumes for Data Sets."

Remote entry services (RES) allow the VS1 user to submit jobs to the central computing center from a workstation and to route output to workstations. A new chapter "Controlling Output to a Workstation" has been added to the section "Obtaining Output."

A new parameter, HOLD, allows the VS1 user to delay the writing of an output data set until it is requested, as described under "Delaying the Writing of an Output Data Set" in the chapter "Writing Output Data Sets."

By assigning a job to an installation-prescribed job class, the VS1 user can take advantage of time slicing and dynamic dispatching, as described under "Dynamic Dispatching and Time Slicing" in the chapter "Job Scheduling."

Unit affinity cannot be requested for new data sets, as noted under "Sharing a Unit Between Data Sets" in the chapter "Requesting Units and Volumes for Data Sets."

A warning about the printing of return codes is included in the chapter "Conditional Execution of Job Steps."

Changes Relating to Both the VS1 and VS2 User:

Incorrect information was included in the chapter "Using a Dedicated Data Set." This chapter has been rewritten and retitled "Using a Dedicated Data Set for Allocating a Temporary Data Set."

The chapter "Requesting Space for a Data Set" has been rewritten to improve clarity. The information is now included in two chapters, "Requesting Space for a Single Data Set" and "Requesting Space for a Group of Data Sets."

Summary of Amendments for GC28-0617-1 VS1 Release 1 VS2 Release 1

OS/VS JCL Services, GC28-0617-1, replaces **OS/VS1 Job Management** Services, GC28-0617-0. The overview of job management (Part I) in **Job Management Services** is replaced by an introduction to JCL and JCL services in this book. The services themselves have been reorganized, and additional services are included. In general, **OS/VS JCL Services** applies to both VS1 and VS2; certain information that pertains to only one system is marked as such in the text.

The following chapters or additions to chapters apply to VS1, as well as VS2:

- "Defining the Checkpoint Data Set" in the chapter "Restarting a Job"
- "Requesting Space for a Data Set"
- "Creating and Using Private and Temporary Libraries"
- "Using a Dedicated Data Set"
- "Requesting an Abnormal Termination Dump" in the chapter "Controlling the Output Listing of JCL Statements, Messages, and Dumps"
- "Requesting a Specific Image" in the chapter "Printer Forms and Print Chain Control"
- "Writing Cataloged and In-stream Procedures"
- "Using Symbolic Parameters"

Two technical changes for VS1 update incorrect information that was included in **OS/VS1 Job Management Services:**

- you need not code MSGLEVEL=1 when automatic restart is to be performed;
- you can request a special character set for a 1403 printer by coding the UCS and UNIT parameters.

Introduction

You can write programs in any one of a number of languages. The operating system will translate the language into machine language so that your instructions can be executed and your work performed. However, in addition to coding programs to actually perform work, you must submit your programs to the operating system: a collection of related problem programs is submitted as a **job**; you define a job with the **job control language (JCL).**

A job can consist of one or more **job steps;** each job step is a unit of work associated with one processing program or one cataloged procedure and related data. (A cataloged procedure is a set of job control statements that has been placed in a partitioned data set called the procedure library; you can retrieve a cataloged procedure by coding its name on an EXEC statement.) You identify each job step with an EXEC statement; each data set used by a job step, with a DD statement; and the job itself, with a JOB statement. These three job control statements and six additional statements are summarized under "The JCL Statements."

In addition to identifying data sets, job steps, and the job, you can code parameters on JCL statements to request resources and services from the operating system. The operating system is responsible for managing all the resources of the computing system and will automatically perform many services in processing your job; however, you can influence the processing of your job by coding JCL parameters. For example, the operating system selects your job for execution, but you can influence when your job is selected, or you can delay its selection, by coding parameters on the JOB statement. You can ask for resources --for example, you can request a specific volume on which you want a data set written. A list of services provided by coding JCL parameters and an outline of the organization of services used in this book are included under "Introducing the JCL Services."

The JCL Statements

The job control language consists of nine statements. The name and purpose of each statement is summarized in Figure 1.

Basically, each job requires only the use of the JOB statement (to identify the job), EXEC statements (to identify each step), and DD statements (to identify data sets used by the job). The null statement is optional, but its use at the end of a job insures that JCL statements from another job will not be read as part of your job; the delimiter statement can be used to indicate the end of data in the input stream. You code PROC and PEND statements when you write an in-stream procedure. (An in-stream procedure is a set of job control statements placed in the input stream that can be used any number of times during a job by naming the procedure on an EXEC statement.) The use of these statements in in-stream procedures, and the optional use of the PROC statement in cataloged procedures, is discussed in the chapter "Writing Cataloged and In-stream Procedures." The command statement provides the facility to submit operator commands through the input stream; this statement is used primarily by the operator. The comment statement is useful to make your programs readily understandable by other programmers and by yourself. Details on coding all these statements, and a fuller description of each statement, are included in the **OS/VS JCL Reference**, GC28-0618.

Name of Statement	Purpose
job (JOB)	marks the beginning of a job; assigns a name to the job
execute (EXEC)	marks the beginning of a job step; identifies the program to be executed or the cataloged or in-stream procedure to be called; assigns a name to the step
data definition (DD)	identifies a data set and describes its attributes
delimiter (/* or two characters designated by the user)	indicates the end of data placed in the input stream
null (//)	marks the end of a job
procedure (PROC)	for cataloged procedures, assigns default values to parameters defined in the procedure; for in-stream procedures, marks the beginning of the procedure
procedure end (PEND)	marks the end of an in-stream procedure
comment (//*)	contains comments
command	enters operator commands through the input stream

Figure 1. Job Control Statements

The parameters that can be coded on JCL statements give JCL its versatility. There are two types of parameters: **positional parameters** must appear in a specified order on a JCL statement; **keyword parameters** consist of a keyword followed by one or more values and must follow any positional parameters coded on the statement. Complete lists of all the possible positional and keyword parameters that can be coded on JCL statements are included in the **OS/VS JCL Reference**, GC28-0618. Syntax rules for coding the parameters are also described in the **OS/VS JCL Reference.**

Introducing the JCL Services

JCL services described in this publication are divided into five sections:

- running your job
- defining and describing data sets
- special data sets
- obtaining output
- cataloged and in-stream procedures

Each section is divided into chapters describing individual services: why you would want to request the service and how to request the service.

Not every service provided by JCL is included in this book. The following list is intended to acquaint you with the services available; the list notes where the service is described and what statement, parameters or subparameters you code to request the service. Individual services provided by coding subparameters of the DCB parameter are not included: the services you can request with DCB subparameters depend on what access method you are using. For lists of DCB subparameters that can be coded for each access method, see the **OS/VS JCL Reference**, GC28-0618; services provided by many DCB subparameters are described in greater detail in **OS/VS Data Management Services Guide**, GC26-3783. For example, the **JCL Reference** tells you the data set organizations you can request in the DSORG subparameter; the **Data Management Services Guide** describes the different data set organizations in detail.

The list is divided into six areas:

- running your job
- defining and describing data sets
- special data sets
- obtaining output
- cataloged and in-stream procedures
- TSO and TCAM services

List of JCL Services

RUNNING YOUR JOB

Service	Publication(s) where Described	JCL Statement, Parameter, or Subparameter Used
automatic priority group, using (VS2 only)	*JCL Services*, "Job Scheduling"	PRTY parameter on JOB statement; DPRTY parameter on EXEC statement
conditional execution of job steps	*JCL Services*, "Conditional Execution of Job Steps"	COND parameter on JOB or EXEC statement
delaying job initiation	*JCL Services*, "Job Scheduling"	TYPRUN = HOLD parameter on JOB statement
dispatching priority, assigning (VS2 only)	*JCL Services*, "Job Scheduling"	DPRTY parameter on EXEC statement
executing programs contained in libraries	*JCL Reference*, "The PGM Parameter"	PGM parameter on EXEC statement
job class, assigning	*JCL Services*, "Job Scheduling"	CLASS parameter on JOB statement
job priority, assigning	*JCL Services*, "Job Scheduling"	PRTY parameter on JOB statement
job scheduling	*JCL Services*, "Job Scheduling"	PRTY, CLASS and TYPRUN = HOLD parameters on JOB statement; DPRTY parameter on EXEC statement
limiting the amount of time a job uses the CPU	*JCL Reference*, "The TIME Parameter"	TIME parameter on JOB statement
limiting the amount of time a job step uses the CPU	*JCL Reference*, "The TIME Parameter"	TIME parameter on EXEC statement
passing accounting information to accounting routines	*JCL Reference*, "The ACCT Parameter"	ACCT parameter on EXEC statement
passing information to processing program	*JCL Reference*, "The PARM Parameter"	PARM parameter on EXEC statement
restarting a job	*Checkpoint/Restart*, "Use of the Restart Facilities", *JCL Services*, "Restarting a Job"	RD parameter on JOB or EXEC statement; RESTART parameter on JOB statement
scanning JCL for errors (VS1 only)	*JCL Reference*, "The TYPRUN Parameter"	TYPRUN = SCAN parameter on JOB statement
specifying accounting information	*JCL Reference*, "Accounting Information Parameter"	accounting information parameter on JOB statement
storage for execution of a program, requesting	*JCL Services*, "Requesting Storage for Execution of a Program"	REGION and ADDRSPC parameters on JOB or EXEC statement
time slicing facility, using	*JCL Services*, "Job Scheduling"	PRTY parameter on JOB statement; DPRTY parameter on EXEC statement

DEFINING AND DESCRIBING DATA SETS (Part 1 of 2)

Service	Publication(s) where Described	JCL Statement, Parameter, or Subparameter Used
assigning specific tracks on a direct access volume to a data set	*JCL Services*, "Requesting Space for a Single Data Set"	SPACE parameter on DD statement
bypassing disposition processing	*JCL Services*, "Defining a Dummy Data Set"	DUMMY or DSNAME = NULLFILE parameter on DD statement
cataloging a data set	*JCL Services*, "Disposition Processing of Data Sets"	CATLG subparameter of DISP parameter on DD statement
completing the data control block	*Data Management Services Guide* "The Data Control Block"; *JCL Reference*, "The DCB Parameter"	DCB parameter on DD statement
contiguous space for a data set, requesting	*JCL Services*, "Requesting Space for a Single Data Set"	CONTIG subparameter of SPACE parameter on DD statement
data set disposition, specifying	*JCL Services*, "Disposition Processing of Data Sets"	DISP parameter on DD statement
exclusive control of a data set, requesting	*JCL Services*, "Insuring Data Set Integrity"	DISP parameter on DD statement
identifying a data set to the system	*JCL Reference*, "Identifying a Data Set to the System"	DDNAME, DSNAME, and LABEL parameters on DD statement
including data in the input stream	*JCL Reference*, "The * Parameter", "The DATA Parameter", "The DLM Parameter"	*, DATA, and DLM parameters on DD statement
insuring data set integrity	*JCL Services*, "Insuring Data Set Integrity"	DISP parameter on DD statement
keeping a data set	*JCL Services*, "Disposition Processing of Data Sets"	KEEP subparameter of DISP parameter on DD statement
label type, specifying	*JCL Reference*, "The LABEL Parameter"	LABEL parameter on DD statement
multiple units, requesting	*JCL Services*, "Requesting Units and Volumes for a Data Set"	unit count or P subparameter of UNIT parameter on DD statement
multivolume data sets, creating and retrieving	*JCL Services*, "Requesting Units and Volumes for a Data Set"	unit count and volume sequence number subparameters of VOLUME parameter on DD statement
optimizing channel use (VS1 only)	*JCL Reference*, "The AFF Parameter"	AFF parameter on DD statement
passing a data set	*JCL Services*, "Disposition Processing of Data Sets"	PASS subparameter of DISP parameter on DD statement

DEFINING AND DESCRIBING DATA SETS (Part 2 of 2)

Service	Publication(s) where Described	JCL Statement, Parameter, or Subparameter Used
postponing definition of a data set	*JCL Reference*, "The DDNAME Parameter"	DDNAME parameter on DD statement
private volumes, using	*JCL Services*, "Requesting Units and Volumes for a Data Set"	PRIVATE and RETAIN subparameters of VOLUME parameter on DD statement
protecting a data set	*JCL Reference*, "The Label Parameter"	PASSWORD, NOPWREAD, RETPD, and EXPDT subparameters of LABEL parameter on DD statement
releasing unused space	*JCL Services*, "Requesting Space for a Single Data Set"	RLSE subparameter of SPACE parameter on DD statement
shared control of a data set, requesting	*JCL Services*, "Insuring Data Set Integrity"	SHR subparameter of DISP parameter on DD statement
sharing tracks or cylinders between data sets	*JCL Services*, "Requesting Space for a Group of Data Sets"	SPLIT parameter on DD statement
sharing units between data sets	*JCL Services*, "Requesting Units and Volumes for a Data Set"	AFF subparameter of UNIT parameter on DD statement
sharing volumes between data sets	*JCL Services*, "Requesting Units and Volumes for a Data Set"	SER or REF subparameter of VOLUME parameter on DD statement
space for directory or index, requesting	*JCL Services*, "Requesting Space for a Single Data Set"	SPACE or SUBALLOC parameter on DD statement
space for a group of data sets, requesting	*JCL Services*, "Requesting Space for a Group of Data Sets"	SPLIT or SUBALLOC parameter on DD statement
space for a single data set, requesting	*JCL Services*, "Requesting Space for a Single Data Set"	SPACE parameter on DD statement
suballocating data sets	*JCL Services*, "Requesting Space for a Group of Data Sets"	SUBALLOC parameter on DD statement
uncataloging a data set	*JCL Services*, "Disposition Processing of Data Sets"	UNCATLG subparameter of DISP parameter on DD statement
units, requesting	*JCL Services*, "Requesting Units and Volumes for a Data Set"	UNIT parameter on DD statement
unit separation, requesting (VS1 only)	*JCL Services*, "Requesting Units and Volumes for a Data Set"	SEP subparameter of UNIT parameter on DD statement
volumes, requesting	*JCL Services*, "Requesting Units and Volumes for a Data Set"	VOLUME parameter on DD statement
whole cylinders, requesting	*JCL Services*, "Requesting Space for a Single Data Set"	ROUND subparameter of SPACE parameter on DD statement

SPECIAL DATA SETS

Service	Publication(s) where Described	JCL Statement, Parameter, or Subparameter Used
checkpoint data set, defining	*Checkpoint/Restart,* "Use of the Checkpoint Facilities"; *JCL Services*, "Restarting a Job"	SYSCHK DD statement
concatenated data sets, defining	*JCL Reference,* "Programming Notes"	DD statements
dedicated data sets, using for allocating a temporary data set	*JCL Services*, "Using a Dedicated Data Set for Allocating a Temporary Data Set"	DD statement
dummy data set, defining	*JCL Services*, "Defining a Dummy Data Set"	DUMMY or DSNAME = NULLFILE parameter on DD statement
generation data groups, creating and using	*JCL Reference,* "Creating and Retrieving Generation Data Sets"	DD statement
indexed sequential data sets, creating and using	*Data Management Services Guide,* "Processing an Indexed Sequential Data Set"; *JCL Reference,* "Creating and Retrieving Indexed Sequential Data Sets"	DD statement
private libraries, creating and using	*JCL Services*, "Creating and Using Private and Temporary Libraries"	JOBLIB or STEPLIB DD statement
temporary libraries, creating and using	*JCL Services*, "Creating and Using Private and Temporary Libraries"	DD statement

OBTAINING OUTPUT (Part of 1 of 3)

Service	Publication(s) where Described	JCL Statement, Parameter, or Subparameter Used
abnormal termination dump, requesting	*JCL Services,* "Controlling the Ouptut Listing of JCL Statements, Messages, and Dumps"	SYSABEND or SYSUDUMP DD statement
alignment of forms, requesting	*JCL Services*, "Printer Forms and Print Chain Control"	ALIGN subparameter of FCB parameter on DD statement
assigning messages to an output class	*JCL Services,* "Controlling the Output Listing of JCL Statements, Messages, and Dumps"	MSGCLASS parameter on JOB statement
assigning an output data set to an output class	*JCL Services*, "Writing Output Data Sets"	SYSOUT parameter on DD statement

Obtaining Output (Part 2 of 3)

Service	Publication(s) where Described	JCL Statement, Parameter, or Subparameter Used
controlling the output listing of JCL statements, messages and dumps	*JCL Services*, "Controlling the Output Listing of JCL Statements, Messages, and Dumps"	MSGLEVEL and MSGCLASS parameters on JOB statement; SYSABEND or SYSUDUMP DD statement
listing of JCL statements and messages, requesting	*JCL Services*, "Controlling the Output Listing of JCL Statements, Messages and Dumps"	MSGLEVEL and MSGCLASS parameters on JOB statement
controlling output to a workstation (VS1 only)	*JCL Services*, "Controlling Output to a Workstation"	SYSOUT, DEST parameter on DD statement
delaying the writing of an output data set	*JCL Services*, "Writing Output Data Sets"	HOLD=YES parameter on DD statement
dump, requesting	*JCL Services*, "Controlling the Output Listing of JCL Statements, messages and Dumps"	SYSABEND or SYSUDUMP DD statement
fold option, requesting when you request a special character set	*JCL Services*, "Printer Forms and Print Chain Control"	FOLD subparameter of UCS parameter on DD statement
holding an output data set	*JCL Sergices*, "Writing Output Data Sets"	HOLD=YES parameter on DD statement
multiple copies of an output data set (VS1 only)	*JCL Services*, "Requesting Multiple Copies of an Output Data Set"	COPIES parameter on DD statement
operator verification of special character sets, requesting	*JCL Services*, "Printer Forms and Print Chain Control"	VERIFY subparameter of UCS parameter on DD statement
operator verification of a specific image on a 3211 printer, requesting	*JCL Services*, "Printer Forms and Print Chain Control"	VERIFY subparameter of FCB parameter on DD statement
printer forms and print chain control	*JCL Services*, "Printer Forms and Print Chain Control"	UCS, FCB, and SYSOUT parameters on DD statement
routing output to another destination	*JCL Services*, "Controlling Output to a Workstation"	DEST parameter on DD statement
special character set, requesting	*JCL Services*, "Printer Forms and Print Chain Control"	UCS parameter on DD statement
special output form, requesting	*JCL Services*, "Printer Forms and Print Chain Control"	SYSOUT parameter on DD statement
special image for a 3211 printer, requesting	*JCL Services*, "Printer Forms and Print Chain Control"	FCB parameter on DD statement

Obtaining Output (Part 3 of 3)

Service	Publication(s) where Described	JCL Statement, Parameter, or Subparameter Used
specifying an output device	*JCL Services*, "Writing Output Data Sets"	UNIT parameter on DD statement
suppressing the writing of an output data set	*JCL Services*, "Defining a Dummy Data Set"	DUMMY or DSNAME = NULLFILE parameter on DD statement
using an installation-written writer routine	*JCL Services*, "Writing Output Data Sets"	SYSOUT parameter on DD statement
writing output data sets	*JCL Services*, "Writing Output Data Sets"	SYSOUT or UNIT parameter on DD statement

CATALOGED AND IN-STREAM PROCEDURES

Service	Publication(s) where Described	JCL Statement, Parameter, or Subparameter Used
adding DD statements to a procedure	*JCL Services*, "Using Cataloged and In-stream Procedures"	DD statement
assigning values to symbolic parameters	*JCL Services*, "Using Symbolic Parameters"	PROC or EXEC statement
calling cataloged and in-stream procedures	*JCL Services*, "Using Cataloged and In-stream Procedures"	EXEC statement
modifying parameters on EXEC and DD statements in a procedure	*JCL Services*, "Using Cataloged and In-stream Procedures"	EXEC statement
nullifying symbolic parameters	*JCL Services*, "Using Symbolic Parameters"	EXEC or PROC statement
using cataloged and in-stream procedures	*JCL Services*, "Using Cataloged and In-stream Procedures"	EXEC, DD statements
using symbolic parameters	*JCL Services* "Using Symbolic Parameters"	PROC, EXEC, DD statements
writing cataloged and in-stream procedures	*JCL Services*, "Writing Cataloged and In-stream Procedures"	PROC, PEND, EXEC, DD statements

TSO AND TCAM SERVICES

Service	Publication(s) where Described	JCL Statement, Parameter, or Subparameter Used
accessing messages received from a terminal via TCAM	*JCL Reference*, "The QNAME Parameter"	QNAME parameter on DD statement
indicating that a data set is going to or coming from a terminal	*JCL Reference*, "The TERM Parameter"	TERM parameter on DD statement
requesting notification when background job is complete	*JCL Reference*, "The NOTIFY Parameter"	NOTIFY parameter on DD statement

Descriptions of JCL Services

Running Your Job

The operating system is responsible for reading your job into the system, interpreting your JCL statements to determine the requirements of your job and job steps, satisfying those requirements, scheduling your job, and selecting it for execution. The system will automatically perform most of these services for you, but you can code JCL parameters to influence how these services are performed and to request resources your job requires for its execution. For example, the system will schedule your job for execution, but you can influence when your job is selected by coding the CLASS and PRTY parameters on the JOB statement.

The section is divided into four chapters:

- Job Scheduling
- Requesting Storage for Execution of a Program
- Conditional Execution of Job Steps
- Restarting a Job

When jobs are read into the system, they are placed in an input queue: the input queue is divided into job class queues and, within each job class queue, jobs are placed according to their priority. Jobs in the same class with the same priority are placed in the input queue in the order they were read into the system. An initiator selects jobs from the input queue. (An initiator can be thought of as a guide assigned to lead jobs of specified classes through the system. There can be only as many jobs active in the system concurrently as there are initiators started by the operator.) An initiator is assigned job classes to process: it selects jobs from the first class assigned to it according to the priority of the jobs until no more jobs exist in that class, and then selects jobs from the next class assigned. You influence how your job is placed in the input queue --therefore, when your job is selected for execution-- by assigning a job class and priority to your job. **In VS2**, you can also assign a dispatching priority to job steps in your job; the dispatching priority determines in what order job step tasks in your job will use real storage and CPU resources.

Although you can influence your job's selection by assigning a job class and priority to your job, you cannot predict whether a job in one job class queue will be selected for execution before another job in a different job class queue. When jobs exist in the same job class queue, you cannot be certain that one job will complete execution before the other job is selected, even if you assign a higher priority to the first job. In some cases, you might submit two jobs, for example, JOBA and JOBB, where JOBA must complete execution before JOBB is initiated --JOBA might create records that JOBB will use. You will have to delay JOBB's initiation until JOBA completes execution. A strong possibility can exist that resources a job requires will not be available --in this case, you can delay the job's initiation until required resources are available. You delay a job's initiation by coding TYPRUN=HOLD on the JOB statement.

Assigning a Job to a Job Class

Job classes are established by an installation to group jobs with similar characteristics. By assigning jobs to job classes, the installation tries to avoid contention between jobs that require the same resources by preventing them from running concurrently. For example, the installation might assign jobs that run for less than one minute to class C and jobs that have high I/O requirements to class D. When a job's characteristics could place it in one of several job classes (**i.e.**, a job might run for less than one minute **and** have high I/O requirements), you must determine which characteristic is most important in achieving a good balance of jobs in the computing system. The job class itself is a letter from A through O; its meaning is defined by the individual installation.

You assign your job to a job class by coding the CLASS parameter on the JOB statement:

```
//PGM   JOB ...CLASS=C
```

If you do not code the CLASS parameter, the reader assigns a default class of A to the job. If you code an incorrect job class (a letter other than A through O), the job is abnormally terminated. If you code an inactive job class -- a class that has not been assigned to an initiator -the job is placed in the input queue but is not selected until an initiator is started to process that job class.

Dynamic Dispatching and Time Slicing (VS1 only)

VS1 offers two options to provide more efficient use of the CPU: dynamic dispatching and time slicing. During system generation, your installation can specify time slicing or dynamic dispatching for a group of contiguous partitions. Both options can be included in the same system, but cannot be specified for the same partitions.

With time slicing, each task in each partition specified for time slicing is assigned an equal interval of time to retain control of the CPU; therefore, no task in the group can monopolize the CPU.

With dynamic dispatching, tasks in the group are considered either I/O-bound (relying heavily on input/output operations) or CPU-bound (making heavy demands on real storage). Each task is assigned an equal time interval to retain control of the CPU: if a task does not use its entire interval of allotted time, it is considered I/O-bound; if it does, it is considered CPU-bound. I/O-bound tasks are given higher dispatching priority within the group so that system through-put can be increased. (Dispatching priority is a number assigned to tasks, used to determine the order in which they will use the CPU.)

Your installation assigns job classes to each partition; different specific job classes should be assigned to partitions that include dynamic dispatching and to partitions that include time slicing. To take advantage of dynamic dispatching or time slicing, assign your job to the appropriate job class established by your installation.

Assigning a Priority to Your Job

Within a job class, jobs are selected for execution from the input queue according to job priority. Jobs with the same class and priority are placed in the input queue in a first in/first out order.

You assign a priority to your job in the PRTY parameter on the JOB statement:

```
//PGM   JOB ...CLASS=C,PRTY=10
```

The priority is a number from 0 to 13; 13 is the highest priority. In the above example, an initiator assigned to process job class C will begin execution of PGM after all jobs in class C with a higher priority and before all jobs in class C with a lower priority. If other jobs in class C also have a priority of 10, the initiator will choose the job that was read into the system first.

If you do not code a priority, the system will assign the default established in the reader procedure.

Note for VS1: The system programmer can establish a number less than 13 as the highest priority for jobs submitted from a work station. If you are submitting a job from a work station and code a priority greater than the established limit, that priority will be replaced by the default.

Assigning a Dispatching Priority to Job Steps (VS2 only)

Dispatching priority determines in what order tasks will use real storage and CPU resources. A **task** is a unit of work for the central processing unit from the standpoint of the control program; a task initiated by an initiator in accordance with specifications on an EXEC statement is known as a **job step task**. By coding a dispatching priority, you influence when job steps in your job will use real storage and CPU resources.

You do not have to code a dispatching priority --if you do not, job steps in your job are assigned the same dispatching priority as the job. In some cases, however, you may want a job step to have a higher or lower priority than the job. For example, in STEP3 of a job, you code COND=ONLY on the EXEC statement, indicating that the step should be executed **only** if a preceding step abnormally terminated. (The COND parameter is described in detail in the chapter "Conditional Execution of Job Steps.") You can assign this step a dispatching priority higher than the job's dispatching priority to get the job through the system quickly. In another job, one step makes heavy demands for real storage. You can assign this step a dispatching priority lower than the job's priority, so that the step does not monopolize CPU resources.

To assign a dispatching priority, code the DPRTY parameter on the EXEC statement. In the DPRTY parameter, you can code two values. The system substitutes these values in the following formula to form the dispatching priority:

```
(value1 x 16) + value2 = step's dispatching priority
```

Value1 has the same meaning as the value assigned in the PRTY parameter on the JOB statement. If you do not code value1, the system assumes a default value of 0. If you do not code value2, the system assumes a value of 11. If you omit the DPRTY parameter completely, the step has the same dispatching priority as the job; the job's dispatching priority is computed by substituting the job's priority in the following formula:

```
(priority x 16) + 11 = job's dispatching priority
```

You can code numbers between 0 and 15 for value1 and value2; a higher number indicates a higher dispatching priority. However, you should avoid assigning 15 to value1 --15 is used for certain system tasks. For example, if you code:

```
DPRTY=(10,5)
```

the dispatching priority is (10 x 16) + 5 = 165. If you code:

```
DPRTY=8
```

8 is substituted for value1 in the formula, and 11 is assumed for value2; the dispatching priority is (8 x 16) + 11 = 139. If you code:

```
DPRTY=(,12)
```

0 is assumed for value1 and 12 is substituted for value2; the dispatching priority is (0 x 16) + 12 = 12.

For example, you assign a priority of 8 to a job named ACCOUNT. STEP2 of the job makes heavy demands on real storage --you want to assign this step a lower dispatching priority than the job. STEP3 is executed only if a preceding step terminates abnormally --you want to assign a higher dispatching priority to STEP3:

```
//ACCOUNT       JOB        ...PRTY=8
//STEP1         EXEC       PGM=UPDATE
                 .
                 .
//STEP2         EXEC       PGM=ENHANCE,DPRTY=5
                 .
                 .
//STEP3         EXEC       PGM=ERROR,COND=ONLY,DPRTY=(10,13)
```

In STEP1, you did not code the DPRTY parameter: the job step has the same dispatching priority as the job.

Priorities and Time Slicing (VS2 only)

Time slicing is an option of VS2 that lets each task of a specified priority have control of the CPU for a specified interval of time. Normally a task maintains control until it is complete, until a higher-priority task becomes ready, or until it must wait for some event (such as an I/O operation). With time slicing, a group of tasks are allotted an interval of time which is divided among them --all tasks within the group are given an equal slice of CPU time and no task within the group can monopolize the CPU.

Your installation provides for time slicing during system generation, at which time it specifies the priorities that will be time sliced. To include a job in a time sliced group, code the priority specified at system generation as your job's priority in the PRTY parameter on the JOB statement. To include a job step in a time sliced group, code the priority specified at system generation as the first value (value1) in the DPRTY parameter on the EXEC statement; the second value (value2) must either be omitted or assigned the value 11.

Priorities and Automatic Priority Group (APG) (VS2 only)

Automatic priority group (APG) is an option of VS2 that assigns a single priority to a group of tasks in an attempt to provide optimum use of CPU and I/O resources by these tasks. APG is a method of achieving a good mix of CPU-bound tasks (tasks that make heavy demands for real storage) and I/O-bound tasks (tasks that rely heavily on input/output operations). Each task in the APG is dispatched with a time interval --if a task uses its entire interval of allotted time (i.e., does not relinquish control of the CPU voluntarily), it is considered CPU-bound; tasks that do not use their entire interval of allotted time are considered I/O-bound. I/O-bound tasks are ordered so that those which use smaller portions of their time interval are ranked high among the tasks waiting for the CPU. CPU-bound tasks receive control in a cyclic manner, ensuring that any available CPU time is distributed equitably among them.

Your installation specifies the priority associated with APG during system generation or system initialization. To include your job in the APG, code this priority as your job's priority in the PRTY parameter on the JOB statement. To include a job step in the APG, code the priority associated with the APG as the first value (value1) in the DPRTY parameter; the second value (value2) in the DPRTY parameter must either be ommitted or assigned a value of 11.

Delaying Job Initiation

To delay a job's initiation, code TYPRUN=HOLD on the JOB statement. The job is removed from the input queue and placed on a hold queue until the operator issues a RELEASE command for the job. (A hold queue is simply a waiting list for jobs whose initiation is being delayed.) You must notify the operator when you delay a job's initiation --no message is issued when a job is read into the system, and if the operator does not check, he will not know that a job has been placed on the hold queue. When the operator releases the job, it is again placed in an input queue, according to class and priority, and is eligible for execution.

For example, you are submitting two jobs, JOBA and JOBB. JOBB requires a data set created by JOBA. You can delay the initiation of JOBB until JOBA completes execution:

```
//JOBB      JOB      ...TYPRUN=HOLD
```

Include a note with your job to tell the operator that JOBB is being placed on the hold queue and should be released when JOBA completes execution. The operator issues a DISPLAY command to learn if JOBA has completed execution. When JOBA is complete, he will issue the RELEASE command for JOBB.

Requesting Storage for Execution of a Program

In VS, storage available for execution of your programs is divided into **real storage** and **external page storage:**

- real storage is the storage of System/370 from which the central processing unit can directly obtain instructions and data and to which it can directly return results.
- external page storage is auxiliary storage that contains programs in the form of fixed-length blocks called pages.

When a program is selected, it is brought into external page storage and divided into pages. The supervisor is responsible for transferring pages of your program from external page storage to real storage for execution. This paging is done automatically by the supervisor; to you, it appears as if your entire program exists in real storage. (The concept of paging is described in greater detail in **Introduction to Virtual Storage in System/370,** GR20-4260.)

Real storage and external page storage are the physical counterparts of virtual storage: virtual storage is addressable space that appears to the user as real storage, from which instructions and data are mapped into real storage locations. The size of virtual storage is limited by the addressing scheme of the computing system and by the amount of auxiliary storage available, rather than by the actual number of real storage locations.

When to Request Real Storage for a Program

For most programs, the supervisor transfers pages of your program to real storage as they are required for execution; not all pages of your program are necessarily in real storage at one time and the pages that are in real storage at once do not necessarily occupy contiguous space. Certain programs, however, must have all their pages in contiguous real storage while they are executing --they cannot be paged **during** execution. These programs include:

- programs that modify a channel program while it is active, such as the Online Test Executive Program (OLTEP) in VS2;
- programs that are highly time-dependent, such as the Magnetic Ink Character Recognition (MICR) programs.

These programs must be placed into an area of virtual storage called the **nonpageable dynamic area,** whose virtual addresses are identical to real addresses; they are the only programs for which you should request **real** storage. You request real storage with the REGION parameter; the meaning of the REGION parameter differs in VS1 and VS2 --for details see "How to Request Storage with the REGION parameter" for the system you are using.

In both VS1 and VS2, you identify programs that must not be paged during execution by coding the ADDRSPC (address space) parameter on the JOB or EXEC statement and specifying the subparameter REAL. When coded on the JOB statement, you are specifying that each step of the job must not be paged during its execution. To specify this requirement for a specific step, code ADDRSPC=REAL on the EXEC statement for that step. If you code the ADDRSPC parameter on the JOB statement, the parameter is ignored on EXEC statements in the job. The default assumed if you do not code the ADDRSPC parameter is ADDRSPC=VIRT, indicating that the program can be paged during its execution. Therefore you must code ADDRSPC=REAL if your program must be brought into contiguous real storage for its execution.

How to Request Storage with the REGION Parameter (VS1 only)

In OS/VS1, you code the REGION parameter only for programs that must not be paged during their execution -- the REGION parameter is ignored unless you also code ADDRSPC=REAL. If you code ADDRSPC=REAL but do not code the REGION parameter, the system supplies a default established in the reader procedure.

In the REGION parameter, specify the number of contiguous 1024-byte areas of real storage required. If you request an odd number, the system increases the number to the next highest even number.

The amount you specify must include any additional requests your program makes during its execution (for example, a request made with the GETMAIN macro instruction). Any request for additional storage is actually a request for real storage from the area specified in the REGION parameter. The size of your request has no relationship to the size of the virtual storage partition, since the job does not execute in virtual storage; the maximum size you request, therefore, depends on the physical size of the nonpageable dynamic area and the extent of system activity at the time the request is made.

Note: Main storage hierarchy support and the rollout/rollin feature are not supported in VS. However, you need not recode statements that contain the ROLL parameter or that contain REGION specifications originally made for hierarchy support. The system will check the ROLL parameter for correct syntax, but will otherwise ignore it. In the REGION parameter, the system will round the values specified to even numbers, if the numbers were odd, and add the values. For example, if a REGION parameter originally specified for hierarchy support is:

```
//PGM    JOB ...REGION=(11K,18K)
```

the system will round 11K to 12K, add the values, and assign 30K of real storage to each step of the job named PGM.

The REGION parameter can be coded on either the JOB or EXEC statement. When you code the REGION parameter on the JOB statement, you are requesting that much storage for each step of the job; to specify a different region size for each step, code the REGION parameter on the EXEC statements of the job steps in the job. (If the REGION parameter is coded on the JOB statement, REGION parameters coded on the job's EXEC statements are ignored.)

In the following example, you are specifying that the entire job must be in real storage during its execution and are requesting that the system assign 60 contiguous 1024-byte areas of real storage to each step.

```
//PGM    JOB      ...ADDRSPC=REAL,REGION=60K
```

How to Request Storage with the REGION Parameter (VS2 only)

You can code the REGION parameter for all your programs in VS2. The meaning of the REGION parameter differs, however, depending on if the program cannot be paged during its execution (you code ADDRSPC=REAL) or if pages of the program can be transferred to real storage as required for execution (ADDRSPC=VIRT is coded or implied).

When you code ADDRSPC=REAL and the REGION parameter, the amount of space you request is allocated to your program from the nonpageable dynamic area --your request is actually a request for real storage. Specify the required number of contiguous 1024-byte areas of real storage in the REGION parameter. The amount of real storage you request cannot exceed the size of the nonpageable dynamic area; your installation determines the size of the nonpageable dynamic area during system generation and therefore limits the amount of real storage that can be allocated to your program. If the number you specify is not an even multiple of 4K, the system will round your request up to a multiple of 4K (4K is the size of a page in VS2). For example, if you code:

```
//PROGRAM        JOB      ...ADDRSPC=REAL,REGION=22K
```

the system will round up your request to 24K and assign 24 contiguous 1024-byte areas of real storage to your program in the nonpageable dynamic area.

When you code the REGION parameter for programs that can be paged during their execution, the system allocates the space you request in external page storage. The supervisor obtains real storage for pages of your program as required for execution --your request in the REGION parameter is **not** a request for real storage. Specify the number of contiguous 1024-byte areas of virtual storage your program requires. If the number is not an even multiple of 64K, the system will round your request up to a multiple of 64K. (64K is the size of a **segment** in VS2, a continuous area of virtual storage which is allocated to a job or system task.) For example, if you code:

```
//DOTHIS         JOB      ...REGION=128K
```

the system assumes ADDRSPC=VIRT and assigns 128 contiguous 1024-byte areas of virtual storage to your program.

Whether you are requesting space in real storage (**i.e.**, you code ADDRSPC=REAL) or in virtual storage (**i.e.,**) you do not code the ADDRSPC parameter or code ADDRSPC=VIRT), the amount of space you request must include any additional requests your program makes during its execution (for example, a request made with the GETMAIN macro instruction). Any request for additional storage is actually a request for storage from the area specified in the REGION parameter.

Note: Main storage hierarchy support and the rollout/rollin feature are not available in VS. However, you need not recode statements that contain the ROLL parameter or that contain REGION specifications originally made for hierarchy support: the system will check the ROLL parameter for correct syntax, but will otherwise ignore it; in the REGION parameter, the system will add the values coded and use this sum as your request.

The REGION parameter can be coded on either the JOB or EXEC statement. When you code the REGION parameter on the JOB statement, you are requesting that much storage for each step of the job; to specify a different region size for each step, code the REGION parameter on the EXEC statements of the job steps in the job. If the REGION parameter is coded on the JOB statement, REGION parameters coded on the job's EXEC statements are ignored.

If you do not code the REGION parameter, a default region size established in the reader procedure is used.

Conditional Execution of Job Steps

Depending on the results of one step of a job, you may not wish to execute subsequent steps -- if a compilation fails, you would not want to waste computing time attempting subsequent linkage editing or execution steps. You can specify tests to determine whether to bypass or execute job steps, based on the results from previous steps.

The results of a job step can be reflected in a **return code,** a number from 0 to 4095. The compiler, assembler and linkage editor programs and problem programs written in assembler language, PL/I, FORTRAN, American National Standard COBOL and RPG can set return codes. Some return codes are standard for certain programs; for example, a return code of 8 issued by a compiler or linkage editor indicates that serious errors were found and execution is likely to fail. In problem programs you can assign a number as the return code to signify a certain condition. For example, if STEP1 of a job reads accounts to be processed in subsequent job steps, you might set a return code of 10 if no delinquent accounts are found. Before you execute STEP3 to process delinquent accounts, you could test the return code from STEP1; if the return code from STEP1 is 10 -- there are no delinquent accounts -- you can skip STEP3. You specify the test to check the return code from STEP1 by coding the COND parameter of the job control language. You can code the COND parameter on either a JOB or EXEC statement.

Note: In VS1, return codes issued by each step are included on the output listing. If a step does not issue a return code, however, the message is still printed and can contain meaningless information.

Specifying Return Code Tests

In the COND parameter, you can specify up to eight tests to determine if the system should bypass a job step. (If you specify more than eight tests, the system issues a JCL error message and the job is failed.) Each test consists of a number from 0 to 4095 and a logical operator indicating how that number is to be compared with the return code. The logical operators are:

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

If the system determines that a comparison is true, the job step is skipped (if COND was coded on the EXEC statement) or all remaining job steps are skipped (if COND was coded on the JOB statement).

For example, if you code COND=((10,GT),(20,LT)), you are asking, "Is 10 greater than the return code **or** is 20 less than the return code?"

If the return code is 12, neither test is satisfied; no job step is skipped. All the tests you specify must be false if processing is to continue without skipping any job steps.

If the return code is 25, the first test is still false, but the second test is satisfied: 20 is less than 25. The system will bypass one job step or all remaining job steps, depending on if the COND parameter was coded on the EXEC statement or the JOB statement.

Determining Further Execution of the Job

Code the COND parameter on the JOB statement to determine if execution of the job should continue.

The test you specify in the COND parameter on a JOB statement is used to check the return code from each step before the next step is processed. At the end of each step, the initiator compares the return code from the step with the number (or numbers) you specified in the COND parameter on the JOB statement. If any of the tests are satisfied, the rest of the steps are bypassed and the job is terminated.

For example, a program written in the assembler language issues return codes indicating the severity of errors found in the program: a return code of 0 indicates no errors or warnings were found; 4 indicates possible errors; and higher return codes indicate more severe errors. The job consists of a compiler step, linkage editor step, and execution step. If any errors are found, you want the job terminated. Since all steps issue the same return codes to indicate the same conditions, it is practical to code the COND parameter on the JOB statement:

```
//PGM        JOB      ...COND=(4,LE)
```

Determining the Execution of a Single Step

By coding COND on the EXEC statement, you can determine whether a single step will be executed or bypassed.

You should code COND on the EXEC statement when:

1. you want to specify different tests for each job step;
2. if a test you specify is true, you want to skip just that one step, rather than bypassing all subsequent steps in the job;
3. you want to name a specific step whose return code is to be tested;
4. you want to specify special conditions for executing a job step.

The initiator checks the COND parameter on the EXEC statement. If one of the tests you specify is satisfied, the system bypasses that step and goes on to the next step.

You can instruct the system to test the return code from a particular step or from every preceding step. Include the name of the step if you want just that step's return code tested; if that step was bypassed, the test is ignored. If you do not include a stepname, the system checks the return code from every preceding step.

STEP1 of a job prepares a company's payroll; STEP3 makes a monthly deduction for additional health insurance coverage. If the deduction is not to be made this week from any of the paychecks, STEP1 issues a return code of 15. On the EXEC statement for STEP3 you can instruct the system to skip STEP3 if no deduction is to be made:

```
//STEP3      EXEC        ...COND=(15,EQ,STEP1)
```

If a preceding step called a procedure, you can request the system to check the return code issued by a step in the procedure by coding the stepname and procedure stepname. In the above example, if the return code was issued by a procedure step named PROCSTEP in a procedure called by STEP1, you would code:

```
//STEP3      EXEC        ...COND=(15,EQ,STEP1.PROCSTEP)
```

caution: If a job step refers to a data set created in a preceding step, that data set will not exist if the preceding step was bypassed. If a data set was cataloged in a preceding job step and you make a backward reference to that data set, unit and volume information for the data set will not be available if the preceding step was skipped.

In addition to specifying conditions for bypassing a step, you can specify conditions for executing a step. Normally, all subsequent steps are bypassed if one step abnormally terminates. However, you can request the system to execute a step **even** if a previous step abnormally terminated by coding EVEN in the COND parameter:

```
//STEP3     EXEC        ...COND=EVEN
```

To instruct the system to execute a step **only** if a previous step abnormally terminated, code ONLY:

```
//STEP3     EXEC        ...COND=ONLY
```

If, however, the error causing termination occurs during the scheduling of the job, before the program receives control, the rest of the job steps are bypassed even if you do code EVEN or ONLY -- this will happen if the system encounters JCL errors or is unable to allocate space to a data set.

caution: If a job step that specifies the EVEN or ONLY subparameter refers to a data set that was to be created or cataloged in a preceding step, the data set may be incomplete if the step creating it abnormally terminated.

When you code the EVEN or ONLY subparameter, you can also specify up to seven tests to check return codes from previous steps. If one of the return code tests is satisfied, even though the conditions for the EVEN or ONLY subparameter are also satisfied, the step is bypassed. The return code tests override the EVEN or ONLY subparameter if the conditions both specify are met.

If you code

```
//STEP5     EXEC        ...COND=((10,EQ,STEP1),(20,LT),EVEN)
```

you are instructing the system to:

1. bypass STEP5 if 10 is equal to the return code issued by STEP1;
2. bypass STEP5 if any of the previous steps issued a return code greater than or equal to 20;
3. if the return code tests are not satisfied, execute STEP5 even if a previous step abnormally terminated.

Specifying Tests on Both the JOB and EXEC Statements

The COND parameter on the JOB statement overrides the COND parameter on an EXEC statement: if the test specified on the JOB statement is satisfied, all subsequent steps are bypassed no matter what you code in the COND parameter on the EXEC statements of these steps.

Restarting a Job

When a job step abnormally terminates, you may have to resubmit the job for execution; this means lost computer time and a delay in obtaining the desired results. The operating system provides checkpoint/restart facilities to reduce the effects of abnormal termination. When a job step terminates abnormally or when a system failure occurs, the checkpoint/restart facilities allow you to restart the step from the beginning or from a checkpoint within the step. You can request that the restart automatically follow abnormal termination or you can request restart later by submitting a new job.

This chapter describes how you code JCL to request checkpoint/restart services; a complete description of planning for and using the checkpoint/restart facility is documented in **OS/VS Checkpoint/Restart**, GC26-3784.

Types of Restart

Basically, there are two types of restart:

- **step restart**, from the beginning of a job step
- **checkpoint restart**, from a checkpoint within a job step. You establish checkpoints in a job step by coding the CHKPT macro instruction for each checkpoint. (The CHKPT macro instruction is described in **OS/VS Data Management Macro Instructions**, GC26-3793.)

You can request that either type of restart automatically follow abnormal termination (called automatic restart) or you can request either type by submitting a new job (called deferred restart).

When you submit a job for deferred restart, you actually resubmit the original job with certain changes indicating where restart is to occur (at the beginning of a step or at a checkpoint within the step). If necessary, you can make more extensive changes, such as corrections to data that will be processed after restart. At times, you may wish to make such changes and then restart a job step that has terminated normally but has produced incorrect results.

Automatic restart is possible only when the abnormal completion code is one of a set of codes specified at system generation. All automatic restarts must be authorized by the operator. If there is an uncorrectable error during the automatic step or checkpoint restart, the output data sets for the job are printed, providing you with the output from all of the steps up to and including the step that abnormally terminated. You will also receive a virtual storage dump if you provided a SYSABEND or SYSUDUMP DD statement in your job. (For details on requesting a dump, see the chapter "Controlling the Output Listing of JCL Statements, Messages, and Dumps.")

Requesting Restart

You specify the type of restart that can occur by coding the RD (restart definition) parameter on the JOB or EXEC statement. If you want to allow different types of restart for the different steps in your job, code the RD parameter on the EXEC statement; when the RD parameter is coded on the JOB statement, the restart request applies to every step in the job and any RD parameters coded on EXEC statements are ignored.

One of four possible subparameters can be coded:

- R -- Automatic step restart is permitted if no checkpoint is established in the step before abnormal termination occurs. If a checkpoint is established before the step abnormally terminates, only checkpoint restart can occur, unless you cancel the CHKPT macro instruction before restart is performed. If you do cancel the CHKPT macro instruction before restart is performed (by coding a CHKPT macro instruction and specifying CANCEL), automatic step restart can be performed.

- RNC -- Automatic step restart is allowed and automatic checkpoint restart is not allowed. Specifying RD=RNC suppresses the action of all CHKPT macro instructions included in your program.
- NC -- Neither automatic step restart nor automatic checkpoint restart is allowed. The action of all CHKPT macro instructions is suppressed.
- NR -- CHKPT macro instructions can establish checkpoints but automatic restart (whether checkpoint or step) is not allowed. Code RD=NR when you might want to resubmit the job at a later time and restart the job from a checkpoint.

For example, if you code:

```
//PGM   JOB ...RD=RNC,MSGLEVEL=(1,1)
```

automatic step restart is permitted; automatic checkpoint restart is not allowed.

When you resubmit a job to be restarted (deferred restart), you must code the RESTART parameter on the JOB statement. If you omit the RESTART parameter, execution of the job is not resumed at a point you indicate, execution of the entire job is repeated.

In the RESTART parameter, you specify the step at which execution should be resumed (deferred step restart) or the step **and** checkpoint (deferred checkpoint restart). When you specify a checkpoint, execution of the job is resumed within the step.

As the first subparameter, you specify the step at or within which execution will be resumed. If a step calls a cataloged procedure and you want to resume execution at or within a step in the procedure, specify both the job step name and procedure step name; for example:

```
//PGM   JOB ...RESTART=JOBSTEP2.PROCSTP3
```

Execution will begin at PROCSTP3 in the cataloged procedure called by JOBSTEP2. If you want execution to begin at or within the first step, you can code an *. For example:

```
//PGM   JOB ...RESTART=*
```

Execution will resume at the beginning of the first step; if the first step calls a cataloged procedure, execution will begin at the first step in the procedure.

To resume execution **within** a step, follow the stepname with the name of the checkpoint:

```
//PGM   JOB ...RESTART=(*,CHKPT2)
```

In this example, execution resumes at CHKPT2 in the first step of the job.

If you request deferred checkpoint restart, you must include a DD statement in the resubmitted job that defines the checkpoint data set.

Defining the Checkpoint Data Set

The name of the DD statement defining the checkpoint data set must be SYSCHK and the statement must immediately precede the first EXEC statement of the resubmitted job. (If you do include a SYSCHK DD statement, but restart is to begin at a step, the statement is ignored.)

The checkpoint data set contains entries describing the checkpoints you created in the job. (For information on creating checkpoints in your job, see **OS/VS Checkpoint/Restart**, GC26-3784.) The system automatically writes these entries into a checkpoint data set; the serial number of the volume on which a checkpoint is written is included in the console message printed after the writing of the checkpoint entry. You must indicate on the SYSCHK DD statement what volume the checkpoint entry you are using is found.

When the checkpoint data set is not cataloged, you code the VOLUME parameter and specify the volume serial number of the volume on which the checkpoint entry is written. If the checkpoint data set is cataloged, you need not code the VOLUME parameter unless the checkpoint entry exists on a tape volume other than the first volume of the data set; then, you must code either a volume sequence number or the volume serial number. If you code a volume serial number,you must code the UNIT parameter.

In the DSNAME parameter, you code the name of the checkpoint data set; if the data set is partitioned, do not include a member name. The DISP parameter must specify or imply a status of OLD and disposition of KEEP. Code the LABEL parameter if the data set does not have standard labels; if the data set exists on 7-track magnetic tape with nonstandard or no labels, you must also code DCB=TRTCH=C. (The TRTCH subparameter of the DCB parameter specifies the recording technique for seven-track tape; for details, see "The DCB Parameter" in the **OS/VS JCL Reference**, GC28-0618; details on specifying label type are also included in the **OS/VS JCL Reference** under "The LABEL Parameter.")

For example, the checkpoint data set named CHKLIB is cataloged and the checkpoint entry you are using exists on the first volume of the data set:

```
//ALAS      JOB      RESTART=(*,CHKPT2)
//SYSCHK    DD       DSNAME=CHKLIB,DISP=OLD
//STEP1     EXEC
```

In the following example, the checkpoint data set named TRY.AGAIN is not cataloged and exists on 7-track magnetic tape with nonstandard labels; the checkpoint entry you are using to restart the step exists on the volume with serial number 438291:

```
//ALACK     JOB      RESTART=(STEP2,CHKPT4)
//SYSCHK    DD       DSNAME=TRY.AGAIN,DISP=OLD,UNIT=3400-2,
//                   VOL=SER=438291,LABEL=(,NSL),DCB=TRTCH=C
```

If the RESTART parameter on the JOB statement in the preceding example were RESTART=STEP2, deferred step restart would be performed and the SYSCHK DD statement would be ignored.

Modifying A Job Before Deferred Restart

You can make changes to your job before submitting it for deferred restart. For example, you might vary device and volume configurations, alter data, or request restart on an alternate system with the same configuration used originally.

Some changes, however, are required before restarting the step. You must check all backward references to steps that precede the restart step and eliminate all backward references used in the PGM and COND parameters on the EXEC statement and the SUBALLOC parameter and VOLUME=REF=reference on the DD statements. (A backward reference of VOLUME=REF=reference is allowed if the referenced statement includes the volume serial numbers in the SER subparameter of the VOLUME parameter.)

Other required changes depend on whether you are requesting deferred step restart or deferred checkpoint restart; they are described below.

Making Changes Before Deferred Step Restart

Modifications before performing deferred step restart may be required in two cases:

1. A data set was defined as NEW during the original execution. If it was created during the original execution, you must change the data set's status to OLD, define a new data set, or delete the data set before resubmitting the job.
2. A data set was passed and was to be received by the restart step or a step following the restart step. If the passed data set is not cataloged, you must supply, in the receiving step, volume serial numbers, device type, data set sequence number, and label type. (Label type cannot be retrieved from the catalog.)

To limit the number of modifications required before you resubmit the job, you can assign conditional dispositions during the original execution. (Data sets assigned a temporary name or no name can only be assigned a conditional disposition of DELETE.) If deferred step restart will be performed, conditional dispositions should be used:

- to delete all new data sets created by the restart step.
- to keep all old data sets used by the restart step, other than those passed to the step. (If a nontemporary data set is defined as DISP=(OLD,DELETE), it is very important that you assign a conditional disposition of KEEP.)
- to catalog all data sets passed from steps preceding the restart step to the restart step or to a step following the restart step.

Making Changes Before Deferred Checkpoint Restart

When performing deferred checkpoint restart, the system will automatically make some modifications for the restart step, using information contained in the checkpoint entry.

An internal representation of your statements is kept as control information within the system. Some of the control information for the restart step or steps following the restart step may have to be modified before execution can be resumed at a checkpoint. The following modifications for the restart step are automatically made by the system, using information contained in the checkpoint entry:

- The status of data sets used by the step is changed from NEW to OLD. (If a new data set was assigned a nonspecific volume and was not opened before the checkpoint was established, this change is not made.)
- If nonspecific volumes were requested for a data set used in the restart step, the assigned device type and volume serial numbers are made part of the control information.
- For a multivolume data set, the volume being processed when the checkpoint was established is mounted.

The only required modification that you must make to a control statement is to supply certain information about a data set that was being passed by a step preceding the restart step to a step following the restart step. You must supply, in the receiving step, volume serial numbers, device type, data set sequence number, and label type. You will not have to make these modifications if, during the original execution, you assigned a conditional disposition of CATLG to such data sets and used standard labels. If the data set is cataloged, the system can retrieve this information from the catalog. (Label type cannot be retrieved from the catalog.) You should also use conditional dispositions to keep all data sets used by the restart step. Data sets assigned a temporary name or no name can only be assigned a conditional disposition of DELETE. Therefore, if you plan a deferred checkpoint restart, you should not define your data sets as temporary. (For any nontemporary data set that may be deleted, it is very important that you assign a conditional disposition of KEEP.)

Before resubmitting the job for checkpoint restart, you can make other modifications to control statements associated with the restart step or steps following the restart step. The following items apply to the step in which restart is to occur:

- The DD statements in the restart step can be altered, but the statements must have the same names as used originally. You can also include additional DD statements.
- If a data set was open at the time a checkpoint was established and restart is to begin at that checkpoint, DD statements in the restart step can define the same data set. If there is no need to process a data set after restart, you can define the data set by coding the DUMMY parameter or DSNAME=NULLFILE on the DD statement provided that: (1) the basic sequential access method (BSAM) or the queued sequential access method (QSAM) was being used to process the data set when the checkpoint was established (2) the data set is not the checkpoint data set that is being used to restart the job step, and (3) the job step is not restarted from a checkpoint that was established in an end-of-volume exit routine for the data set. The name of the DD statement must be the same as the one used for the data set during the original execution of your program.
- If DUMMY is not specified, the DD statements must define the same data sets. Also, the data sets must not have been moved on the volume or onto another volume.
- If a data set was not open when the checkpoint was established and is not needed during restart, you can replace the parameters used to define the data set with the DUMMY parameter.
- You can alter the data in the restart step. If you omit the data, a delimiter statement is not required, unless the data was preceded by a DD DATA statement.

Defining and Describing Data Sets

You must define every data set your job uses or creates on a data definition (DD) statement. The **OS/VS JCL Reference,** GC28-0618, describes every parameter you can code on a DD statement and illustrates the parameters necessary to create, retrieve, and extend data sets. This section describes in greater detail how to request certain resources for a data set and how you can instruct the system to handle a data set. The five chapters are:

- Requesting Units and Volumes for Data Sets
- Requesting Space for a Single Data Set
- Requesting Space for a Group of Data Sets
- Disposition Processing of Data Sets
- Insuring Data Set Integrity

Requesting Units and Volumes for Data Sets

On the DD statement defining a data set, you indicate the device on which the data set can be found or will be written by specifying unit and volume information. Input/output devices are grouped according to type; a device type is a kind of device: direct access, magnetic tape, unit record, graphic. A **unit** is a particular device: a 2314 direct access device, a 1403 unit record device; a **volume** is a section of auxiliary storage that is serviced by a single read/write mechanism --for example, a reel of magnetic tape, a drum, or a disk pack.

Specifying Volume Information

Volumes exist on direct access and magnetic tape devices and must be mounted on devices before they can be used. To inform the system on which volume an existing data set can be found or a new data set will be created, you make a **specific** or **nonspecific** volume request.

Specific Volume Requests

A **specific volume request** informs the system of the volume serial number(s) of the volume(s) you require. You must make a specific volume request for an existing data set; when you are creating a data set, you can make either a specific or nonspecific volume request.

A volume request is specific when:

- the data set is passed from an earlier step or is cataloged. The system obtains the volume serial numbers from the passed data set queue or from the catalog; you need not code the UNIT and VOLUME parameters, unless you want to request a private volume, retain a private volume (the volume on which a passed data set resides is automatically retained), code a volume sequence number, or request additional volumes or units. Each of these options is further described in the following paragraphs.
- you specify the serial numbers in the SER subparameter of the VOLUME parameter, **i.e.**, VOL=SER=(948762,945231).
- you refer the system to an earlier specific volume request to copy the volume serial numbers by coding the name of a passed or cataloged data set or a previous DD statement in the REF subparameter of the VOLUME parameter. To refer the system to a passed or cataloged data set, you code VOL=REF=dsname. To refer to a DD statement in the same step, code VOL=REF=*.ddname; in a preceding step, VOL=REF=*.stepname.ddname; or in a procedure step that is in a procedure called by a preceding step, VOL=REF=*.stepname.procstepname.ddname.

Nonspecific Volume Requests

Nonspecific volume requests can be made only for new data sets. When you make a nonspecific volume request, you do not specify volume serial numbers; you need not code the VOLUME parameter unless you are requesting a private volume, want to retain the private volume you request, or request more than one volume.

Note: After volumes are assigned to your data sets, space for the data sets is allocated on those volumes. Data sets for which you made nonspecific volume requests are allocated space in the order their DD statements appear in the job; as a result, the order of the nonspecific volume requests in a job step can influence whether the data sets can be allocated the space they require. For example, if a data set requiring a small amount of space precedes a large data set, space for the small data set may be allocated on a volume with a great deal of free space; however, the space left on the volume after the small data set is allocated may be insufficient to satisfy the large request. In general, it is best to place nonspecific volume requests for data sets that require a great deal of space before other nonspecific volume requests in the job step.

Using Private Volumes

A private volume cannot be allocated to satisfy nonspecific volume requests. Therefore, if you request a private volume, you will be the only user using that volume, unless another job makes a specific volume request for that volume. To request a private volume, code PRIVATE as the first subparameter in the VOLUME parameter.

You can code PRIVATE with both specific and nonspecific volume requests. When making a specific volume request for a direct access volume, you must code PRIVATE if you want a private volume; tape volumes for which you make a specific volume request are automatically made private, so you need not code the PRIVATE subparameter. For example, you are making a specific volume request for a direct access volume and want the volume to be private:

```
VOL=(PRIVATE,SER=485267)
```

The system will automatically demount the volume at the end of the job step unless the volume is being used by another job, the data set is passed, you code RETAIN in the VOLUME parameter, or the volume is permanently resident or reserved (permanently resident volumes are volumes that cannot be physically demounted or that contain system data sets; reserved volumes are volumes that remain mounted until the operator issues an UNLOAD command). If you expect to use a data set for which you requested a private volume in a subsequent step, you can code RETAIN to ensure that the volume remains mounted:

```
VOL=(PRIVATE,RETAIN)
```

The volume will remain mounted until the end of the job. If the data set resides on more than one volume and the volumes are mounted in sequential order, only the last volume is retained.

You need not code RETAIN for a passed data set; the volume on which a passed data set resides automatically remains mounted.

Multivolume Data Sets

If you are creating or extending a data set that may require more than one volume, you should request in the **volume count subparameter** of the VOLUME parameter the maximum number of volumes that may be required. If you are defining an existing multivolume data set and would like to begin processing with other than the first volume, code the **volume sequence number subparameter.**

Requesting Multiple Volumes

You request multiple volumes in the volume count subparameter of the VOLUME parameter. The maximum number of volumes you can request is 255; since each volume must be mounted on a unit before it can be used, you must:

- request as many units as volumes so that each volume will be mounted on a device, or
- for direct access volumes, make sure the volumes are nonsharable. A nonsharable volume can be allocated to only one data set at a time and, therefore, can be demounted after its use by your job so that another volume can be mounted. When you make a specific volume request and request more volumes than units, the system automatically assigns the nonsharable attribute to the volumes. For a nonspecific request for direct access volumes, you must code PRIVATE in the VOLUME parameter. (The system automatically demounts

tape volumes so you do not have to code PRIVATE for tapes.) For example, if you are making a nonspecific volume request for a data set that will require three direct access volumes, you would code:

```
VOL=(PRIVATE,,,3)
```

Positioning Within a Multivolume Data Set

When you are reading or lengthening an existing multivolume data set, you can instruct the system to begin processing other than the first volume by coding the volume sequence number subparameter.

Usually you code a volume sequence number when you are defining an existing cataloged or passed data set; for example,

```
//STATE DD  ...VOL=(,3,REF=DATASET)
```

DATASET is a cataloged data set; the system obtains the volume serial numbers from the catalog and begins processing with the third volume.

If you specify volume serial numbers for an existing data set, the system starts with the volume corresponding to the volume sequence number. For example,

```
//THIS  DD  ...VOL=(,2,VOL=SER=550001,550002,550003)
```

The system begins processing with volume 550002. Volumes 550001 and 550003 are also allocated to the data set and will be mounted when required.

Sharing Volumes Between Data Sets

To conserve space and to use fewer volumes, you can request that data sets be assigned the same volume. Data sets on the same volume have **volume affinity.**

You can request volume affinity either implicitly or explicitly:

- by specifying the same volume serial numbers for the data sets in the SER subparameter of the VOLUME parameter.
- by using the REF subparameter of the VOLUME parameter to indicate that volumes identified in the catalog or on an earlier DD statement in the job are to be assigned to the data set being defined.

Specifying Unit Information

You provide the system with the information it needs to assign a device to a data set in the UNIT parameter. To indicate what unit or type of unit you want, code one of the following:

- the unit address
- the device type
- the group name

The **unit address** is a 3-character address made up of the channel, control unit, and unit number. For example, unit 180 indicates you want channel 1, control unit 8, and unit 0. Specifying a unit address, however, limits unit assignment: the system can assign only that specific unit and, if the unit already is being used, the job must be delayed or cancelled.

A **device type** corresponds to a particular set of features of input/output devices. When you code a device type, you allow the system to assign any available device of that device type. For example, if you want a 2314 disk storage facility, you code:

```
    UNIT=2314
```

The system assigns an available 2314.

Each installation can also define **group names** during system generation to signify a group of devices that may not all be of the same type. When you code a group name, you allow the system to assign any available device included in the group. For example, if the group named DISK includes all 2314 and 3330 disk storage facilities and you code UNIT=DISK, the system assigns an available 2314 or 3330 device.

If a group contains more than one device type (for example, SYSSQ may refer to all tape and direct access devices), you should not code the group name when defining an existing data set. The volume on which the data set resides may require a device different from the one assigned to it. For example, if the data set resides on a tape volume, it must be assigned to a tape device.

Requesting More than One Unit

To increase operating efficiency, you can request multiple units for a multivolume data set or for a data set that may require additional volumes. When each required volume is mounted on a separate device, execution of the job step is not interrupted to allow the operator to demount and mount volumes. You should always request multiple units when the data set may be extended to a new volume if:

- the data set resides on a permanently resident or reserved volume --permanently resident and reserved volumes cannot be demounted in order to mount a new volume.
- the data set shares cylinders with other data sets or is suballocated space. (Suballocated data sets and data sets that share cylinders are described in the chapter "Requesting Space for a Group of Data Sets.")

You request multiple units by:

- coding the unit count subparameter in the UNIT parameter, or
- requesting parallel mounting.

You can request as many as 59 units in the unit count subparameter; for example, if you want three 2314's allocated to your data set, code:

```
    UNIT=(2314,3)
```

You can request **parallel mounting** when you make a specific volume request. The system counts the number of volumes requested (by counting the volume serial numbers specified on the DD statement or counting the volume serial numbers in the catalog or passed data set queue) and assigns that number of devices. You code P in place of the unit count subparameter:

```
//AMPLE DD  DSNAME=ENUF,DISP=OLD,UNIT=(2314,P),VOL=SER=(40653,13262)
```

The system assigns two 2314's to the data set defined by AMPLE -- one for each volume requested in the VOLUME parameter.

Deferred Mounting of Volumes

If your job step includes a data set that might not be used, depending on conditions determined in the job step, you can request that the system not mount the volume containing the data set until the data set is opened. This saves time mounting the volume before the job step begins execution.

Code the DEFER subparameter:

```
UNIT=(2314,,DEFER)
```

The system will assign a 2314 to the data set but will not request that the volume be mounted until it is required.

The DEFER subparameter should not be coded on a DD statement that defines an indexed sequential data set or that defines a new data set to be written to a direct access device.

Unit Separation (VS1 only)

When you make nonspecific volume requests for data sets, the system chooses volumes to be assigned to the data sets. A feature of VS1, called **I/O load balancing**, controls the choice of volumes and devices so that I/O contention on each device is equalized. I/O load balancing monitors the activity to each device; in choosing a device, it considers such variables as the speed of the device and the number of I/O events to each device. Because I/O load balancing reduces contention for devices on a system-wide basis, there is no need to request unit separation for data sets by coding the SEP subparameter of the UNIT parameter: if SEP is coded, it is ignored.

Your installation, however, can exclude the I/O load balancing feature from the system during system generation. If I/O load balancing is excluded, requests for unit separation are valid: you can request that a data set not be assigned to the same device as other data sets.

To request unit separation, code the SEP parameter and list up to eight ddnames of DD statements that define data sets that should not share a device with the data set you are defining:

```
//NOSHARE   DD      ...UNIT=(2314,SEP=(DD1,DD2,DD3))
```

The data set defined by NOSHARE will be assigned to a device different from the devices assigned to DD1, DD2, and DD3. The DD statements you list (in this example, DD1, DD2, and DD3) must precede this statement and must be included in the same job step. If one of the listed DD statements defines a dummy data set, the system ignores the unit separation request for that data set.

Unit separation requests have meaning only for direct access devices. If the system cannot satisfy a request for unit separation, because of insufficient devices available, the request is ignored.

Note for VS2: I/O load balancing is a standard feature in VS2; the SEP subparameter of the UNIT parameter is ignored if coded.

Sharing a Unit Between Data Sets

To conserve the number of devices used in a job step, you can request that an existing data set be assigned to the same device or devices assigned to a data set defined earlier in the job step. When two or more data sets are assigned the same device, the data sets are said to have unit affinity. When the data sets reside on different volumes, unit affinity implies deferred mounting for one of the volumes, since both volumes cannot be mounted on the same device at the same time.

You request unit affinty by coding UNIT=AFF=ddname on a DD statement. The ddname is the name of an earlier DD statement in the same job step, and the system obtains unit information from this statement. The data set defined on the DD statement that requests unit affinity is assigned the same device or devices as the data set defined on the named DD statement. If the ddname refers to a DD statement that defines a dummy data set, the data set defined on the DD statement requesting unit affinity is assigned a dummy status.

Note for VS1: Unit affinity cannot be requested for a new, direct access data set; if you do request unit affinity for a new data set, your job will be abnormally terminated.

When You Do Not Have to Code the UNIT Parameter

In a few cases, the system can obtain unit information from sources other than the UNIT parameter. In these cases, you do not have to code the UNIT parameter:

- When the data set is cataloged. For cataloged data sets, the system obtains unit and volume information from the catalog. However, if VOLUME=SER=serial number is coded on a DD statement that defines a cataloged data set, the system does not look in the catalog. In this case, you must code the UNIT parameter. If the VOLUME parameter is not coded but you request a device in the UNIT parameter, the request for a particular type of device is ignored. (A request for additional devices is, however, honored.)
- When the data set is passed from a previous job step. For passed data sets, the system obtains unit and volume information from an internal table. However, if VOLUME=SER=serial number is coded on a DD statement that defines a passed data set, the system does not look in the internal table. In this case, you must code the UNIT parameter. If the VOLUME parameter is not coded but you request a device in the UNIT parameter, the request for a particular type of device is ignored. (A request for additional devices is, however, honored.)
- When the data set is to use the same volumes assigned to an earlier data set, i.e., VOLUME=REF=reference is coded. In this case, the system obtains unit and volume information from the earlier DD statement that specified the volume serial number or from the catalog. If you request a device in the UNIT parameter, the request for a particular type of device is ignored. (A request for additional devices is, however, honored.)
- When the data set is to share tracks, blocks, or cylinders with an earlier data set, i.e., SUBALLOC or SPLIT is coded. (The SUBALLOC and SPLIT parameters are described in the chapter, "Requesting Space for a Group of Data Sets.") In this case, the system obtains unit and volume information from the earlier DD statement that specifies the total amount of space required for all the data sets. If the VOLUME parameter is coded, it is ignored. If you request a device in the UNIT parameter, the request for a particular type of device is ignored. (A request for additional devices is, however, honored.)

In all of the cases listed above, you can code the UNIT parameter when you want additional devices assigned.

You must not code the UNIT parameter when defining a data set included in the input stream. If UNIT is coded on a DD * or DD DATA statement, the job is abnormally terminated.

Bypassing Allocation of Units and Volumes

When you define a data set as a dummy data set, allocation is bypassed: no units, volumes, or direct access space is allocated to the data set. To define a dummy data set, you code the DUMMY parameter or assign the data set name NULLFILE in the DSNAME parameter. For details, see the chapter "Defining a Dummy Data Set."

Requesting Space for a Single Data Set

You must request space for every data set you create on a direct access volume. To request space, code the SPACE parameter on the DD statement defining the data set. (You can also request space for a group of data sets on a single direct access volume by coding the SPLIT or SUBALLOC parameters --see the chapter "Requesting Space for a Group of Data Sets.") The SPACE parameter provides two ways to request space:

- tell the system how much space you want and let the system assign specific tracks
- tell the system the specific tracks on which you want the data set written.

Letting the system assign specific tracks is the easiest and most frequently-used method of requesting space; the other methods of requesting space are available to increase performance, by minimizing access time and therefore increasing the efficiency of input/output operations. However, in most applications, the increase in efficiency by using alternate methods of requesting space is negligible. Examples of the types of applications that benefit from assigning specific tracks (or by coding the SPLIT and SUBALLOC parameters, in the chapter "Requesting Space for a Group of Data Sets") are included in the detailed description of each method.

Letting the System Assign Specific Tracks

The easiest way to request space is to let the system assign specific tracks; you need specify only the unit of measurement to be used to compute the space requirement, and how many of the units of measurement your data set requires. In addition, this form of the SPACE parameter offers several options; you can request:

- a secondary quantity, to be used if the data set runs out of space
- space for a directory or index
- release of unused space
- contiguous space
- whole cylinders.

The required subparameters and each of the options are discussed in the following paragraphs.

The Basic Request: Unit of Measurement and Primary Quantity

When the system assigns specific tracks, you are required to specify only the **unit of measurement** the system should use to allocate space and how much space you need, called the **primary quantity**. As the unit of measurement, you can specify:

- the average block length of the data, for blocks
- TRK, for tracks
- CYL, for cylinders.

As the primary quantity, you code an integer, indicating how many blocks, tracks, or cylinders you require.

It is easiest to specify an average block length: the system will compute the least number of tracks required to contain the number of blocks you specify and will allocate that number. Specifying block length also maintains device independence: you can code a group name that includes different direct access devices in the UNIT parameter, or you can change the device type in the UNIT parameter without altering your space request. (When a group name includes both tape and direct access devices, the SPACE parameter is ignored if a tape volume is assigned to the data set.)

If the blocks have keys, you must also code the DCB subparameter KEYLEN on the DD statement and specify the key length, **i.e.**, DCB=KEYLEN=key length. For example, the average block length of your data is 1,024 bytes and you expect to write 75 blocks of data; each block is preceded by a key 8 bytes long. The simplest space request, then, is:

```
//REQUEST     DD     ...SPACE=(1024,(75)),DCB=KEYLEN=8
```

The system computes how many tracks you need, depending on what device you request in the UNIT parameter.

When you specify TRK or CYL, you must compute the number of tracks or cylinders required; you should consider such variables as the device type, track capacity, tracks per cylinder, cylinders per volume, data length (blocksize), key length, and device overhead. These variables, and examples of estimating space requirements for partitioned and indexed sequential data sets, are described in **OS/VS Data Management Services Guide**, GC26-3783, under "Data Set Disposition and Space Allocation." Figures illustrating direct access capacities and track capacities are also included in the **OS/VS JCL Reference**, GC28-0618.

Requesting Whole Cylinders

Cylinder allocation allows faster input/output of sequential data sets than does track allocation. When you request space in terms of average block length, you can request that the space allocated begin and end on cylinder boundaries: code ROUND as the last subparameter in the SPACE parameter. For example, extending the previous example of a data set requiring 75 blocks with an average block length of 1024, you would code:

```
//REQUEST     DD     ...SPACE=(1024,(75),,,ROUND)
```

The smallest number of whole cylinders needed to contain your request will be allocated.

How the System Satisfies Your Primary Request

Enough available space must exist on one volume to satisfy your primary request. If enough space is not available on a single volume, the system will abnormally terminate the step or search another volume, depending on the type of volume request you make. Figure 2 illustrates system action for determining if enough space is available to satisfy your primary request.

Type of Volume Request		System Action
Specific volume request (i.e., volume serial numbers are specified)	For a single volume	If sufficient space is not available, job step is abnormally terminated.
	For multiple volumes	Search volumes in order specified until (1) Finds volume with sufficient space. (Volumes with insufficient space for primary quantity will still be used to allocate secondary quantity, if necessary.) (2) Determines none of specified volumes contain sufficient space -- job step is abnormally terminated.
Nonspecific volume request (i.e., system chooses volume)		If space is not available on first volume chosen, system will choose another volume and continue search, causing volumes to be mounted if necessary, until: (1) Volume with sufficient space is found. (2) Determines that no volume with sufficient space is available -- job step is abnormally terminated.

Figure 2. System Action for Determining if Enough Space is Available to Satisfy Primary Quantity

The system attempts to allocate the primary quantity in contiguous tracks or cylinders. If contiguous space is not available, the system satisfies the request with up to five noncontiguous **extents** (i.e., blocks) of space. (If you specify user labels -- i.e., you code SUL in the LABEL parameter-- the system allocates up to four noncontiguous extents of space. The system allocates a track for user labels separate from the primary quantity; this one track is considered an extent, and, therefore, up to four additional extents are allocated to satisfy the primary quantity.) The system uses the limit of five extents for the primary quantity to maintain a level of performance for input/output operations: if a data set is too fragmented on a volume, the speed of input/output operations is proportionately reduced.

In some applications, high efficiency of input/output operations may be important --you can assure that contiguous space is allocated by coding the CONTIG subparameter. See "Requesting Contiguous Space" for details.

A Secondary Request for Space

In the primary quantity, you need not anticipate all future demands for space for a data set. You can code a **secondary** request for space, to be used only if the data set exceeds its allocated space. The secondary quantity will be allocated as often as necessary.

Code an integer following the primary quantity that indicates how many additional tracks, cylinders, or blocks should be allocated. If your request is in units of blocks, you must code the maximum block length of your data, in either the DCB macro instruction or the BLKSIZE subparameter of the DCB parameter on the DD statement: the system uses the maximum block length to compute how many additional tracks to allocate.

A secondary quantity can be requested when you create a data set or when you retrieve an existing data set, whether or not you coded a secondary quantity in the original request. A secondary request for an existing data set is in effect only for the duration of the job step and overrides an original request if one was made. For example, when you created a data set named DARTS, you did not code a secondary quantity. You are retrieving the data set in a later job to lengthen it and want to request 50 additional blocks of space:

```
//PUB       DD        DSN=DARTS,DISP=OLD,SPACE=(1024,(100,50)),
//                    DCB=BLKSIZE=2048
```

The secondary request for 50 blocks is in effect only for the duration of this step. The unit of measurement and primary quantity must be recoded exactly as they appeared in the original request.

How the System Satisfies Your Secondary Request

The system allocates the secondary quantity every time your data set has used its allocated space. The system will attempt to allocate additional space contiguous with the primary quantity; however, if contiguous space is not available, the system allocates up to five noncontiguous extents of space equalling the secondary quantity.

Secondary space need not be allocated on the same volume as the primary quantity. However, the system will allocate all requested space on the same volume until (1) it determines insufficient space is available; or (2) sixteen extents of space have been allocated to the data set on one volume. If either of these conditions occurs, the system will allocate space on another volume as long as you requested sufficient volumes in the VOLUME parameter --see "Multivolume Data Sets" in the chapter "Requesting Units and Volumes for Data Sets."

Requesting Space for a Directory or Index

If you are creating a partitioned data set, you must request space for a directory. A directory is an index used by the system to locate members in a partitioned data set. It consists of 256-byte records, and you must specify, in the SPACE parameter, how many records the directory is to contain. You should request enough space for a directory to allow for growth of the data set: you cannot lengthen the directory, as you can lengthen the data set itself by requesting a secondary quantity. If you run out of space in the directory, you must recreate the data set. For a complete description of the directory, including details on member entries that will enable you to compute how many records to request, see "Processing a Partitioned Data Set" in the **OS/VS Data Management Services Guide**, GC26-3783.

If you are creating an indexed sequential data set and are **not** defining the index on a separate DD statement, you must request space for an index if the data set occupies more than one cylinder: code an integer indicating how many cylinders should be allocated for the index. (The space request for an indexed sequential data set must be in terms of cylinders.)

The directory or index is allocated from the space you request as the primary quantity. Therefore, you must consider the size of your directory or index in estimating the primary space request. The system determines whether you are requesting space for a directory or an index by examining the DSORG subparameter of the DCB parameter on the DD statement. DCB=DSORG=IS or DCB=DSORG=ISU must be included on any DD statement defining an indexed sequential data set. If neither is specified, the system assumes you are requesting space for a directory.

For example, you are creating an indexed sequential data set and requesting 2 cylinders for an index:

```
//INDEXDS        DD       ...SPACE=(CYL,(10,,2)),DCB=DSORG=IS
```

The system recognizes that you are requesting space for an index because of the DCB subparameter DSORG=IS.

Requesting Contiguous Space

If the system cannot allocate the primary quantity in contiguous space, it will allocate up to five extents of noncontiguous space equalling the primary request, as described under "How the System Satisfies Your Primary Request." The efficiency of input/output operations is lessened when space for your data set is divided. However, in most applications, the effect is negligible; only when you are defining an empty data set for suballocation (see the chapter "Requesting Space for a Group of Data Sets") or certain system data sets (e.g., SYS1.PARMLIB) is contiguous space required.

Although contiguous space is not required for most data sets in applications programs, a high level of efficiency in input/output operations might be desired for some applications, notably in teleprocessing. To ensure that contiguous space is allocated, code the CONTIG subparameter:

```
//RESERVS        DD       DSN=FLIGHTS4,DISP=(NEW,KEEP),
//                        SPACE=(CYL,(50),,CONTIG),
//                        UNIT=2314,VOL=SER=436921
```

If contiguous space is not available, the job is abnormally terminated.

If you code a secondary quantity and request contiguous space, the primary request will be satisfied with contiguous space, but the secondary quantity will not necessarily be contiguous.

Releasing Unused Space

When a data set has been created and you do not expect to lengthen it, you can release unused space that was allocated to the data set. You should always do this if you made a large, safe request for primary space. Code RLSE in the SPACE parameter.

You can code RLSE when you create a data set or when you retrieve an existing data set --the unused space is released when the data set is closed. If you requested space in units of tracks, unused tracks are released; in units of cylinders, unused cylinders are released; in units of blocks, unused tracks or cylinders, whichever the system allocated, are released. When coding RLSE for an existing data set, you should recode the unit of measurement and primary quantity exactly as they appeared in the original request. For example, if your original request was:

```
SPACE=(TRK,(100,50))
```

You can release unused tracks by coding, when you retrieve the data set,

```
SPACE=(TRK,(100),RLSE)
```

Assigning Specific Tracks

You can request that specific tracks on a volume be allocated to your data set. This is the most stringent request for space: if any of the tracks you request are occupied, the space cannot be allocated and your job is abnormally terminated. Usually, you request specific tracks in order to place a frequently-used data set near the volume table of contents (VTOC) to minimize access arm movement and thereby increase the speed of I/O operations. A library that is heavily referenced might be a good candidate for placement near the VTOC. However, in most applications, requesting specific tracks is unnecessary.

To request specific tracks, you must code:

- ABSTR as the first subparameter, indicating absolute tracks
- a primary quantity, specifying the number of tracks to be allocated
- the relative track number of the first track to be allocated.

For a partitioned data set, you must also specify how many records you want allocated for a directory. If you are defining an indexed sequential data set and are **not** defining the index on a separate DD statement, you must request space for an index if the data set occupies more than one cylinder. The number of tracks for the index must be equal to one or more cylinders and any other DD statement defining the indexed sequential data set (**i.e.**, a separate DD statement defining an overflow area) must also request specific tracks. The space for the index or directory is allocated from the primary quantity.

To determine the relative track number, count the first track of the first cylinder on the volume as 0, and count through the tracks on each cylinder until you reach the track on which you want the data set to start. (You cannot request track 0.) The system automatically converts the relative track number to an address; this address varies with different devices. For indexed sequential data sets, the relative track number must correspond to the first track on a cylinder. Capacities of direct access devices are included in the **OS/VS JCL Reference**, GC28-0618.

For example, you are creating an indexed sequential data set named WEBSTER on volume 727104 on a 2314 direct access device. You need 4 cylinders for the primary quantity, which includes 1 cylinder for the index. WEBSTER is a heavily-used library and you want to place it near the VTOC. On volume 727104, the VTOC begins on the seventh cylinder; on a 2314 direct access device, 20 tracks equal 1 cylinder. To place WEBSTER directly before the VTOC --starting at the beginning of the third cylinder-- you would code:

```
//INDEXDS   DD        DSN=WEBSTER,DISP=( ,KEEP),UNIT=2314,VOL=SER=727104,
//                    SPACE=(ABSTR,(80,40,20)),DCB=DSORG=IS
```

80 is your primary quantity, equalling 4 cylinders; 40 is the relative track number of the first track on the third cylinder; 20 is your request for 1 cylinder for the index.

Requesting Space for a Group of Data Sets

You must request space for every data set you create on a direct access volume. In most applications, you request space for each data set by coding the SPACE parameter -- see the chapter "Requesting Space for a Single Data Set." But in some cases, to increase performance, you may want to place a group of data sets on a single volume in a certain order. JCL provides two methods for doing this; you can:

- share cylinders between two or more related data sets in a single job step; portions of each data set occupy tracks within every allocated cylinder. This method is useful when you are processing large data sets with corresponding records.
- suballocate space for each data set from an empty data set you define that contains enough space for all the data sets in the group; the data sets can be placed in contiguous space in a specific order.

More detailed examples of when to use each method are included in the detailed description of the method.

Sharing Cylinders Between Data Sets

Sharing cylinders between data sets is useful when you are creating two or more large data sets with corresponding records. For example, a college has one data set containing students' names, identification numbers, and addresses; a second data set contains each student's courses; a third data set lists all the courses, the enrollment in each course, and the grade earned by each student in each course. If these data sets share cylinders, considerable time can be saved when they are processed: each data set occupies a portion of the tracks in every allocated cylinder; movement of the access arm is decreased and processing time is therefore decreased. The decrease in time, however, is significant only for large data sets, and you should request a private volume: if other data sets are using the volume concurrently, the benefit is lost. (For details on requesting a private volume, see "Using Private Volumes" in the chapter "Requesting Units and Volumes for Data Sets.")

To share cylinders, you define the data sets by coding a sequence of DD statements using the SPLIT parameter to request space.

The Sequence of DD Statements

Each DD statement in the sequence defines one of the data sets in the group. On the first DD statement, you must:

- define the first data set in the group.
- request space for all the data sets in the group; if the system cannot allocate this space on a single volume, the job is abnormally terminated.
- indicate how many tracks per cylinder are to be allocated to this data set.
- code unit and volume information.

Optionally, you can code a secondary quantity, which is allocated to any data set in the group that runs out of space.

On subsequent DD statements in the sequence, the SPLIT parameter simply indicates how many tracks per cylinder are to be allocated to the data set. You need not code unit and volume information --the VOLUME parameter, if coded, is ignored. The UNIT parameter is also ignored, with the exception of a request for additional devices. (See "Requesting a Secondary Quantity" for details on when to request additional devices.)

You can request space for the data set either in terms of average block length or cylinders. The way you indicate the number of tracks per cylinder to be allocated to each data set depends on the unit of measurement you code.

Requesting Blocks of Space

To request blocks of space, specify the average block length of the data and how many blocks you expect to write for all the data sets combined. The system computes for you how many cylinders are required, depending on what device you request in the UNIT parameter.

To indicate how many tracks per cylinder are to be allocated to each data set, specify a percent of the total tracks on a cylinder. The system computes how many tracks the percent indicates and rounds down to the next full track; as a result, the percent you request must equal at least one full track or the step is abnormally terminated. For example, if you request 5% of the tracks on a cylinder on a 3330, you are requesting .95 tracks and the job step is abnormally terminated; if you request 10% of the tracks on a 3330, you are requesting 1.90 tracks and the system allocates one track per cylinder.

In the following example, the average block length of your data is 1,024 bytes and you need 800 blocks for three data sets. You want to allocate 20% of the tracks on each allocated cylinder to the first data set, named CAMEL; 35% to the data set named LLAMA; and 45% to the data set named OSTRICH. The DD statements would be:

```
//DD1    DD   DSN=CAMEL,DISP=(,KEEP),UNIT=2314,
//            VOL=SER=253540,SPLIT=(20,1024,(800))
//DD2    DD   DSN=LLAMA,DISP=(,KEEP),SPLIT=35
//DD3    DD   DSN=OSTRICH,DISP=(,KEEP),SPLIT=45
```

You choose the percent of tracks to be allocated to each data set so that, when you reach the end of a cylinder, you will have finished processing the portions of all three data sets on that cylinder. You must consider the size of each data set, the size of the records in each data set, and what type of operation you are performing, e.g., reading, writing, modifying records.

Requesting Cylinders

To request cylinders, you specify CYL as the unit of measurement and an integer indicating how many cylinders should be allocated for all the data sets in the group. When you specify CYL, you must compute the number of cylinders required, considering the device type, track capacity, tracks per cylinder, cylinders per volume, data length (blocksize), keylength, and device overhead. These variables are described in **OS/VS Data Management Services Guide**, GC26-3783. Figures illustrating direct access capacities and track capacities are also included in the **OS/VS JCL Reference**, GC28-0618.

To indicate how many tracks per cylinder should be allocated to each data set, you simply specify a number of tracks. For example, you are requesting 7 cylinders on a 2314 for three data sets named KING, QUEEN, and JACK. On a 2314, each cylinder contains 20 tracks: KING should occupy 8 tracks per cylinder; QUEEN, 6 tracks per cylinder; and JACK, 6 tracks per cylinder.

You would code:

```
//DDA    DD   DSN=KING,DISP=(,KEEP),UNIT=2314,
//            VOL=SER=123456,SPLIT=(8,CYL,(7))
//DDB    DD   DSN=QUEEN,DISP=(,KEEP),SPLIT=6
//DDC    DD   DSN=JACK,DISP=(,KEEP),SPLIT=6
```

You choose the number of tracks per cylinder to be allocated to each data set so that, when you reach the end of a cylinder, you will have finished processing the portions of all three data sets on that cylinder. You must consider the size of each data set, the size of the records in each data set, and what type of operation you are performing, e.g., reading, writing, modifying records.

Requesting a Secondary Quantity

You can specify a secondary quantity in the SPLIT parameter on the first DD statement in the sequence. The system will allocate additional blocks or cylinders (whichever you requested as the primary quantity) to any data set in the group that runs out of space. (If you requested blocks, you must code the maximum block length of the data in the BLKSIZE subparameter of the DCB parameter or in the DCB macro instruction. The system uses the maximum block size to determine how many additional tracks to allocate.) If you do not request a secondary quantity and a data set runs out of space, the job step is abnormally terminated.

Additional space is not split with the other data sets and can be allocated on another volume, if you requested multiple volumes in the VOLUME parameter. If the data set might be extended to another volume, you should also request an additional device --the volume containing the shared data sets need not be demounted, then, in order to mount the volume for the secondary quantity. You request multiple devices in the UNIT parameter --see "Requesting More than One Unit" in the chapter "Requesting Units and Volumes for Data Sets." The request for an additional device can be coded on any DD statement in the sequence.

For example, in a preceding example, you requested 800 blocks with an average block length of 1,024 bytes. The first data set required 20% of the tracks on each cylinder. To include a secondary request for 35 additional blocks, you could code:

```
//DD1    DD  DSN=CAMEL,DISP=( ,KEEP),UNIT=(2314,2),
//           VOL=(PRIVATE,,2),SPLIT=(20,1024,(800,35))
```

An additional unit and volume is requested in case the secondary quantity must be allocated on another volume.

Suballocating Space

The method of suballocating space is primarily used to reserve a block of space for a group of data sets. You first create a master data set in contiguous space on a single volume that contains enough space for all the data sets in the group, but that does not itself contain any data. Then you suballocate space from the master data set for data sets in the group. An installation might reserve blocks of space for different departments, or distinct applications, or to give programmers a certain amount of work space.

For example, a master data set reserves 8 cylinders of space on a 2314 for use by an accounting department, DEPT41. DEPT41 creates four data sets, suballocating space for each from the master data set. Each new data set is assigned to the first available area of unused space in the master data set, so that the data sets can be placed in a specific order (i.e., in the order in which they are defined). If DEPT41 is processing some of these data sets at the same time, processing time can be decreased by making the volume on which they reside private. (A private volume cannot be allocated to satisfy a nonspecific volume request; therefore, other data sets will not be allocated to a private volume unless they specifically request it by coding its volume serial number.) Access-arm movement is decreased, since all the data sets occupy a contiguous area on the volume and other data sets are not using the volume.

Defining the Master Data Set

You define the master data set by coding the SPACE parameter. The SPACE parameter offers two methods for requesting space: letting the system assign specific tracks and requesting specific tracks. When you let the system assign specific tracks, you code a subset of the available subparameters. You must code:

- the unit of measurement and primary quantity
- CONTIG to request contiguous space.

Optionally, you can code the ROUND subparameter to request whole cylinders when the unit of measurement is average block length. Both methods of requesting space are described in detail in the chapter "Requesting Space for a Single Data Set." The specific subparameters listed above are described under "The Basic Request: Unit of Measurement and Primary Quantity," "Requesting Contiguous Space", and "Requesting Whole Cylinders" in that chapter.

You must include unit and volume information on the DD statement defining the master data set. Sufficient contiguous space to satisfy the primary quantity must exist on a single volume or the job step is abnormally terminated.

For example, to define the master data set to be used by DEPT41, you could code:

```
//MASTER    DD  DSN=DEPT41,DISP=(,KEEP),UNIT=2314,
//              VOL=SER=123456,SPACE=(CYL,(8),,CONTIG)
```

Suballocating Space from the Master Data Set

To suballocate space from the master data set, you code the SUBALLOC parameter. The SUBALLOC parameter is very much like the SPACE parameter when you let the system assign specific tracks. You must specify a unit of measurement and a primary quantity --the unit of measurement need not be the same as you specified when defining the master data set. Optionally, you can request secondary space and space for a directory. For details on coding these requests, see "The Basic Request: Unit of Measurement and Primary Quantity", "A Secondary Request for Space," and "Requesting Space for a Directory or Index" in the chapter "Requesting Space for a Single Data Set." Details on how a secondary space request is satisfied are included below.

In the SUBALLOC parameter, you must also identify the master data set from which space is to be suballocated --see "Identifying the Master Data Set" in this chapter.

How a Secondary Space Request is Satisfied

Secondary space allocated to your data set is not allocated from the master data set and can be allocated on a separate volume, if you requested more than one volume when you defined the master data set. If the data set might be extended to another volume, you should also request an additional device, so that the volume containing the master data set need not be demounted. You can request an additional device in the UNIT parameter on either the DD statement defining the master data set or the DD statement defining the data set to be suballocated. (For details on how to request an additional device, see "Requesting More than One Unit" in the chapter "Requesting Units and Volumes for Data Sets.") With the exception of a request for an additional device, the UNIT and VOLUME parameters are ignored, if coded, on a DD statement that defines a suballocated data set.

Identifying the Master Data Set

In the SUBALLOC parameter, you must identify the master data set from which space is to be suballocated. You can suballocate space from an existing master data set --**i.e.**, you need not create the master data set in the same job as you create a data set to be suballocated. However, your job must include a DD statement defining the master data set. You refer to this DD statement in the SUBALLOC parameter by coding:

- ddname --if the DD statement defining the master data set appears in the same job step.
- stepname.ddname --if the DD statement appears in an earlier job step.
- stepname.procstepname.ddname --if the DD statement appears in a procedure step that is part of a cataloged or in-stream procedure called by an earlier job step.

Example of Suballocating Space for Data Sets

An accounting department, DEPT41, defines a master data set reserving 8 cylinders of space on a 2314. Three data sets are suballocated from this space in two different jobs:

- the first data set, named FIRST, is a partitioned data set requiring 3 cylinders: you must request space for a directory containing 10 records.
- the second data set, named SECOND, requires 50 tracks; you also want to request a secondary quantity of 25 tracks. If space for the secondary quantity is not available on the same volume as the master data set, you want the secondary quantity allocated on another volume: you must request multiple volumes when defining the master data set and request an additional device.
- in a later job, you define a third data set, named THIRD; the average block length of the data in THIRD is 1024 bytes and you expect to write 100 blocks of data.

You would code:

```
//JOBA       JOB      ...
//STEP1      EXEC     PGM=INVENT
//MASTER     DD       DSN=DEPT41,DISP=(NEW,CATLG),UNIT=2314,
//                    VOL=(PRIVATE,,2),SPACE=(CYL,(8),,CONTIG)
//SUB1       DD       DSN=FIRST,DISP=(,KEEP),
//                    SUBALLOC=(CYL,(3,,10),MASTER)
//STEP2      EXEC     PGM=REDO
//SUB2       DD       DSN=SECOND,DISP=(,KEEP),UNIT=(,2)
//                    SUBALLOC=(TRK,(50,25),STEP1.MASTER)

//JOBB       JOB      ...
//STEPA      EXEC     PGM=CONVERT
//MASTER     DD       DSN=DEPT41,DISP=OLD
//SUB3       DD       DSN=THIRD,DISP=(,KEEP),
//                    SUBALLOC=(1024,(100),MASTER)
```

Disposition Processing of Data Sets

Disposing of data sets at the end of a job step is known as **disposition processing.** You request disposition processing by coding the DISP parameter on the DD statement defining the data set. In the DISP parameter, you can code:

- data set status as the first subparameter, indicating if the data set is new, is old, can be shared with other jobs, or can be lengthened;
- normal disposition as the second subparameter, indicating how the data set should be handled if the job step terminates normally;
- conditional disposition as the third subparameter, indicating how the data set should be handled if the job step terminates abnormally.

If you do not code one of the subparameters, or omit the DISP parameter entirely, the system supplies default values, as described under "Default Disposition Processing."

Specifying Data Set Status

You indicate a data set's status by coding one of the following:

- NEW - the data set is being created in this job step.
- OLD - the data set existed before this job step.
- SHR - the data set existed before this job step and can be read simultaneously by other jobs.
- MOD - the system assumes the data set exists and will position the read/write mechanism after the last record in the data set; if the system cannot find volume information for the data set, the system assumes the data set will be created in the job step.

When you code SHR, you are requesting **shared control** of the data set and your job should be reading the data set only. When you code NEW, OLD, or MOD, you are requesting **exclusive control** of the data set. Shared and exclusive control are described in the chapter "Insuring Data Set Integrity."

Specifying a Disposition for the Data Set

You can specify one disposition, called a normal disposition, to be used when the job step terminates normally (i.e., successfully) and another disposition, called the conditional disposition, to be used when the job step terminates abnormally.

For normal disposition, you can request as the second subparameter that the data set be:

- deleted by coding DELETE
- kept by coding KEEP
- cataloged by coding CATLG
- uncataloged by coding UNCATLG
- passed by coding PASS

For conditional disposition (the third subparameter of the DISP parameter), you can code all of the above with the exception of PASS.

Disposition processing differs for data sets on direct access volumes and data sets on magnetic tape volumes. A direct access volume contains a **volume table of contents** (VTOC) which consists of control blocks describing the data sets and available space on the volume.

The handling of tape and direct access volumes when you specify a particular disposition is described below.

Deleting a Data Set

Specifying DELETE requests that the data set's space on the volume be released at the end of the job step (when coded as the normal disposition) or if the step abnormally terminates (when coded as the conditional disposition). If the data set resides on a tape volume, the tape is rewound and the volume is available for use by other job steps. If the data set exists on a direct access volume, the control block describing the data set is removed from the VTOC and the space on the volume is then available to other data sets.

In one case, however, a data set on a direct access volume will not be deleted, even though you specify DELETE: when the expiration date or retention period has not expired. You can specify a length of time that a data set must be kept by assigning a retention period or expiration date in the LABEL parameter on the DD statement. Specifying a retention period or expiration date is described in the **OS/VS JCL Reference**, GC28-0618, under "The LABEL Parameter."

If you are deleting a cataloged data set, the entry for the data set in the system catalog is also removed, provided the system obtained volume information for the data set from the catalog (**i.e.**, the volume's serial number was not coded on the DD statement). If the system did not obtain volume information from the catalog, the data set is still deleted but its entry in the catalog remains. If an error is encountered while attempting to delete a data set, its entry in the catalog will not be removed. (The data set will or will not be deleted, depending on where the error occurs.) You can use the IEHPROGM utility program to delete an entry from the catalog. (The IEHPROGM utility is described in **OS/VS Utilities**, GC35-0005.)

DELETE is the only valid conditional disposition for a data set with no name or a temporary name.

Keeping a Data Set

Specifying KEEP instructs the system to keep a data set intact until a subsequent job step or job requests that the data set be deleted or until the expiration date or retention period is passed. (You can specify an expiration date or retention period, indicating the length of time a data set must be kept, in the LABEL parameter on the DD statement. If you do not specify a time period, the system assumes a retention period of 0 days. Coding an expiration date or retention period is described under "The LABEL Parameter" in the **OS/VS JCL Reference**, GC28-0618.)

For data sets on direct access devices, the entry describing the data set in the VTOC and the data set itself is kept intact. For data sets on tape, the volume is rewound and unloaded and a KEEP message is issued to the operator.

Cataloging a Data Set

To more easily keep track of and retrieve data sets, the system provides a cataloging facility. The catalog is itself a data set that is organized into levels of indexes; entries in the lowest-level index contain data set names and volume information for the data sets. By cataloging data sets, you can group collections of data sets; when retrieving a cataloged data set, you do not have to specify volume information, you need only code the DSNAME parameter and a status in the DISP parameter other than NEW. If the data set name is a qualified name (a qualified name is composed of multiple names separated by periods; each name corresponds to an index level in the catalog), all but the lowest-level index must exist in the catalog. (You create indexes in the catalog by using the IEHPROGM utility -- see **OS/VS**

Utilities, GC35-0005; qualified names are described in the **OS/VS JCL Reference**, GC28-0618, under "Specifying the DSNAME Parameter.")

To request that a data set be cataloged, code CATLG as the disposition; the system creates an index entry in the catalog that points to the data set. The disposition CATLG implies KEEP.

You can specify a disposition of CATLG for an already cataloged data set. This should be done when you are lengthening the data set with additional output (a status of MOD is coded) and the data set may exceed one volume. If the system obtained volume information for the data set from the catalog (**i.e.**, the volume's serial number was not coded on the DD statement) and you code DISP=(MOD,CATLG), the system updates the entry to include the volume serial numbers of any additional volumes.

A collection of cataloged data sets that are kept in chronological order can be defined as a **generation data group** (GDG). The entire GDG is stored under a single data set name; each data set within the group, called a **generation data set**, is associated with a generation number that indicates how far removed the data set is from the original generation. If you do not code a disposition when creating a generation data set, or specify a disposition other than CATLG, the system assumes CATLG. For more information on defining and creating generation data groups, see the **OS/VS JCL Reference**, GC28-0618.

If the data set name is enclosed in apostrophes, you must not assign the disposition CATLG to it.

Uncataloging a Data Set

To remove the entry describing a data set from the catalog, code UNCATLG as the disposition. Specifying UNCATLG does **not** request the initiator to delete the data set -- just the reference in the catalog is removed. When you request use of the data set in a subsequent job or job step, you must include volume information on the DD statement.

Passing a Data Set

If more than one step in a job requests the same data set, each step using the data set can pass the data set for use by a subsequent step. When a data set is passed, the volume containing the data set remains mounted; when a subsequent step uses that data set, allocation and disposition processing does not have to be performed for the data set.

To pass a data set, you code PASS as the normal disposition; PASS cannot be specified as the conditional disposition. You continue to code PASS each time the data set is referred to until the last time it is used in the job. At this time, you assign it a final disposition. If you do not assign the data set a final disposition, the system deletes the data set if it was created in the job and keeps the data set if it existed before the job.

If the data set exists on a direct access volume, the volume remains mounted. A magnetic tape volume containing a passed data set remains mounted, but the tape is rewound between steps unless you request otherwise in the CLOSE macro instruction. (A description of the CLOSE macro instruction is included in **OS/VS Data Management Macro Instructions**, GC26-3793.)

Default Disposition Processing

If you do not code the DISP parameter, or omit one of the subparameters, the system supplies default values.

If you do not specify a data set status, the system assumes NEW. If you do not code the second or third subparameters, the system determines how a data set should be handled according to the status of the data set: data sets that existed before the job step are automatically kept (data sets for which you coded OLD, SHR, or MOD when volume information is available): data sets created in the job step are automatically deleted (data sets for which you coded NEW, MOD when volume information is not available, or for which you did not code a status).

If a step abnormally terminates before it actually begins execution (for example, during allocation of units and volumes or direct access space), the system ignores the disposition you code and again automatically keeps existing data sets and deletes new data sets.

For example, if you code:

```
DISP=( ,PASS,CATLG )
```

the system assumes the data set is new. If the job step abnormally terminates during its execution, the system will catalog the data set, as instructed by the conditional disposition of CATLG. If, however, the step abnormally terminates before it actually begins execution, the system will delete the data set, since it is a new data set.

Bypassing Disposition Processing

If you define a data set as a dummy data set, the DISP parameter, if coded, is ignored and disposition processing is not performed. For details, see "Defining a Dummy Data Set."

Insuring Data Set Integrity

Your job must receive control of the nontemporary data sets it requests: you can request either **exclusive** control, allowing no other job to use the data set, or **shared** control, allowing the data set to be used by other jobs that also request shared control. The process of securing control of data sets for use by a job is called **data set integrity processing.**

Data set integrity processing avoids conflict between two or more jobs that request use of the same data set. For example, two jobs, one named READ and another named MODIFY, both request the data set FILE. READ wants only to read and copy certain records; MODIFY deletes some records and changes other records in the data set FILE. If both jobs have control of FILE concurrently, READ cannot be certain of the records contained in FILE --cannot be sure of the integrity of the data set. MODIFY should have **exclusive control** of the data set; READ can **share** control of FILE with other jobs that also want only to read the data set. You indicate the type of control a data set requires in the DISP parameter on the DD statement defining the data set.

Exclusive Control of a Data Set

When a job has **exclusive control** of a data set, no other job can use that data set until the job having exclusive control terminates. A job should have exclusive control of a data set in order to modify, add, or delete records.

In some cases, you may not need exclusive control of the entire data set. You can request exclusive control of a block of records by coding the DCB, READ, WRITE, and RELEX macro instructions. (These instructions are described in **OS/VS Data Management Macro Instructions**, GC26-3793.)

To request exclusive control of a data set, you code NEW, OLD, or MOD as the first subparameter of the DISP parameter.

Shared Control of a Data Set

A data set on a direct access storage device can be used concurrently by several jobs, if these jobs request shared control of the data set; however, none of the jobs should change the data set in any way.

To request shared control, you code SHR as the first subparameter in the DISP parameter. If more than one step of your job requests a data set, you must code SHR every time you define the data set if it is to be used by concurrently executing jobs. Data set integrity processing is performed once for a job; a data set has one type of control -- either shared or exclusive -- for the entire job. If you code NEW, OLD, or MOD on any reference to a data set, the system assigns exclusive control to the data set for the entire job; a reference requesting exclusive control will override any number of references requesting shared control.

How the System Performs Data Set Integrity Processing

Data set integrity processing is performed only for nontemporary data sets. (A temporary data set is, by definition, a new data set that is created and deleted in the same job. Another job cannot request a temporary data set; therefore, there is no possibility of conflict, and data set integrity processing is unnecessary.)

The system recognizes a nontemporary data set by the data set name assigned to it in the DSNAME parameter. You do not have to code the DSNAME parameter for temporary data sets; if you do, the name begins with the characters &&. Any data set name, then, that does not begin with && indicates a nontemporary data set, even though the data set may be created and deleted within the job. (A data set name preceded by one ampersand is treated as a symbolic parameter if a value is assigned to it; if no value is assigned, a data set name preceded by only one ampersand is treated as the name of a temporary data set. Symbolic parameters and assigning values to symbolic parameters are described under "Using Symbolic Parameters.")

To secure control of a data set for a job, the system **enqueues** on the data set, marking the data set as requested by that job and noting what kind of control was requested. The job will receive control of the data set if:

- the data set is not being used by another job, or
- the data set is being used by another job but both the job requesting the data set and the job using the data set request shared control.

For example, a job named READ requests shared control of a data set named FILE; if FILE is being used by a job named LOOKAT and LOOKAT also requests shared control, both READ and LOOKAT can use the data set at the same time.

A job will **not** receive control of a data set if:

- the data set is being used by another job and that job has exclusive control, or
- the data set is being used by another job (with either exclusive or shared control), but the job requesting use of the data set requests exclusive control.

For example, the job named MODIFY requests exclusive control of the data set FILE; FILE is already being used by the job LOOKAT. MODIFY cannot receive control of the data set until LOOKAT has terminated.

If any of the data sets a job requests are not available, the system issues a message to the operator indicating the unavailable data sets. The message and the operator's response differ in VS1 and VS2.

In VS1, the operator replies with one of the following responses:

- RETRY -- the initiator attempts to enqueue again on the data sets.
- CANCEL -- execution of the job is cancelled.
- HOLD -- the job is placed in a HOLD state until released by the operator when the data sets are available.

In VS2, the system issues the message "JOB IS WAITING ON DATA SETS." The initiator that started the job will automatically wait until the required data sets become available, unless the operator cancels the job.

Special Data Sets

Data sets can be defined to satisfy a special purpose. Such data sets are usually defined with a special ddname, a specific data set name, or a specific parameter. This section describes three types of special data sets:

- Creating and Using Private and Temporary Libraries
- Defining a Dummy Data Set
- Using a Dedicated Data Set for Allocating a Temporary Data Set

Creating and Using Private and Temporary Libraries

A library is simply a partitioned data set -- a data set in direct access storage that is divided into partitions, called members, each of which can contain a program or part of a program. Each partitioned data set contains a directory (or index) that the control program can use to locate a program in the library. All programs that can be executed must exist in a library --**i.e.**, must be members of a partitioned data set. There are three types of libraries:

- the system library
- private libraries
- temporary libraries

The system library is a partitioned data set named SYS1.LINKLIB that contains frequently used programs and programs used by the system. You need not define the system library in your job; the system will automatically look in the system library for a program you want executed.

A private library is a partitioned data set that contains programs not used frequently enough to warrant being included in the system library. You inform the system that a program exists in a private library by coding a DD statement defining that library. You can define a private library to be used throughout the job by coding a DD statement with the ddname JOBLIB, or define a library to be used in a specific step by coding a DD statement with the ddname STEPLIB.

A temporary library is a partitioned data set created in the job to store a program , as a member of the partitioned data set, until it is executed in a following step. For example, if in your job you want to assemble, linkage edit, and then execute a program, you must make the output of the linkage editor a member of a library. Any library that you create and delete in the same job is a temporary library.

To execute a program contained in a library, code the PGM parameter as the first parameter on the EXEC statement. If the program exists in the system library, simply code the program name, **i.e.**, PGM=program name. If the program exists in a private library, code either PGM=program name or PGM=*.stepname.ddname or PGM=*.stepname.procstepname.ddname. Stepname and procstepname identify the job step or job step and procedure step defining the library; the named DD statement must define the library; the named DD statement must define the program as a member of a partitioned data set. To call a program contained in a temporary library that is not defined with a JOBLIB DD or STEPLIB DD statement you must code PGM=*.stepname.ddname or PGM=*.stepname.procstepname.ddname.

If you define a private library, the system looks first in that library for a program you want executed; if it does not find the program in the private library, it then searches the system library.

This chapter describes how to code JCL statements to create or retrieve private and temporary libraries. Complete information on creating a partitioned data set, adding members to and deleting members from a partitioned data set, is included in **OS/VS Data Management Services Guide**, GC26-3783.

Creating a Private Library

Use the JOBLIB DD statement to create a private library. The JOBLIB DD statement must appear immediately after the JOB statement --do not use the ddname JOBLIB unless you are defining a private library. The library defined with a JOBLIB DD statement is automatically available to every step in your job. (The STEPLIB DD statement is included among the DD statements in a step and is available only to that step unless you pass the library or redefine it in subsequent steps; since the library defined on a JOBLIB DD statement is available to every step, it is easier to create a library with the JOBLIB DD statement.)

When you create the library on the JOBLIB DD statement, you are creating a partitioned data set. Steps in your job must add members to the library before those members (programs) can be used by subsequent steps.

On the JOBLIB DD statement, you assign the library a name in the DSNAME parameter, give unit and volume information in the UNIT and VOLUME parameters (a partitioned data set must be contained on one direct access volume; if, however, you make a nonspecific volume request, you need not code the VOLUME parameter), request space for the entire library in the SPACE parameter, and assign a data set status and disposition in the DISP parameter. Code NEW as the data set status and either CATLG or PASS as the data set disposition. When you specify CATLG, the library is cataloged, available throughout the job, and kept at the end of the job. When you specify PASS, the library is available throughout the job, but is deleted at job termination. (If you do not code a disposition, or code a disposition other than CATLG or PASS, the system assumes PASS.) You must also code the DCB parameter if complete data control block information is not included in the data set label.

Adding Members to a Private Library

You add members to the library in job steps within the job. Code a DD statement that defines the library and names the member to be added to the library: in the DSNAME parameter, follow the library name with the name of the program you are adding to the library, for example, DSNAME=LIBRARY(PROGRAM). Do not code the SPACE parameter: you requested space for the entire library on the JOBLIB DD statement. In the DISP parameter, specify MOD as the data set status: the partitioned data set already exists since you created it in the JOBLIB statement, and you are lengthening it with a new member. If you cataloged the library in the JOBLIB DD statement, .e., coded DISP=(NEW,CATLG), you must not respecify CATLG when you add a member: you need not code a disposition at all. For a cataloged library, you do not have to specify unit and volume information, except in one instance: if you are adding a member to the library in the first step of your job, you must supply unit and volume information; the library is not cataloged until the first step completes the execution. You can refer to the JOBLIB DD statement for unit and volume information by coding VOL=REF=*.JOBLIB.

In the following example, JOBLIB DD statement creates a library named GROUPLIB; STEP1 adds the programnamed RATE to the library; STEP2 calls the program RATE:

```
//EG          JOB
//JOBLIB      DD        DSNAME=GROUPLIB,DISP=(NEW,CATLG),
//                      UNIT=2314,VOL=SER=727104,
//                      SPACE=(CYL,(50,3,4))
//STEP1       EXEC      PGM=FIND
//ADDPGMD     DD        DSNAME=GROUPLIB(RATE),DISP=OLD,
//                      VOL=REF=*.JOBLIB
//STEP2       EXEC      PGM=RATE
```

In STEP1, the system looks for the program named FIND in the system library --the private library created on the JOBLIB DD statement does not actually exist until a member is added to it. In STEP2, the system looks for the program named RATE first in the private library.

Retrieving an Existing Private Library

If you are retrieving several programs from one library (i.e., several steps in your job will be using the library), use the JOBLIB DD statement to define the library: the library will be available in every step of the job for which you do not code a STEPLIB DD statement. The JOBLIB DD statement must appear immediately after the JOB statement. To make a library available in a single step, define the library on a STEPLIB DD statement. The STEPLIB DD statement is included with the DD statements for a step (in no specific order) and is available only to that step, unless you pass the library and retrieve it in a subsequent step. Use the ddnames JOBLIB and STEPLIB only when you are defining private libraries.

The system will search for a program in the private library you define before it searches the system library. If both JOBLIB and STEPLIB DD statements appear in a job, the STEPLIB definition has precedence, i.e., the private library defined by the JOBLIB DD statement is not searched for any step that contains the STEPLIB definition. If you want the JOBLIB definition ignored but the step does not require use of another private library, define the system library on the STEPLIB DD statement:

```
    //STEPLIB       DD      DSNAME=SYS1.LINKLIB,DISP=SHR
```

You retrieve a private library as you would any partitioned data set: if the library is cataloged, or in the case of a STEPLIB definition, passed from a previous step, you need not specify unit and volume information; otherwise, you must code the UNIT and VOLUME parameters.

For both cataloged and uncataloged libraries, you code: the DSNAME parameter, specifying the name of the library; the DCB parameter, if complete data control block information is not included in the data set label; and the DISP parameter, specifying data set status and disposition. Normally, you will want to specify SHR as the data set status: SHR indicates that the data set is old, but also allows other jobs to simultaneously use the library. All references to the library in your job must specify SHR if the data set is to be shared; do not code SHR, however, if you will be adding members to the library in your job. (A more thorough discussion of sharing a data set is included in the chapter "Insuring Data Set Integrity.") Code PASS as the data set disposition for a library defined on the JOBLIB DD statement: PASS makes the library available throughout the job. (If you do not code a disposition, the system assumes PASS.) For a library defined on a STEPLIB DD statement, code any valid disposition, depending on how you want the data set treated after its use in the job step: for example, if the library is not cataloged, and you want it to be cataloged, code CATLG; if you want the library deleted, code DELETE.

The following job includes both JOBLIB DD and STEPLIB DD statements:

```
//CAMILLE   JOB
//JOBLIB    DD      DSNAME=LIB5.GRP4,DISP=SHR
//STEP1     EXEC    PGM=FIND
//STEP2     EXEC    PGM=GATHER
//STEPLIB   DD      DSNAME=ACCOUNTS,DISP=(SHR,KEEP),
//                  UNIT=2314,VOL=SER=727104
```

In STEP1, the system searches the library named LIB5.GRP4, defined on the JOBLIB DD statement, for the program named FIND. In STEP2, the system searches the library named ACCOUNTS, defined on the STEPLIB DD statement, for the program named GATHER. If the program is not found in the private library, the system searches the system library.

You can add a program to an existing library by coding a DD statement in a job step that defines the library and names the program to be added --see "Adding Members to a Private Library" for details on coding this DD statement. The new member must be added to the library before it can be executed --i.e., the step that adds the program to the library must precede the step that calls the program. Do not code SHR as the data set's status when modifying the library.

Concatenating Private Libraries

If your job uses programs contained in several libraries, you can concatenate these libraries on one JOBLIB DD statement or one STEPLIB DD statement; all the libraries you concatenate must be existing libraries. Omit the ddname from all the DD statements defining the libraries, except the first:

```
//JOBLIB     DD   DSNAME=D58.LIB12,DISP=(SHR,PASS)
//           DD   DSNAME=D90.BROWN,DISP=(SHR,PASS),
//                UNIT=3330,VOL=SER=411731
//           DD   DSNAME=A03.EDUC,DISP=(SHR,PASS)
```

This entire group must appear immediately after the JOB statement. When you concatenate libraries using STEPLIB as the ddname, the entire group appears as part of the DD statements for the step.

The system will search the libraries for a program in the order in which the DD statements defining the libraries are coded.

Temporary Libraries

Temporary libraries are libraries that are created and deleted within the job. It is not necessary to define a temporary library on a JOBLIB DD or STEPLIB DD statement: simply code a DD statement creating a partitioned data set and adding the program to it in the step that produces the program. You can then retrieve this program in a subsequent step.

For example, STEP2 illustrated below calls the program IEWL, which linkage edits object modules to form a load module that can be executed. You must place the results of the linkage edit step in a library, so that a subsequent step can use those results. Since the results are not a program other jobs will call, it is logical to place the program in a temporary library:

```
//STEP2      EXEC     PGM=IEWL
//RESULT     DD       DSNAME=&&PARTDS(PROG),UNIT=2314,
//                    DISP=(NEW,PASS),SPACE=(1024,(50,20,1))
//STEP3      EXEC     PGM=*.STEP2.RESULT
```

You call the program in STEP3 by naming the step in which the library was created and the name of the DD statement that defines the program as a member of a library. If STEP2 had called a procedure, and the DD statement named RESULT were included in PROCSTEP3 of the procedure, you would code PGM=*.STEP2.PROCSTEP3.RESULT.

Defining a Dummy Data Set

To save processing time, you might not want a data set to be processed every time the job is executed. For example, while testing a program, you might want to suppress the writing of an output data set until you are sure it will contain meaningful output; you might want to skip the reading of a data set to be used only once a week. When you define a dummy data set, input/output operations are bypassed, disposition processing is not performed, and devices and storage are not allocated to the data set.

You define a dummy data set by:

- coding the DUMMY parameter on the DD statement, or
- assigning the data set name NULLFILE in the DSNAME parameter on the DD statement.

Coding the DUMMY Parameter

Code DUMMY as the first parameter on the DD statement. DUMMY is a positional parameter: it must precede all keyword parameters on the DD statement.

When the DUMMY parameter is coded, all other parameters on the DD statement, with the exception of the DCB parameter, are ignored. (The parameters are checked for syntax, however; if a parameter is coded incorrectly, a JCL error message is issued.) Therefore, although you may code UNIT, VOLUME, and DISP, no devices or external storage is allocated to the data set and no disposition processing is performed. The DCB parameter must be coded if you would code it for normal I/O operations. For example, when an OPEN routine requires a BLKSIZE specification to obtain buffers and BLKSIZE is not specified in the DCB macro instruction, you should supply this information in the DCB parameter on the DD statement. (A description of the DCB parameter is included in the **OS/VS JCL Reference**, GC28-0618.) When a DD statement that overrides a procedure DD statement contains the DUMMY parameter, all of the parameters coded on the procedure DD statement are nullified, except for the DCB parameter.

If you request unit or volume affinity with a dummy data set, the data set requesting affinity is assigned a dummy status. (Unit and volume affinity is described in the chapter "Requesting Units and Volumes for a Data Set.")

When you want the data set to be processed, replace the DD statement containing the DUMMY parameter with a DD statement containing the parameters required to define the data set. When a procedure DD statement contains the DUMMY parameter, you can nullify it by coding the DSNAME parameter on the overriding DD statement and assigning a data set name other than NULLFILE.

Note: In VS2, the time-sharing option (TSO), which can be included in the system during system generation, provides conversational time sharing from remote terminals. During the TSO LOGON procedure, you can code a DD statement and specify the parameter DYNAM to indicate that dynamic allocation of data sets is to be used; no devices or external storage is allocated to the data set, but space is reserved in internal tables so that data set requirements that arise during the terminal session can be satisfied. If a DD DYNAM statement is used in the background (batch processing), or in the foreground before allocation, the DYNAM parameter has the same effect as coding DUMMY.

Coding DSNAME=NULLFILE

Assigning the name NULLFILE in the DSNAME parameter has the same effect as coding DUMMY. The data set is assigned a dummy status; no devices or storage is allocated and no disposition processing is performed. All parameters except for DSNAME and DCB are ignored. (You must code the DCB parameter when defining a dummy data set if you would code it for normal I/O operations.)

When you want the data set to be processed, replace the name NULLFILE with another data set name. (Assigning names to data sets is described in the **OS/VS JCL Reference**, GC28-0618, under "Specifying the DSNAME Parameter.")

Requests to Read or Write a Dummy Data Set

When your program asks to read a dummy data set, an end-of-data-set exit is taken immediately. When your program requests that the data set be written, the request is recognized but no data is transmitted. Your program must use the basic sequential access method (BSAM) or queued sequential access method (QSAM) when requesting to write a dummy data set; if any other access method is used, the job is terminated.

If you define a data set as a dummy data set, the DISP parameter, if coded, is ignored and disposition processing is not performed. For details, see "Defining a Dummy Data Set."

Using a Dedicated Data Set for Allocating a Temporary Data Set

Temporary data sets are data sets that are created and deleted within the same job. To save the time required to repeatedly assign and release space to temporary data sets, your installation can define **dedicated data sets.**

To create a dedicated data set, your installation adds a DD statement defining the dedicated data set to an initiator procedure. (An initiator procedure is simply the cataloged procedure for an initiator.) When the initiator is started, space is allocated to the dedicated data set; every job step running under the initiator can then use the dedicated data set as a temporary data set.

Defining the Temporary Data Set

The parameters that define the temporary data set are illustrated in Figure 3.

Parameter	**Comments**
DSNAME=&ddname	Ddname specifies the name of the DD statement in the initiator procedure that defines the dedicated data set.
SPACE	You must request space in terms of average block length; any secondary quantity you code overrides a secondary quantity specified for the dedicated data set. If the data set is partitioned, include a request for the directory.
UNIT	This must be coded, in case the dedicated data set is not used. You can request either a magnetic tape or direct access device.
DISP	If coded, the DISP parameter must specify (NEW,DELETE).
DCB	Unless you code required DCB subparameters for the data set, the system will use DCB subparameters coded by a previous user of the dedicated data set. If you code a secondary quantity in the SPACE parameter, you must specify the maximum block length of your data in the BLKSIZE subparameter.

Figure 3. Defining a Temporary Data Set in order to Use the Space Allocated to a Dedicated Data Set

The system will use the space allocated to the dedicated data set for your data set, unless:

- The total space (primary and secondary requests) requested for the temporary data set exceeds the total space (primary and secondary requests) allocated to the dedicated data set.
- The temporary data set and dedicated data set do not both have either sequential or partitioned organization. For example, if the dedicated data set is partitioned (therefore, space for a directory is requested in the SPACE parameter) and the temporary data set is sequential (no space for a directory is requested), the dedicated data set will not be used.
- Both the temporary and dedicated data sets are partitioned, but the temporary data set's request for a directory is larger than the space allocated for the dedicated data set's directory.
- The temporary data set is an ISAM data set.

If any of these conditions are true, normal allocation of the temporary data set will occur.

For example, the ddname of a dedicated data set is DEDICAT. To request that the space allocated to DEDICAT be used for a temporary data set, you code:

```
//DD1    DD      DSNAME=&DEDICAT,UNIT=2314,
//               SPACE=(1024,(100,25)),DISP=(NEW,DELETE),
//               DCB=BLKSIZE=2048
```

Your installation sets up the guidelines for using dedicated data sets. When an initiator is started, the operator can assign job classes to it to process: certain job classes can always be assigned to an initiator containing a dedicated data set. When you assign your job to one of these job classes, you can use the dedicated data set. The same ddname defining a dedicated data set can be included in more than one initiator procedure: you could code this ddname when you want to use a dedicated data set. The guidelines for knowing when a dedicated data set will be available to your job depend on the individual installation.

Obtaining Output

Output from your job can include a listing of JCL statements and messages, a dump in the event of abnormal termination, and output data sets. You request output by coding JCL parameters. Output data sets and dumps must be defined on DD statements; you request listings of JCL statements and messages by coding parameters on the JOB statement. When you request that an output data set be printed, you can also request options that control how the data set is printed.

This section includes five chapters:

- Controlling the Output Listing of JCL Statements, Messages, and Dumps
- Writing Output Data Sets
- Requesting Multiple Copies of an Output Data Set (VS1 only)
- Printer Forms and Print Chain Control
- Controlling Output to a Workstation (VS1 only)

Controlling the Output Listing of JCL Statements, Messages, and Dumps

The system produces messages about your job concerning allocation of units and volumes, disposition of data sets, and termination of job steps and the job. You can request that these messages, called allocation/termination messages, and/or the JCL statements from your job and from cataloged procedures called by your job be included on an output listing. You route allocation/termination messages to an output device by assigning the messages from your job to an output class.

In the event of abnormal termination of a step, you can also request a **dump**, containing the contents of parts of virtual storage. You must include a DD statement definign a data set to contain the dump with the job control statements for the step.

Requesting Listing of JCL Statements and Messages

By coding the MSGLEVEL parameter on the JOB statement, you inform the system of what statements and messages you want included on the output listing.

As the first subparameter, you code 0, 1, or 2, to indicate what statements you want:

0 only the JOB statement.

1 all JCL statements from the job (which includes in-stream procedures) and from cataloged procedures called by the job, including the internal representation of cataloged procedure statements after symbolic parameters are substituted.

2 input JCL statements from the job, which includes in-stream procedures. (Statements from cataloged procedures called by the job are not included.)

The notation used on the output listing to identify cataloged and in-stream procedure statements is described in the chapter "Using Cataloged and In-Stream Procedures."

As the second subparameter, you code 0 or 1 to instruct the system to write:

0 no allocation/termination messages, unless the job abnormally terminates. If the job does terminate abnormally, allocation/termination messages are included on the output listing.

1 all allocation/termination messages.

If you omit the MSGLEVEL parameter or one of the subparameters, the default value in the input reader procedure is used. This default is (1,1), which requests all JCL statements and all allocation/termination messages, unless changed by your installation.

For example, if you want only the JOB statement displayed and all allocation/termination messages, code:

```
//PGM   JOB ...MSGLEVEL=(0,1)
```

If the IBM-supplied default is used, you can omit the second subparameter and code:

```
//PGM   JOB ...MSGLEVEL=0
```

Assigning Messages to an Output Class

To route system messages to a system output device, you assign the messages to an output class. Output classes, designated by a letter (A-Z) or a number (0-9), are defined by the installation to group output that will be written to the same device. For example, class W might be reserved for output to be written to a printer and requiring a special form.

You assign messages to an output class by coding the MSGCLASS parameter on the JOB statement:

```
//PGM   JOB ...MSGCLASS=W
```

If you do not code the MSGCLASS parameter, a default value established in the input reader procedure is used. This default is A unless changed by your installation.

Ordinarily, you will want system messages for a job to be written to the same device as output data sets from that job: assign the same output class to output data sets and messages or omit the MSGCLASS parameter and assign the default output class for messages to output data sets. (You assign data sets to an output class in the SYSOUT parameter on the DD statement -- see the chatper "Writing Output Data Sets.")

Requesting an Abnormal Termination Dump

To obtain a dump in the event of abnormal termination of a job step, you must code a DD statement defining a data set to which the dump can be written. The name of the DD statement must be either SYSABEND or SYSUDUMP. (If you include more than one DD statement defining a dump, only the last statement is used.)

When you code SYSUDUMP as the ddname, your dump contains only the contents of the processing program's virtual storage area. When you code a SYSABEND DD statement, the contents of your dump differs in VS1 and VS2:

- **in VS1**, coding SYSABEND provides a dump containing the processing program's virtual storage area, the system nucleus, the pageable supervisor, and the entire system queue area (the system queue area, SQA, is an area of virtual storage reserved for system-related control blocks).
- **in VS2**, the SYSABEND DD statement provides a dump containing the processing program's virtual storage area, the system nucleus, the entire system queue area (the SQA is an area of virtual storage reserved for system-related control blocks), all local system queue areas (an LSQA is one or more segments associated with each virtual storage region that contains job-related system control blocks), and any active link pack area (LPA) modules for the failing task (an LPA is an area of virtual storage containing selected reenterable and serially reusable routines that can be used concurrently by all tasks in the system). If GTF is active in the system and performing an internal trace, you will receive GTF trace records; if GTF is active but performing an external trace, no trace information is included in the dump. (GTF is a feature of OS/VS that allows you to trace selected system events.)

Complete descriptions of dumps and information on reading dumps are included in the **OS/VS1 Debugging Guide**, GC24-5093, and the **OS/VS2 Debugging Guide**, GC28-0632.

To have the dump printed, you either assign the dump to an output class in the SYSOUT parameter on the DD statement or code the UNIT parameter and specify the unit record device. For details, see the chapter "Writing Output Data Sets." To store the dump, define the data set as you would any other data set, coding the DSNAME, DISP, UNIT, VOLUME, and, if the data set will exist on a direct access device, SPACE parameters. In the DISP parameter, code DELETE as the normal disposition: if the job terminates normally, you do not need the dump. You can code KEEP or CATLG as the conditional disposition, but it is not necessary. The system will not delete a data set defined with a SYSABEND DD or SYSUDUMP DD statement if the step abnormally terminates.

The following DD statement requests that a dump be printed if the job step abnormally terminates:

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

To store the dump, you could code:

```
//SYSUDUMP  DD      DSNAME=DUMP,DISP=(NEW,DELETE),
//                  UNIT=2400,VOL=SER=147958
```

Writing Output Data Sets

There are two ways to instruct the system to write output data sets:

1. by assigning the data set to an output class;
2. by specifying the device on which the output will be written.

When you assign a data set to an output class, it is written by routines called output writers: in VS1, output writers include the system output writer and the direct system output (DSO) writer; in VS2, the output writer is the system output writer. When you specify the device you want, allocation routines assign that device, if it is available, exclusively to the job that requests it and data management routines write the output.

Assigning Output Data Sets to Output Classes

The purpose of output classes is to group output with similar characteristics that will be written to the same device. There are 36 possible output classes, each designated by a letter from A through Z or a number from 0 through 9. The letter and number names have no inherent meaning -each installation defines its own output classes when the system is generated. For example, output class W might contain output to be written to a printer and requiring a special form; class J might be reserved for high-volume output.

To assign an output data set to an output class, you code the SYSOUT parameter on the DD statement defining the data set:

```
//DATASET   DD  SYSOUT=W
```

If you want the output data set and the messages from your job to be printed on the same output listing, specify the same output class in the SYSOUT parameter as you specified for messages in the MSGCLASS parameter. If you omitted the MSGCLASS parameter, code the default output class for messages in the SYSOUT parameter. (For details on coding the MSGCLASS parameter, see the chapter "Controlling the Output Listing of JCL Statements, Messages, and Dumps.")

Processing Output Classes

The operator controls the processing of output classes by issuing commands to start writers (the START command), modify the classes of output the writer processes (the MODIFY command), and stop the writer (the STOP command). In the START command, the operator can assign output classes to the writer to be processed; if he does not specify output classes, defaults from the writer procedure are used. The writers used to process output classes differ in VS1 and VS2.

The **system output writer** is available in both VS1 and VS2; it is the most efficient way to write output. The output is first written to a direct access device. When a system output writer is started, it writes the output from the direct access device to a system output device according to the output classes it was assigned to process and, within an output class, according to the priority of the job which produced the output. (Output in a single output class from jobs with the same priority or from a single job are written in a first in/first out order.) System output devices are simply the devices on which output classes are written; they include printers, punches, and magnetic tape.

In VS1, the **direct system output (DSO) writer** is also available to write output classes. The DSO writer writes output data sets directly from a job to a system output device while the job is executing; messages from the job are first written to a direct access device and then written to a system output device at job termination.

Using an Installation-Written Writer Routine

Instead of using the IBM-supplied output writer routine, your installation can provide its own routine. If you want your installation's routine to be used, specify the name of the routine in the SYSOUT parameter. For example, if you are assigning an output data set to class B and want the installation-written routine named WRITE to be used to write the data set, code:

```
//OUTPUT    DD  SYSOUT=(B,WRITE)
```

Delaying the Writing of an Output Data Set (VS1 only)

You can delay the writing of an output data set until the operator requests that the data set be written. The reasons to delay writing a data set are varied, for example: if a data set is very large and not immediately needed, you might not want to monopolize an output device until other, smaller data sets are written: if a data set requires special forms that are not immediately available --e.g., a data set containing payroll information requires 5,000 payroll checks-- the data set will not be printed until the operator supplies those forms; if you are routing the data set to another destination, the data set will not be printed until that destination requests it.

Code HOLD=YES on the DD statement defining the data set:

```
//LARGE     DD      SYSOUT=W,HOLD=YES
```

The data set is placed on a hold queue until the operator releases it by issuing a ROUTE command. You must notify the operator when you code HOLD=YES for a data set: when a data set is placed on a hold queue, no message is sent to the operator. If you are routing the data set to another destination, you can notify that destination by coding a SEND command. Details on routing a data set to another destination and coding the SEND command are included in the chapter "Controlling Output to Workstations (VS1 only)."

Job Separators

To make it easier to separate the output from different jobs, your installation can include a routine in the writer procedure to write job separators, for example, three listing pages or three punched cards containing the name of the job whose output follows and the output class.

In VS1, the operator can specify one of two writer procedures for direct system output (DSO) processing: the DSO procedure or the DSOJS procedure. If DSOJS is specified, job separators are automatically written to separate the output from different jobs.

Specifying the Device

To write an output data set without using the output writers, you can code the UNIT parameter on the DD statement defining the data set and specify the device on which you want the data set written. The system will allocate the device exclusively to your job if the device is available: no other job can write output to that device until it is released -- jobs cannot share an output device as they can when you assign output to output classes.

Data management routines write the output from the program to the device specified in the UNIT parameter.

If you code both the UNIT parameter and the SYSOUT parameter when defining a data set, the UNIT parameter is ignored.

Specifying a particular output device in the UNIT parameter is the least efficient method to route system output.

Suppressing the Writing of an Output Data Set

Whether you route an output data set by coding the SYSOUT parameter or the UNIT parameter, you can suppress the writing of the data set by defining it as a dummy data set. This is useful when you are testing a program and do not want data sets printed until you are sure they will contain meaningful output. Suppressing the writing of a data set saves processing time.

If you are routing an output data set by coding the SYSOUT parameter, code the DUMMY parameter to define the data set as a dummy data set. When DUMMY is coded, the SYSOUT parameter is ignored and the data set is not written.

If you are specifying the device on which the data set will be written in the UNIT parameter, you can assign the data set a dummy status by coding DUMMY or by assigning the data set name NULLFILE. All parameters other than DUMMY or DSNAME=NULLFILE and DCB are ignored; no units are assigned to the data set. When your program requests that the data set be written, the request is recognized but no data is transmitted. Your program must use the basic sequential access method (BSAM) or queued sequential access method (QSAM) when requesting to write a dummy data set; if any other access method is used, the job is terminated.

For details on coding the DUMMY parameter or DSNAME=NULLFILE, see the chapter "Defining a Dummy Data Set."

Requesting Multiple Copies of an Output Data Set (VS1 only)

With VS1 you can control the number of hard copies produced by the printer, punch, or tape. You can obtain as many as 255 copies of an output data set by:

- coding the COPIES parameter on the DD statement defining the data set, or
- requesting that the operator specify the desired number of copies in the REPEAT parameter of the WRITER command.

Requesting Multiple Copies with the COPIES Parameter

When you assign a data set to an output class in the SYSOUT parameter (see the chapter "Writing Output Data Sets"), you can also code the COPIES parameter and request as many as 255 copies of the data set:

```
//RECORD    DD  SYSOUT=W,COPIES=32
```

In the above example, you are requesting 32 copies of the data set. If you omit the COPIES parameter, a default value of 1 is assumed.

The job is cancelled if you code COPIES on a DD statement that does not include the SYSOUT parameter or if you specify a value less than 1 or greater than 255.

Requesting Multiple Copies with the WRITER Command

You can obtain multiple copies of an output data set by requesting that the operator specify the number of copies in the REPEAT parameter of the WRITER command. For example, if the operator specifies REPEAT=2, two copies of the data set will be printed. The command must be issued while the writer is processing the data set.

If you want multiple copies of **all** the output data sets in one class for a job, the operator can specify both the number of copies desired and the subparameter JOB in the REPEAT parameter of the WRITER command; for example:

```
REPEAT=(3,JOB)
```

When JOB is specified, the number indicates you want **additional** copies of the data sets; in this example, four copies of each data set will be printed all together. The command must be issued while the writer is processing the job's output.

A maximum of 255 copies can be specified. If multiple copies are requested in both the COPIES parameter on the DD statement and in the WRITER command, the number of copies specified in the command overrides the number specified on the DD statement.

Printer Forms and Print Chain Control

When requesting that an output data set be printed, you can give the system special instructions on how to print the data set; you can request:

- a special output form;
- a special character set, when output is being printed by a 3211 or 1403 printer with the universal character set;
- a specific image, which controls how many lines per inch are printed and the length of the form, when the data set is written to a 3211 printer.

Requesting a Special Output Form

To request that an output data set be printed on a special form, include the form number in the SYSOUT parameter on the DD statement defining the data set. For example, if you assign the data set OUTPUT to output class W which routes the data set to a printer and you want OUTPUT printed on form 1014, code:

```
//OUTPUT    DD  SYSOUT=(W,,1014)
```

The system will issue a message to the operator instructing him to supply form 1014 to the printer.

Requesting A Special Character Set

The universal character set (UCS) feature allows for different sets of characters to be printed for commercial and scientific applications. This feature can be requested for 3211 and 1403 printers during system generation.

To request a special character set for a 3211 or 1403 printer, specify the code identifying the character set in the UCS parameter. The codes for the IBM standard special character sets are:

1403	**3211**	**Characteristics**
AN	A11	Arrangement A, standard EBCDIC character set, 48 characters
HN	H11	Arrangement H, EBCDIC character set for FORTRAN and COBOL, 48 characters
	G11	ASCII character set
PCAN		Preferred alphameric character set, arrangement A
PCHN		Preferred alphameric character set, arrangement H
PN	P11	PL/I alphameric character set
QN		PL/I preferred alphameric character set for scientific applications
RN		Preferred character set for commercial applications of FORTRAN and COBOL
SN		Preferred character set for text printing
TN	T11	Character set for text printing, 120 characters
XN		High-speed alphameric character set for 1403, Model 2
YN		High-speed preferred alphameric character set for 1403, Model 3 or N1

Not all of these character sets may be available at your installation. In addition, your installation can design character sets to meet special needs; these character sets are assigned a unique code by the installation.

The operator is responsible for mounting the chain or train corresponding to the character set you request. If you do not code the UCS parameter, the operator supplies a default.

For both the 3211 and 1403 printers, you can code the UCS parameter with the UNIT parameter; for example:

```
//OUTPUT        DD      UNIT=1403,UCS=TN
```

With the 3211 printer, you can also code the UCS parameter with the SYSOUT parameter:

```
//OUTPUT        DD      SYSOUT=C,UCS=A11
```

Requesting the Fold Option

You request the fold option when uppercase and lowercase data is to be printed in uppercase only. To request folding, code FOLD following the character set you request:

```
UCS=(T11,FOLD)
```

You must specify a character set code when you request folding.

Requesting Operator Verification

You can request that the operator visually verify that the character set image corresponds to the graphics of the chain or train mounted on the printer. The character set image is displayed on the printer before the data set is printed. Code VERIFY as the last subparameter in the UCS parameter:

```
UCS=(A11,,VERIFY)
```

You must specify a character set code when you request operator verification.

Requesting a Specific Image

You request a specific image for a 3211 printer by coding an image identifier in the FCB parameter. The image identifier is a code that identifies the image to be loaded into the **forms control buffer** (FCB). (The forms control buffer is a buffer containing 180 positions that is used to store vertical formatting information; each position corresponds to a line on the form. The FCB is part of the 3811 control unit, which serves as an interface between the system and a 3211 printer.) The image identifier is retrieved from SYS1.IMAGELIB or defined in your program by specifying the address of a forms control buffer through the exit list facility of the DCB macro instruction. (This facility is described in **OS/VS Data Management Services Guide,** GC26-3783.)

IBM provides two standard FCB images, STD1 and STD2. STD1 specifies that 6 lines per inch are to be printed on an 8.5 inch form. STD2 specifies that 6 lines per inch are to be printed on an 11 inch form.

Your installation can provide additional images. If you omit the FCB parameter and the data set is written to a 3211 printer, the default image is used if it is currently in the buffer. Otherwise, the operator is requested to specify an image.

Requesting Alignment of Forms

To request the operator to check the alignment of the printer forms before the data set is printed, code ALIGN as the second subparameter of the FCB parameter:

```
//OUTPUT        DD      UNIT=3211,FCB=(STD1,ALIGN)
```

Requesting Operator Verification

You can request that the operator visually verify that the image displayed on the printer is the desired image by coding VERIFY as the second subparameter of the FCB parameter. Coding VERIFY also gives the operator an opportunity to align the forms --you cannot code ALIGN if you code VERIFY. For example:

```
//OUTPUT        DD      SYSOUT=A,FCB=(STD2,VERIFY)
```

If output class A routes output to a 3211 printer, the system loads the image identified by STD2 into the forms control buffer. If output class A does not correspond to a 3211 printer, the FCB parameter is ignored.

Controlling Ouptut to a Workstation (VS1 only)

In VS1, **remote entry services** (RES) provide the facility to submit jobs to a central computing center from a workstation and to route output to workstations. This avoids the inefficient procedure of putting the jobs from a workstation together, submitting them to the central computing center, having the operator there enter the jobs, and waiting for the output. Jobs are entered directly from the workstation, and output from jobs is printed or punched on remote devices, routed directly from the central computing center to the workstation.

When you submit a job from a workstation, the output is automatically returned to your workstation: you simply assign output data sets to an output class in the SYSOUT parameter and messages from your job to an output class in the MSGCLASS parameter. The system output writer offers many of the same options for writing data sets that you can request when submitting the job at the central computing center. You can request:

- that a data set be held until the operator requests that it be printed; see "Delaying the Writing of an Output Data Set" in the chapter "Writing Output Data Sets."
- a special output form by specifying a form number in the SYSOUT parameter --see the chapter "Printer Forms and Print Chain Control."
- multiple copies of the data set --see the chapter "Requesting Multiple Copies of an Output Data Set."

RES also provides an additional option: whether you are at a remote station or at the central computing center, you can request that a data set be routed to another destination. To do this, you code the DEST parameter, as described in this chapter.

Routing Output to Another Destination

Users at workstations are grouped into **destinations.** Each destination is identified by a user identification of 1-7 alphameric characters, established by the system programmer. (The central computing center is identified by the user identification CENTRAL.) A workstation can be identified by a single identification --for example, a branch office of a bank; or by several identifications --for example, a separate identification for each of the departments at a college.

Each user identification is associated with an authorization value, also established by the system programmer, that controls to whom it can send output. A destination can send output only to other destinations with an authorization value equal to or greater than its own. This provides a means to control the output a destination can receive. For example, a shipping department is assigned an authorization value equivalent to 6 (the value is actually expressed in hexadecimal); destinations that can send output (orders, inventories, etc.) to the shipping department are assigned authorization values lower than or equal to 6; destinations that cannot send output to the shipping department are assigned authorization values greater than 6. Every destination, however, can send output to CENTRAL.

To route an output data set to another destination, code the identification of that destination in the DEST parameter on the DD statement defining the data set:

```
//RECORDS         DD       SYSOUT=A,DEST=LOC5
```

If you do not code the DEST parameter, or code an invalid identification, the output is automatically returned to you. If you are not authorized to send output to the specified destination, the output is returned to you with a warning message.

You can notify another destination of the output it will receive by issuing the SEND command, as described under "Sending Messages to Other Destinations."

Sending Messages to Other Destinations

The SEND command allows users to send messages to one another and to the central operator. At a workstation, the operator is usually the only one who would enter the SEND command from the console. However, an applications programmer can code the SEND command on a command statement to be included in the input stream. This discussion will address the SEND command only in the following situations:

- you wish to notify another destination that you have routed a data set to it.
- you wish to notify another destination that a data set routed to it is on the hold queue.

Not every option of the SEND command is included here; for a complete description, see **OS/VS1 RES: Workstation User's Guide**, GC28-6879.

To issue the SEND command in one of the above situations, code:

- SEND or SE, to identify the command.
- the text of the message enclosed in apostrophes, limited to 115 characters, including blanks.
- the destination to which the message is directed, USER=userid. (Multiple destinations can be specified, if they are enclosed in parentheses.)
- when the message is to be sent. If you do not indicate this, the system assumes NOW --the message is sent as soon as the command statement is processed; if the destination receiving the message is not logged-on, the message is not sent and a diagnostic message is returned to you. Usually you will want to specify LOGON -- the message is sent immediately, if the receiving destination is logged-on; if the destination is not logged-on, the message is saved and sent when the destination next establishes connection with the central system.

The command statement can be included in the input stream, either within a job (before an EXEC statement, a null statement, or another command statement) or between jobs. However, if the command is placed between jobs and is coded incorrectly, the command will not be executed and no message will be sent: it is safest, then, to include a command statement within a job. (Complete details on coding a command statement are included in the **OS/VS JCL Reference**, GC28-0618.)

For example, you are routing a data set containing bank records to the destination identified as DEPT58 and are placing the data set on the hold queue:

```
//RECORDS        DD       SYSOUT=A,DEST=DEPT58,HOLD=YES
```

This DD statement is included in the job named COMPUTE --when a data set is placed on the hold queue, it is identified by the name of the job which produced it.

You want to send a message to DEPT58, notifying them that the data set is on the hold queue:

```
//  SEND  'data set from job COMPUTE on hold queue',
//  USER=DEPT58,LOGON
```

To remove this data set from the hold queue, the operator at DEPT58 issues a ROUTE command, specifying the name of the job which produced the output (in this case COMPUTE) and HOLD=NO.

Cataloged and In-Stream Procedures

Applications that require many control statements and are used on a regular basis can be considerably simplified through the use of cataloged and in-stream procedures. A **cataloged procedure** is a set of job control statements that are placed in a partitioned data set known as the procedure library; an **in-stream procedure** is a set of job control statements that are placed in the input stream within a job. You can execute a procedure simply by specifying its name on an EXEC statement in your job.

This section describes how to write and use cataloged and in-stream procedures; it is divided into three chapters:

- Writing Cataloged and In-Stream Procedures
- Using Cataloged and In-Stream Procedures
- Using Symbolic Parameters

Writing Cataloged and In-Stream Procedures

Cataloged and in-stream procedures are simply the job control statements needed to perform an application. A procedure contains one or more **procedure steps**, each step consisting of an EXEC statement that identifies the program to be executed and DD statements defining the data sets to be used or produced by the program. The program you request on the EXEC statement must exist in a private or the system library. If you do request a program that is contained in a private library, the procedure step calling that program must include a DD statement with the ddname STEPLIB that defines the private library; the STEPLIB DD statement is described in the chapter, "Creating and Using Private and Temporary Libraries."

Cataloged and in-stream procedures cannot contain:

- EXEC statements that refer to other cataloged or in-stream procedures;
- JOB, delimiter, or null statements;
- DD statements defining private libraries to be used throughout the job (DD statements with the ddname JOBLIB);
- DD statements defining data in the input stream (statements including the * or DATA parameters).

Identifying an In-stream Procedure

To identify an in-stream procedure, you code the PROC and PEND job control statements.

On the PROC statement, which must be the first statement in an in-stream procedure, you assign the procedure a name. This name is the name that a programmer codes to call the procedure. Optionally, you can also assign default values to symbolic parameters contained in the procedure and code comments. (A symbolic parameter is a symbol preceded by an ampersand that stands for a parameter, a subparameter, or a value in a procedure; including symbolic parameters in a procedure is described in detail in the chapter "Using Symbolic Parameters.") If you do not assign default values to symbolic parameters on the PROC statement, you cannot code comments. The simplest form of the PROC statement, to identify an in-stream procedure named PAYROLL, would be:

```
//PAYROLL       PROC
```

The PEND statement marks the end of the in-stream procedure. You can include a name on the PEND statement and comments, but these are optional. Both of the following examples are acceptable:

```
//ENDPROC       PEND         end of in-stream procedure
//              PEND
```

The following example illustrates an in-stream procedure named SALES consisting of two procedure steps. Note that STEP2 includes a STEPLIB DD statement to define the private library in which the program JUGGLE can be found.

```
//SALES      PROC
//STEP1      EXEC         PGM=FETCH
//DD1A       DD           DSNAME=RECORDS(BRANCHES),DISP=OLD
//DD1B       DD           DSNAME=RECORDS(MORGUE),DISP=MOD
//STEP2      EXEC         PGM=JUGGLE
//STEPLIB    DD           DSNAME=PRIV.WORK,DISP=OLD
//DD2A       DD           SYSOUT=A
//           PEND
```

Placing a Cataloged Procedure in a Procedure Library

The major difference between cataloged an in-stream procedures is that, whereas in-stream procedures are placed within the job that calls them, cataloged procedures must be placed in a procedure library before they can be used. A **procedure library** is simply a partitioned data set containing cataloged procedures. IBM supplies a procedure library named SYS1.PROCLIB, but your installation can have additional procedure libraries with different names. When a programmer calls a cataloged procedure, he receives a copy of the procedure; therefore, a cataloged procedure can be used by more than one programmer simultaneously.

To add a procedure to a procedure library, you use the IEBUPDTE utility program. You can also use the IEBUPDTE utility to permanently modify an existing procedure. (Before modifying an existing cataloged procedure, however, you must notify the operator; he must delay the execution of jobs that might use the procedure library while it is being updated.) Details on using the IEBUPDTE utility are included in **OS/VS Utilities**, GC35-0005. Before placing a cataloged procedure in a procedure library, you can test it by first running it as an in-stream procedure.

No special job control statements are used to identify a cataloged procedure. The PEND statement is never used and the PROC statement is optional. You need code the PROC statement as the first statement in a cataloged procedure only when you want to assign default values to symbolic parameters. The name of the PROC statement is not necessarily the name of the cataloged procedure; you assign the procedure a name when you add it to the procedure library.

Allowing for Changes in Cataloged and In-stream Procedures

The usefulness of cataloged and in-stream procedures is destroyed if a programmer who uses the procedure has to permanently modify the procedure every time he wants to make a change. When you write a procedure, you can define, as symbolic parameters, those parameters, subparameters and values that are likely to vary each time the procedure is used. For details on coding symbolic parameters, see the chapter "Using Symbolic Parameters."

Using Cataloged and In-Stream Procedures

To use a cataloged or in-stream procedure, you specify the procedure name on an EXEC statement; you can modify the procedure by adding DD statements, overriding, adding, or nullifying parameters on EXEC and DD statements, and assigning values to symbolic parameters. Calling and modifying procedures is explained in greater detail in the following paragraphs.

How to Call Cataloged and In-Stream Procedures

To call a cataloged or in-stream procedure, you identify the procedure on the EXEC statement of the step calling the procedure by coding:

- the procedure name; or
- PROC= the procedure name,

as the first operand on the EXEC statement.

A cataloged procedure must exist in the procedure library before you attempt to use it --if the cataloged procedure exists in SYS1.PROCLIB, the job that adds the procedure to the library must terminate before the job that calls it is selected for execution. The input stream reader is responsible for fetching cataloged procedures you call that exist in a private procedure library (as opposed to the IBM-supplied procedure library, SYS1.PROCLIB). Therefore, a cataloged procedure in a private procedure library must exist before a job that calls it is read into the system. When using an in-stream procedure, include the procedure, beginning with a PROC statement and ending with a PEND statement, with the job control language for your job; the procedure must follow the JOB statement but appear before the EXEC statement that calls it. You can include as many as fifteen uniquely-named in-stream procedures in one job and can use each procedure as many times as you wish in the job.

To call a cataloged procedure named PROCESSA, you would code:

```
//CALL      EXEC      PROCESSA        or
//CALL      EXEC      PROC=PROCESSA
```

On the EXEC statement, you can also code changes you would like to make for this execution of the procedure.

Modifying Cataloged and In-Stream Procedures

You can modify a procedure by:

- assigning values to or nullifying symbolic parameters contained in the procedure;
- overriding, adding, or nullifying parameters on EXEC and DD statements in the procedure;
- adding DD statements to the procedure.

All changes you make are in effect only during the current execution of the procedure. For a discussion of symbolic parameters, see the chapter "Using Symbolic Parameters." Other modifications are described in the following sections.

Modifying Parameters on an EXEC Statement

To override, add, or nullify a parameter on an EXEC statement in a procedure, identify on the EXEC statement that calls the procedure the parameter you are changing, the name of the EXEC statement on which the parameter appears, and the change to be made:

```
//CALL  EXEC    procedurename,parameter.procstepname=value
```

When **overriding** a parameter, the value you code for the parameter on the EXEC statement calling the procedure replaces the value assigned in the procedure. When **adding** a parameter, that parameter is used in the execution of the procedure step. When **nullifying** a parameter, you do not follow the equal sign with a value; the value assigned to the parameter in the procedure is ignored. All changes you make are in effect only for the current execution of the procedure.

You can make more than one change to each EXEC statement in the procedure, and you can change parameters on more than one EXEC statement in the procedure. You cannot, however, change the PGM parameter. When making changes on different steps in the procedure, you must code all changes for one procedure step before changes to a subsequent step.

For example, the first three EXEC statements in a procedure named IRISH are:

```
//STEP1      EXEC    PGM=YEATS,PARM='*14863'
//STEP2      EXEC    PGM=NOLAN
//STEP3      EXEC    PGM=SYNGE,TIME=(2,30)
```

and you want to make the following changes:

- nullify the PARM parameter in STEP1
- add the COND parameter, specifying the test (8,LT), in STEP2
- change the time limit in the TIME parameter in STEP3 to 4 minutes.

On the EXEC statement calling the procedure, you would code:

```
//CALL  EXEC    IRISH,PARM.STEP1=,
//              COND.STEP2=(8,LT),TIME.STEP3=4
```

You can omit naming the procedure step when you change a parameter. When you do this, the procedure is modified as follows:

- If the PARM parameter is coded, it applies only to the first procedure step. If a PARM parameter appears in a later EXEC statement in the called procedure, it is nullified.
- If the TIME parameter is coded, it applies to the total procedure. If the TIME parameter appears on any of the EXEC statements in the called procedure, it is nullified.
- If any other parameter is coded, it applies to every step in the called procedure. Nullifying the parameter on the EXEC statement calling the procedure causes that parameter to be ignored on every EXEC statement in the procedure; if you assign a value to the parameter on the EXEC statement calling the procedure, the parameter is overridden where it appears in the procedure and added to EXEC statements in the procedure on which it does not appear.

For example, assume the EXEC statements in a procedure named COMPUTE are:

```
//STEP1      EXEC     PGM=LIST,TIME=(1,30)
//STEP2      EXEC     PGM=UPDATE,RD=NC,TIME=2
//STEP3      EXEC     PGM=CHECK,RD=RNC,COND=ONLY
```

You want to make the following changes:

1. assign a time limit of 4 minutes to the entire procedure; TIME parameters on individual EXEC statements in the procedure will be nullified.
2. allow automatic step restart for each step of the job by coding RD=R. The RD parameter will be added to the first step of the job and will override the RD parameters in STEP2 and STEP3.

To call the procedure and make these changes, you would code:

```
//CALL       EXEC     COMPUTE,TIME=4,RD=R
```

During the execution of the procedure, the EXEC statements appear as:

```
//STEP1      EXEC     PGM=LIST,RD=R
//STEP2      EXEC     PGM=UPDATE,RD=R
//STEP3      EXEC     PGM=CHECK,RD=R,COND=ONLY
```

Any parameter changes that affect every step of the job (by omitting the procedure step name) must be coded on the EXEC statement calling the procedure before changes to parameters on different steps (**i.e.**, you include the procedure step name).

Modifying Parameters on a DD Statement

To override, add, and nullify parameters on a DD statement in a procedure, you include a DD statement containing the changes you want to make after the EXEC statement that calls the procedure. The name of the DD statement containing the changes is composed of the procedure step name and the ddname of the DD statement in the procedure:

```
//procstepname.ddname   DD  parameter=value
```

When **overriding** a parameter, the value you code replaces the value assigned to the parameter in the procedure. You can also override a parameter in the procedure by coding a mutually exclusive parameter on the DD statement containing the changes. (A table of mutually exclusive parameters on the DD statement is included in the **OS/VS JCL Reference**, GC28-0618.) When **adding** a parameter, the parameter is added to the DD statement in the procedure for the current execution of the procedure. When **nullifying** a parameter, you do not follow the equal sign with a value; that parameter in the procedure is ignored. You do not have to nullify a parameter when you are replacing it with a mutually exclusive parameter. All changes you make are in effect only for the current execution of the procedure.

You can change more than one parameter on a DD statement and you can change parameters on more than one DD statement in the procedure. However, the DD statements containing the changes must be coded in the same order as the corresponding DD statements in the procedure.

For example, the first two steps of the cataloged procedure TEA are:

```
//STEP1 EXEC    PGM=SUGAR
//DD1A  DD      DSNAME=DRINK,DISP=(NEW,DELETE),
//              UNIT=2400,VOL=SER=568998
//DD1B  DD      UNIT=SYSSQ
//STEP2 EXEC    PGM=LEMON
//DD2A  DD      UNIT=2314,DISP=(,PASS),
//              SPACE=(TRK,(20,2))
```

You want to make the following changes:

1. Change the disposition on the DD statement named DD1A to CATLG.
2. Change the unit on the DD statement named DD1B to TAPE.
3. Change the SPACE parameter on the DD statement named DD2A to SPACE=(CYL,(4,1)).

When calling the procedure, you would code:

```
//CALL              EXEC     TEA
//STEP1.DD1A        DD       DISP=(NEW,CATLG)
//STEP1.DD1B        DD       UNIT=TAPE
//STEP2.DD2A        DD       SPACE=(CYL,(4,1))
```

When changing DCB subparameters, you need code only those subparameters you are changing. The DCB subparameters you do not code (and for which you do not code a mutually exclusive subparameter) remain unchanged. For example, a DD statement named DD1 in a procedure step named STEP1 contains DCB=(BUFNO=1,BLKSIZE=80,RECFM=F,BUFL=80). To change the block size to 320 and the buffer length to 320, you would code:

```
//STEP1.DD1 DD  DCB=(BLKSIZE=320,BUFL=320)
```

The subparameters BUFNO and RECFM remain unchanged.

To nullify the DCB parameter, you must nullify each subparameter. For example, if a DD statement in a procedure contains DCB=(RECFM=FB,BLKSIZE=160,LRECL=80), you must code DCB=(RECFM=,BLKSIZE=,LRECL=) in order to nullify the DCB parameter.

To nullify the DUMMY parameter, code the DSNAME parameter on the overriding DD statement and assign a data set name other than NULLFILE. To nullify all the parameters on a DD statement other than DCB, code DUMMY on the overriding DD statement. (The DUMMY parameter is described in detail in the chapter, "Defining a Dummy Data Set.")

Modifying Parameters on DD Statements that Define Concatenated Data Sets

When a concatenation of data sets is defined in a cataloged procedure and you attempt to override the concatenation with one DD statement, only the first (named) DD statement is overridden. To override others, you must include an overriding DD statement for each DD statement; the DD statements in the input stream must be in the same order as the DD statements in the procedure. The second and subsequent overriding statements must not be named. If you do not wish to change one of the concatenated DD statements, leave the operand field blank on the corresponding DD statement in the input stream. (This is the only case where a blank operand field for a DD statement is valid.)

For example, suppose you are calling a procedure that includes the following sequence of DD statements in STEPC:

```
//DD4    DD   DSNAME=A.B.C,DISP=OLD
//       DD   DSNAME=STRP,DISP=OLD,UNIT=2314,VOL=SER=X12182
//       DD   DSNAME=TYPE3,DISP=OLD,UNIT=2314,VOLUME=SER=BL1421
//       DD   DSNAME=A.B.D,DISP=OLD
```

To override the DD statements that define the data sets named STRP and A.B.D, you would code:

```
//STEPC.DD4 DD
//          DD   DSNAME=INV.CLS,DISP=OLD
//          DD
//          DD   DSNAME=PAL8,DISP=OLD,UNIT=2314,VOL=SER=125688
```

Adding DD Statements to a Procedure

You can add DD statements to a procedure when you call the procedure. These additional DD statements are in effect only during the current execution of the procedure.

To add a DD statement to a procedure step, follow the EXEC statement that calls the procedure and any overriding DD statements for that step with the additional DD statement. The ddname of the DD statement identifies the procedure step to which this statement is to be added and must be assigned a name that is different from all the ddnames in the procedure step. If you do not identify the procedure step in the ddname, the system assumes you are adding the DD statement to the first step of the procedure.

For example, the first step of a cataloged procedure named MART is:

```
//STEP1 EXEC    PGM=DATE
//DDM   DD      DSNAME=BPS(MEMG),DISP=OLD,
//              UNIT=2314,VOLUME=SER=554982
//DDN   DD      UNIT=SYSSQ
```

You want to make the following changes:

1. Change the UNIT parameter on the statement named DDN to UNIT=180.

2. Add a DD statement, specifying UNIT=181.

When calling the procedure, you would code:

```
//PROC          EXEC    MART
//STEP1.DDN     DD      UNIT=180
//STEP1.DDO     DD      UNIT=181
```

Identifying Procedure Statements on an Output Listing

You can request that cataloged and in-stream procedure statements be included on the output listing by coding 1 as the first subparameter in the MSGLEVEL parameter on the JOB statement. (For a description of the MSGLEVEL parameter, see "Controlling the Output Listing of JCL Statements, Messages, and Dumps.") Procedure statements are identified on the output listing as illustrated in Figures 4 and 5. The output listing will also show the symbolic parameters and the values assigned to them.

Columns 123	
XX	cataloged procedure statement you did not override
X/	cataloged procedure statement you did override
XX*	cataloged procedure statement, other than a comment statement, that the system considers to contain only comments
***	comment statement

Figure 4. Identification of Cataloged Procedure Statements on the Output Listing

Columns 123	
++	in-stream procedure statement you did not override
+//	in-stream procedure statement you did override
++*	in-stream procedure statement, other than a comment statement, that the system considers to contain only comments
***	comment statement

Figure 5. Identification of In-stream Procedure Statements on the Output Listing

In order to be modified easily, cataloged and in-stream procedures can contain **symbolic parameters**. A symbolic parameter is a symbol preceded by an ampersand that stands for a parameter, a subparameter, or a value. In the following procedure step, the symbolic parameters are underlined:

```
//STEP1 EXEC     PGM=UPDATE,ACCT=(PGMG,&DEPT)
//DD1   DD       DSNAME=INIT,UNIT=&DEVICE,SPACE=(CYL,(&SPACE,10))
//DD2   DD       DSNAME=CHNG,UNIT=2400,DCB=BLKSIZE=&LENGTH
```

When this procedure is executed, every symbolic parameter must either be assigned a value or nullified; the changes are in effect only for the current execution of the procedure. Therefore, the procedure can be modified each time it is executed, without being permanently changed. Details on how to assign values to or nullify symbolic parameters are included under "Assigning Values to and Nullifying Symbolic Parameters." How to include symbolic parameters when writing a cataloged or in-stream procedure is described in the next section, "Defining Symbolic Parameters When Writing a Procedure."

Defining Symbolic Parameters When Writing a Procedure

Any parameter, subparameter, or value in a procedure that may vary each time the procedure is called is a good candidate for definition as a symbolic parameter. For example, if different values can be passed to a processing program by means of the PARM parameter on one of the EXEC statements, you could define the PARM field as one or more symbolic parameters, **e.g.,** PARM=&ALLVALS or PARM=&DECK&CODE.

The symbolic parameter itself is one to seven alphameric and national (#,@,$) characters preceded by a single ampersand. The first character must be alphabetic or national. Since a single ampersand defines a symbolic parameter, you code double ampersands when you are not defining a symbolic parameter. For example, if you want to pass 543&LEV to a processing program by means of the PARM parameter, you must code PARM='543&&LEV'. The system treats the double ampersand as if a single ampersand had been coded, and only one ampersand appears in the results.

Parameters coded on the EXEC statement cannot be defined as symbolic parameters, although you can define subparameters of the parameters symbolically. For example, you must **not** code &ACCT; however, you can code ACCT=(43877,&DEPT).

The definitions used to signify symbolic parameters should be consistent in all the cataloged and in-stream procedures at an installation. For example, every time the programmer is to assign his department number to a symbolic parameter, no matter which procedure he is calling, the symbolic parameter could be defined as &DEPT. In different procedures, you could code ACCT=(43877,&DEPT) and DSNAME=LIBRARY.&DEPT.TALLY. The programmer would assign his department number to the symbolic parameter wherever that symbolic parameter appears in a procedure.

The same symbolic parameter can appear more than once in a procedure, as long as the value assigned to the symbolic parameter is a constant in the procedure. Therefore, you could use &DEPT more than once in a procedure, if the department number to be assigned is the same in each case. But if you have two DD statements and include a symbolic parameter for the primary quantity of the space request on each DD statement, you would not want to use the same symbolic parameter, since the requests for primary quantity could be different for the two data sets. Only one value can be assigned to each symbolic parameter used in a procedure;

if you assign more than one value to a symbolic parameter, only the first value is used and that value is substituted wherever the symbolic parameter occurs.

Caution Concerning Leading and Trailing Commas

All symbolic parameters must be assigned values or nullified before the procedure is executed. (When you write a procedure, you can assign default values to the symbolic parameters, or the programmer can assign values when he calls the procedure; for details, see "Assigning Values to and Nullifying Symbolic Parameters.") When a symbolic parameter is nullified, a delimiter, such as a leading or trailing comma, is not automatically removed. Only when the symbolic parameter is a positional **sub**parameter followed by other subparameters should the comma remain. In other cases, the remaining comma will cause a syntax error.

For example, you code for a unit request:

```
UNIT=(2314,&MORE,DEFER)
```

If &MORE is nullified, the comma before it must remain, since the unit count subparameter is positional and a comma must indicate its absence if other subparameters follow. When &MORE is nullified, the parameter will appear as:

```
UNIT=(2314,,DEFER)
```

However, if you code:

```
VOLUME=SER=(111111,&SERNO)
```

and &SERNO is nullified, a leading comma will remain and cause a JCL syntax error. If a symbolic parameter is a positional parameter followed by other parameters in the statement, such as

```
//DEFINE        DD       &POSPARM,DSN=ATLAS,DISP=OLD
```

the comma will remain at the beginning of the operand field if &POSPARM is nullified and again cause a syntax error.

In these cases, you should not code the comma. When a symbolic parameter **follows** information that does not vary, such as in VOLUME=SER=(111111,&SERNO), you do not have to code any delimiter. The system recognizes the symbolic parameter when it encounters the single ampersand. For this example, you would code:

```
VOLUME=SER=(111111&SERNO)
```

When a value is assigned to the symbolic parameter, a comma must be included in the value, i.e., SERNO=',222222'. (Since the comma is a special character, the value is enclosed in single apostrophes. For rules on when special characters must be enclosed in apostrophes, see the **OS/VS JCL Reference**, GC28-0618.)

When a symbolic parameter **precedes** information that does not vary, a period may be required after the symbolic parameter to distinguish the end of the symbolic parameter from the beginning of the information that does not vary. A period is required after the symbolic parameter when the character following the symbolic parameter is:

- an alphabetic, numeric, or national (#,@,$) character;
- a left parenthesis or a period.

The system recognizes the period as a delimiter and the period does not appear in the procedure after the symbolic parameter is assigned a value or nullified. (A period will appear after the value when two consecutive periods are coded.)

Therefore, you should place a period after a symbolic parameter that stands for a positional parameter followed by other parameters in the statement:

```
//DEFINE         DD       &POSPARM.DSN=ATLAS,DISP=OLD
```

If &POSPARM is nullified, the statement appears as:

```
//DEFINE         DD       DSN=ATLAS,DISP=OLD
```

When you assign a value to &POSPARM, you must include a comma:

```
POSPARM='DUMMY,'
```

These rules are in effect whenever you concatenate a symbolic parameter with information that does not vary. For example, placing a symbolic parameter **after** information that does not vary:

- DSNAME=LIBRARY(&MEMBER)
- DSNAME=USERLIB.&LEVEL

In these examples, the system recognizes the symbolic parameter when it encounters the & .

And placing a symbolic parameter **before** information that does not vary:

- PARM='&OPTION+15'

  &OPTION is not followed by period because of the +.
- DSNAME=&QUAL.246

  The period is required because a numeric character follows the symbolic parameter.
- DSNAME=&LIBRARY.(MEMG)

  The period is required because a left parenthesis follows the symbolic parameter.
- DSNAME=&DOCNO..TXT

  The period is required because a period follows the symbolic parameter. A single period will appear in the results.

You can also define two or more symbolic parameters in succession without including a comma, for example, PARM=&DECK&CODE. If a comma is desired in the results, a comma must then be included in the value assigned to the symbolic parameter.

Assigning Default Values to Symbolic Parameters

You can assign default values to the symbolic parameters coded in the procedure on the PROC statement. The PROC statement must always appear as the first statement in an in-stream procedure; the PROC statement must be coded as the first statement in a cataloged procedure only if you want to assign defaults. Generally, you should assign defaults to every symbolic parameter in a procedure to limit the amount of coding necessary each time the procedure is called. See the next section, "Assigning Values to and Nullifying Symbolic Parameters", for details.

Assigning Values to and Nullifying Symbolic Parameters

When a procedure containing symbolic aprameters is used, each symbolic parameter must either be assigned a value or nullified. Symbolic parameters are assigned values or nullified in one of two ways:

- the programmer who uses the procedure codes the symbolic parameter on the EXEC statement calling the procedure, either assignint it a value or nullifying it;
- the programmer who writes the procedure assigns a default value to or nullifies the symbolic aprameter on the PROC statement, which must be the first statement in an in-stream procedure and can be the first statement in a cataloged procedure.

The default assigned to a sybmolic parameter on a PROC statement is overridden when that symbolic parameter is assigned a value or nullified on the EXEC statement that calls the procedure.

Default values are not necessarily assigned to symbolic parameters in a procedure. Before using any procedure, you must find out what symbolic parameters are used, the meaning of each symbolic parameter, and what default, if any, is assigned. The PROC statement is optional in cataloged procedures; if the PROC statement is not included, no default values can be assigned to symbolic parameters in the procedure.

You need not code the symbolic parameters in any specific order when you assign values to or nullify them.

Assigning a Value to a Symbolic Parameter

To assign a value to symbolic parameter, you code:

```
symbolic parameter=value
```

Omit the ampersand that precedes the symbolic parameter in the procedure. For example, if the symbolic parameter &NUMBER appears on a DD statement in the procedure, code NUMBER=value on the PROC statement (if you are writing the procedure and assigning defaults) or on the EXEC statement that calls the procedure (if you are using the procedure and want this value to be in effect only for the current execution of the procedure).

There are some rules for assigning values to symbolic parameters:

1. The length of the value you assign is limited only in that the value cannot be continued onto another statement. However, when a symbolic parameter is concatenated with other information (for example, a data set name is LIBRARY.&DEPT..MACS), the combined length of the value you assign and the concatenated information cannot exceed 120 characters.
2. If the value contains special characters, enclose the value in apostrophes (the enclosing apostrophes are not considered part of the value). If the special characters include apostrophes, each must be shown as two consecutive apostrophes.
3. If more than one value is assigned to a symbolic parameter as a default on the PROC statement, only the first value encountered is used; likewise, if more than one value is assigned to a symbolic parameter on an EXEC statement, only the first value encountered is used.
4. If a symbolic parameter is a positional parameter followed by other parameters in the statement, it should be followed in the procedure by a period instead of a comma; for example:

```
//DEFINE        DD       &POSPARM.DSN=ATLAS,DISP=OLD
```

Symbolic parameters that are keyword subparameters should appear in the procedure without a preceding comma; for example:

```
VOLUME=SER=(111111&SERNO)
```

This is necessary so that, if the symbolic parameter is nullified, a leading or trailing comma will not cause a JCL syntax error. (For a more complete discussion of this, see "Caution Concerning Leading and Trailing Commas.")

In these cases, you must include a comma when you assign a value to the symbolic parameter, i.e.,

```
POSPARM='DUMMY,'
SERNO=',222222'
```

Since the comma is a special character, the value must then be enclosed in apostrophes.

Nullifying a Symbolic Parameter

To nullify a symbolic parameter, code:

```
symbolic parameter=
```

Omit the ampersand that precedes the symbolic parameter in the procedure and do not follow the equal sign with a value.

For example, a DD statement in an in-stream procedure named TIMES is:

```
//DD8        DD       UNIT=3211,UCS=&UCSINFO
```

If you are writing the procedure and want to nullify &UCSINFO as a default on the PROC statement, code:

```
//TIMES      PROC         UCSINFO=
```

If you are calling the procedure, and no default was assigned to &UCSINFO, or if &UCSINFO was assigned a value on the PROC statement, you would nullify the parameter on the EXEC statement that calls the procedure by coding:

```
//CALL       EXEC         TIMES,UCSINFO=
```

Example of a Procedure Containing Symbolic Parameters

The cataloged procedure named PAYROLL contains the following statements:

```
//       PROC    DEPT=D58,GROUP=PGMRA,DEVICE=2314,
//               VOLCNT=2,SERNO=,POSPARM='DUMMY,'
//STEP1 EXEC    PGM=GATHER
//DD1A  DD      DSNAME=FILE.&DEPT..CLASSA,DISP=OLD
//STEP2 EXEC    PGM=DEDUCT
//DD2A  DD      DSNAME=MEDICAL(&GROUP),DISP=MOD
//DD2B  DD      DSNAME=LIST,UNIT=&DEVICE,VOL=(,,&VOLCNT)
//STEP3 EXEC    PGM=COMPUTE
//DD3A  DD      DSNAME=MASTER,UNIT=2314,VOL=SER=(111111&SERNO)
//DD3B  DD      &POSPARM.SYSOUT=A
```

The PROC statement is included in order to assign defaults to the symbolic parameters in the procedure.

When using this procedure, you want to override the following symbolic parameters and assign these values:

&GROUP	CLERK
&VOLCNT	3
&SERNO	222222
&POSPARM	nullify

On the EXEC statement that calls the procedure, you would code:

```
//CALL   EXEC     PAYROLL,GROUP=CLERK,VOLCNT=3,SERNO=',222222',
//                POSPARM=
```

Glossary

The following terms are defined as they are used in this manual. If you do not find the term you are looking for, refer to the Index or to the IBM Data Processing Glossary, GC20-1699.

IBM is grateful to the American National Standards Institute (ANSI) for permission to reprint its definitions from the American National Standard Vocabulary for Information Processing (Copyright© 1970 by American National Standards Institute, Incorporated), which was prepared by Subcommittee X3K5 on Terminology and Glossary of American National Standards Committee X3. ANSI definitions are marked with an *

allocation/termination messages: messages produced by the system concerning allocation of resources, disposition of data sets, and termination of job steps and the job.

automatic priority group (APG): in VS2, a group of tasks at a single priority level that are dispatched according to a special algorithm that attempts to provide optimum use of CPU and I/O resources by these tasks.

automatic restart: a restart that takes place during the current run, that is, without resubmitting a job; an automatic restart can occur within a step or at the beginning of a step. Contrast with deferred restart.

backward reference: a facility of the job control language that permits you to copy information from or refer to DD statements that appear earlier in the job.

catalog: the collection of all data set indexes that are used by the control program to locate a volume containing a specific data set.

cataloged data set: a data set that is represented in an index or hierarchy of indexes in the system catalog; the indexes provide the means for locating the data set.

cataloged procedure: a set of job control statements that has been placed in a partitioned data set called the procedure library and that can be retrieved by coding the name of the procedure on an execute (EXEC) statement or started by a START command.

checkpoint data set: a sequential or partitioned data set containing a collection of records (called checkpoint entries) that contain the status of a job and the system at the time the records are written. These records provide the information necessary for restarting a job without having to return to the beginning of the job.

checkpoint restart: the process of resuming a job at a checkpoint within the job step that was abnormaly terminated. The restart can be automatic or deferred, where deferred restart involves resubmitting the job. Contrast with step restart.

checkpoint/restart facility: a facility for restarting execution of a program at some point other than at the beginning, after the program was terminated due to a program or system failure. A restart can begin at a checkpoint within a job step or at the beginning of a job step.

command statement: a job control statement that is used to issue commands to the system through the input stream.

comment statement: a job control statement used to include information that may be helpful in running a job or reviewing an output listing.

concatenated data sets: a group of logically connected data sets that are treated as one data set for the duration of a job step.

data definition (DD) statement: a job control statement that describes a data set associated with a particular job step.

data management: a major function of the operating system that involves organizing, cataloging, locating, storing, retrieving, and maintaining data.

data set: the major unit of data storage and retrieval in the operating system, consisting of a collection of data in one of several prescribed arrangements and described by control information to which the system has access.

dedicated data set: a data set assigned to an initiator that is allocated space when the initiator is started; every job step running under the initiator can use the dedicated data set as a temporary data set.

deferred restart: a restart performed by the system on resubmission of a job by the programmer; deferred restart can begin within a step or at the beginning of a step. Contrast with automatic restart.

delimiter statement: a job control statement used to mark the end of data.

direct system output (DSO) writer: in VS1, a job scheduler function that controls the writing of a job's output data sets directly to an output device during execution of the job.

dispatching priority: a number assigned to tasks, used to determine the order in which they will use the central processing unit.

disposition processing: a function performed by the initiator at the end of a job step to keep, delete, catalog, or uncatalog data sets, or pass them to a subsequent job step, depending on the data set status or the disposition specified in the DISP parameter of the DD statement.

dummy data set: a data set for which operations such as disposition processing, input/output operations, and allocation are bypassed.

***dump:** (1) to copy the contents of all or part of storage, usually from an internal storage into an external storage. (2) the data resulting from the process as in (1).

execute (EXEC) statement: a job control statement that marks the beginning of a job step and identifies the program to be executed or the cataloged or in-stream procedure to be used.

external page storage: the portion of auxiliary storage that is used to contain pages.

forms control buffer (FCB): a buffer containing 180 positions that is used to store vertical formatting information for printing, each position corresponding to a line on the form; the FCB is part of the 3811 control unit, which serves as an interface between the system and a 3211 printer.

generation data group (GDG): a collection of data sets that are kept in chronological order; each data set is called a generation data set.

generation data set: one generation of a generation data group.

group name: a generic name for a collection of I/O devices, for example, DISK or TAPE.

hold queue: a waiting list for jobs whose initiation is to be delayed until the operator releases them from the queue.

indexed sequential data set: a data set in which each record contains a key that determines its location. The location of each record is computed through the use of an index.

initiator: the job scheduler function that selects jobs and job steps to be executed, allocates input/output devices for them, places them under task control, and at completion of the job, supplies control information for writing job output on a system output unit.

initiator procedure: the cataloged procedure that controls an initiator.

input queue: a queue (waiting list) of job definitions on direct access storage arranged in order of assigned job class and assigned priority.

in-stream procedure: a set of job control statements placed in the input stream that can be used any number of times during a job by naming the procedure on an execute (EXEC) statement.

job: a collection of related problem programs, identified in the input stream by a JOB statement followed by one or more EXEC and DD statements.

job class: any one of a number of job categories that can be defined by the installation to classify jobs. By classifying jobs and directing initiators to initiate specific classes of jobs, it is possible to control the mixture of jobs that are performed concurrently.

job class queue: a waiting list of job definitions within the input queue in which jobs assigned the same class are arranged in order of priority; jobs with the same class and priority are placed in a first in/first out order.

job control language (JCL): a high-level programming language used to code job control statements.

***job control statement:** a statement in a job that is used in identifying the job or describing its requirements to the operating system.

job priority: a value assigned to a job that, together with an assigned job class, determines the priority to be used in scheduling the job and allocating resources to it.

job (JOB) statement: the job control statement that identifies the beginning of a job. It contains such information as the name of the job, an account number, and the class and priority assigned to the job.

job step: a unit of work associated with one processing program or one cataloged procedure and related data. A job consists of one or more job steps.

job step task: a task that is initiated by an initiator in accordance with specifications in an execute (EXEC) statement.

keyword parameter: a parameter that consists of a keyword, followed by one or more values.

library: a partitioned data set; see private library, system library, temporary library.

mutually exclusive parameters: parameters that cannot be coded on the same job control statement.

nonpageable dynamic area: an area of virtual storage whose virtual addresses are identical to real addresses; it is used for programs or parts of programs that are not to be paged during execution.

nonsharable volume: a volume that cannot be assigned to two or more data sets.

nonspecific volume request: a request that allows the system to select suitable volumes.

nontemporary data set: a data set that exists after the job that created it terminates.

null statement: a job control statement used to mark the end of a job's control statements and data.

output class: any one of up to 36 different categories, defined at an installation, to which output data produced during a job step can be assigned. When an output writer is started, it can be directed to process from one to eight different classes of output data.

output listing: a form that is printed at the end of a job that can contain such information as job control statements used by the job, diagnostic messages about the job, data sets created by the job, or a dump.

page: a fixed-length block of instructions, data, or both, that can be transferred between real storage and external page storage.

partition: see virtual storage partition.

partitioned data set: a data set in direct access storage that is divided into partitions, called members, each of which can contain a program or part of a program. Each partitioned data set contains a directory (or index) that the control program can use to locate a program in the library.

passed data set: a data set allocated to a job step that is not deallocated at step termination but that remains available to a subsequent step of the same job.

PEND statement: a job control statement used to mark the end of an in-stream procedure.

permanently resident volume: a volume that cannot be physically demounted or that cannot be demounted until it is varied offline (i.e., removed from the control of the central processing unit).

positional parameter: a parameter that must appear in a specified order.

private library: a user-owned library that is separate and distinct from the system library.

private volume: a mounted volume that the system can allocate only to a data set for which a specific volume request is made.

PROC statement: a job control statement that must mark the beginning of an in-stream procedure; it can also be used, in both cataloged and in-stream procedures, to assign values to symbolic parameters in the procedure.

procedure library: a partitioned data set containing cataloged procedures; the IBM-supplied procedure library is named SYS1.PROCLIB.

procedure step: that unit of work associated with one processing program and related data within a cataloged or in-stream procedure. A cataloged or in-stream procedure consists of one or more procedure steps.

qualified name: a data set name that is composed of multiple names separated by periods (e.g., A.B.C.). For a cataloged data set, each name corresponds to an index level in the catalog.

queue: a waiting line or list formed by items in a system waiting for service; for example, tasks to be performed or output to be written by a writer.

reader procedure: the cataloged procedure that controls the input stream reader.

real storage: the storage of a system/370 computing system from which the central processing unit can directly obtain instructions and data, and to which it can directly return results.

remote entry services (RES): the functions added to the job entry subsystem that allow servicing of remote devices by the job entry subsystem of VS1. These services allow jobs and their associated input and output to be entered from remote devices, processed at the central system, and then transmitted back to remote devices.

reserved volume: a volume that remains mounted until the operator issues an UNLOAD command.

restart facility: see checkpoint/restart facility.

return code: a value placed in the return code register at the completion of a program. The value is established by the user and may be used to influence the execution of succeeding programs or, in the case of an abnormal end of task, may simply be printed for programmer analysis.

segment: a continuous 64K area of virtual storage, which is allocated to a job or system task.

specific volume request: a request for volumes that informs the system of the volume serial numbers.

supervisor: the part of the control program that coordinates the use of resources and maintains the flow of CPU operations.

symbolic parameter: a symbol preceded by an ampersand that stands for a parameter or the value assigned to a parameter or subparameter in a cataloged or in-stream procedure. Values are assigned to symbolic parameters when the procedure in which they appear is called.

system generation: the process of using an operating system to assemble and link together all of the parts that constitute another operating system.

system library: a partitioned data set named SYS1.LINKLIB that contains frequently used programs and programs used by the system.

system output device: a device assigned to record output data for a series of jobs.

system output writer: a job scheduler function that writes output data sets onto a system output device, independently of the programs that produced the data sets.

SYS1.LINKLIB data set: see system library.

SYS1.PROCLIB data set: see procedure library.

task: a unit of work for the central processing unit from the standpoint of the control program; therefore, the basic multiprogramming unit under the control program.

temporary data set: a data set that is created and deleted in the same job.

temporary library: a library that is created and deleted within a job.

time sharing option (TSO): in VS2, an option that allows a number of users to execute programs concurrently and to interact with the programs during execution from remote terminals.

time slicing: in VS2, an optional feature that can be used to prevent a task from monopolizing the central processing unit and thereby delaying the assignment of CPU time to other tasks.

unit address: the three-character address of a particular device, specified at the time a system is installed; for example, 191 or 293.

universal character set (UCS) feature: a printer feature that permits the use of a variety of character arrays.

virtual storage: addressable space that appears to the user as real storage, from which instructions and data are mapped into real storage locations. The size of virtual storage is limited by the addressing scheme of the computing system and by the amount of auxiliary storage available, rather than by the actual number of real storage locations.

virtual storage partition: in VS1, a division of the dynamic area of virtual storage, established at system generation, that is allocated to a job step or a system task.

volume: that portion of an auxiliary storage device that is accessible to a single read/write mechanism.

volume table of contents (VTOC): a table on a direct access volume that describes each data set on the volume.

workstation: a terminal device that may or may not be a CPU. At a workstation, an operator can connect into a central system via LOGON, enter jobs, and receive output.

writer procedure: the cataloged procedure that controls the output stream writer.

Index

[ndexes to OS/VS publications are consolidated in the OS/VS Master Index, GC28-0602, and the OS/VS Master [ndex of Logic, GY28-0603. For additional information about any subject listed below, refer to other publications listed for the same subject in the Master Index.

GC28-0617-2

OS/VS JCL Services (File No. S370-36) Printed in U.S.A. GC28-0617-2

IBM®
International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)

OS/VS JCL Services

GC28-0617-2

READER'S COMMENT FORM

Your views about this publication may help improve its usefulness; this form will be sent to the author's department for appropriate action. Using this form to request system assistance or additional publications will delay response, however. *For more direct handling of such requests, please contact your IBM representative or the IBM Branch Office serving your locality.*

Possible topics for comment are:

Clarity Accuracy Completeness Organization Index Figures Examples Legibility

Cut or Fold Along Line

What is your occupation? ____

Number of latest Technical Newsletter (if any) concerning this publication: _______________

Please indicate in the space below if you wish a reply.

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. Elsewhere, an IBM office or representative will be happy to forward your comments.

GC28-0617-2

Your comments, please . . .

This manual is part of a library that serves as a reference source for system analysts, programmers, and operators of IBM systems. Your comments on the other side of this form will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Cut or Fold Along Line

Fold

Fold

First Class
Permit 170
Endicott
New York

Business Reply Mail
No postage stamp necessary if mailed in the U.S.A.

Postage will be paid by:

International Business Machines Corporation
Department G60
P. O. Box 6
Endicott, New York 13760

Fold

Fold

OS/VS JCL Services (File No. S370-36) Printed in U.S.A. GC28-0617-2

IBM®

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)

OS/VS JCL Services

GC28-0617-2

READER'S COMMENT FORM

Your views about this publication may help improve its usefulness; this form will be sent to the author's department for appropriate action. Using this form to request system assistance or additional publications will delay response, however. *For more direct handling of such requests, please contact your IBM representative or the IBM Branch Office serving your locality.*

Possible topics for comment are:

Clarity Accuracy Completeness Organization Index Figures Examples Legibility

Cut or Fold Along Line

What is your occupation? ____

Number of latest Technical Newsletter (if any) concerning this publication: ______________

Please indicate in the space below if you wish a reply.

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. Elsewhere, an IBM office or representative will be happy to forward your comments.

GC28-0617-2

Cut or Fold Along Line

Your comments, please . . .

This manual is part of a library that serves as a reference source for system analysts, programmers, and operators of IBM systems. Your comments on the other side of this form will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Fold Fold

First Class
Permit 170
Endicott
New York

Business Reply Mail
No postage stamp necessary if mailed in the U.S.A.

Postage will be paid by:

International Business Machines Corporation
Department G60
P. O. Box 6
Endicott, New York 13760

Fold Fold

OS/VS JCL Services (File No. S370-36) Printed in U.S.A. GC28-0617-2

IBM®

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)