GC24-5100-4
File No. S370-36

Systems

OS/VS1 JCL Services

Release 6.7

IBM

Fifth Edition (January 1979)

This edition, GC24-5100-4, applies to Release 6.7 of OS/VS1 and to all subsequent releases until otherwise indicated in new editions or Technical Newsletters. This edition incorporates the following two supplements:

GC24-5134 (SU5, OS/VS1 MSS Enhancements: March 15, 1977)
GC24-5131 (SU6, OS/VS1 Subsystem Attachment Support; October 6, 1977)

Changes are continually made to the information herein; before using this publication in connection with the operation of IBM systems, consult *IBM System/370 Bibliography,* GC20-0001, for the editions that are applicable and current.

This edition, GC24-5100-4, is a major revision of GC24-5100-3.

Summary of Amendments

For a detailed list of changes, see page iii.

This publication has been photocomposed through ATMS (an IBM Program Product) and TERMTEXT/Format (an IBM Installed User Program). For information regarding those programs, contact your IBM representative or the IBM branch office in your locality.

Changes or additions to the text and illustrations are indicated by a vertical line to the left of the change.

Summary of Amendments for OS/VS1 JCL Services

Summary of Amendments for GC24-5100-4 VS1 Release 6.7

This revision incorporates the following supplements:
GC24-5134 March 15, 1977 (SU 5)
GC24-5131 October 6, 1977 (SU 6)

Miscellaneous
Corrections and updates resulting from reader's comments are included.

Summary of Amendments for GC24-5100-3 VS1 Release 6

IBM 3203 Model 4 Printer
The 3203 Model 4 Printer is supported on System/370 Models 138 and 148. The IBM standard character sets for the 3203 Printer are listed in the section titled, "Requesting a Special Character Set."

IBM 3800 Printing Subsystem
The Burster-Trimmer-Stacker feature is supported.

Miscellaneous
Editorial Changes and changes resulting from reader's comments are included.

Summary of Amendments for GC24-5100-2 VS1 Release 5

IBM 3800 Printing Subsystem
The IBM 3800 Printing Subsystem information in this publication is for planning purposes only until the product becomes available.

This edition describes the JCL parameters associated with the IBM 3800 Printing Subsystem. These parameters are CHARS, FLASH, FCB, and MODIFY.

High-Density Dump
Describes the use of CHARS and FCB parameters to produce a compressed dump print out on the IBM 3800 Printing Subsystem.

Miscellaneous
Editorial changes and changes resulting from reader's comments are included.

Preface

OS/VS1 JCL Services describes the operating system services that can be requested by coding parameters of JCL (job control language). This publication is for applications programmers with a basic knowledge of computer operating systems and some familiarity with JCL. Background information on VS1 (Virtual Storage, Option 1) is in the *IBM System/370 System Summary,* GA22-7001.

This book contains the OS/VS1 information that was in *OS/VS JCL Services,* GC28-0617, and changes to VS1 JCL since VS1 Release 2. The book also incorporates "Appendix D: Creating and Retrieving Indexed Sequential Data Sets" and "Appendix E: Creating and Retrieving Generation Data Sets," from *OS/VS1 JCL Reference,* GC28-0618. Miscellaneous changes and additions have been made to some of this material.

Part 1, JCL Defines the Job, describes the nine JCL statements and the organization of services used in this book. It also includes a list of JCL services, noting the publication in which each is described, and the parameter, subparameter, or statement used to request the service. All JCL services are not described in this publication.

Part 2, Descriptions of JCL Services, is grouped into the following chapters:

Running Your Job
Describing and Defining Data Sets
Special Data Sets
Obtaining Output
Cataloged and In-Stream Procedures

Each chapter is divided into sections describing when or why to request the service discussed, and how to request or control the service.

JCL parameters are discussed only in the context of requesting services. Complete descriptions of these parameters are in the *OS/VS1 JCL Reference,* GC24-5099.

Other publications to which the text refers:

Data Processing Glossary, GC20-1699

IBM System/370 Bibliography, GC20-0001

IBM 3800 Printing Subsystem Programmer's Guide, GC26-3846

Forms Design Reference Guide for the IBM 3800 Printing Subsystem, GA26-1633

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

Operator's Library: OS/VS1 Reference, GC38-0110

OS/VS1 Checkpoint/Restart, GC26-3876

OS/VS1 Data Management for System Programmers, GC26-3837

OS/VS1 Data Management Macro Instructions, GC26-3872

OS/VS1 Data Management Services Guide, GC26-3874

OS/VS1 Debugging Guide, GC24-5093

OS/VS1 Planning and Use Guide, GC24-5090

OS/VS1 RES: Workstation Users Guide, GC28-6879

OS/VS1 Supervisor Services and Macro Instructions, GC24-5103

OS/VS1 Utilities, GC26-3901

OS/VS1 Access Method Services, GC26-3840

OS/VS1 IBM 3540 Programmer's Reference, GC24-5110

Contents

Figures

Chapter 1: JCL Defines the Job

You can write programs in any one of a number of languages, which the operating system translates into machine language to execute your instructions and perform your work. You define the *job* (a collection of related problem programs) that you submit to the system with JCL (job control language).

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 (execute) statement; each data set used by a job step, with a DD (data definition) statement; and the job itself, with a JOB statement. These three job control statements and six additional statements are summarized under "JCL Statements," later in this chapter.

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 automatically performs 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," later in this chapter.

JCL Statements

The job control language contains nine statements. Figure 1-1 summarizes the purpose of each statement.

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 ensures that JCL statements from another job are not read as part of your job; the delimiter statement can be used to indicate the end of data in the input stream. 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 section "Writing Cataloged and In-Stream Procedures." Use the command statement to submit commands through the input stream. On the comment statement, you can include information to make your programs more understandable by other programmers and by yourself. Coding details and a description of each statement are included in the *OS/VS1 JCL Reference,* listed in the Preface.

JCL is given its versatility by its 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, and syntax rules for coding these parameters, are in the *OS/VS1 JCL Reference,* listed in the Preface.

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 1-1. Each JCL Statement Has a Purpose

Introducing the JCL Services

JCL services described in this publication are divided into the following chapters:

Running Your Job
Defining and Describing Data Sets
Special Data Sets

Obtaining Output
Cataloged and In-Stream Procedures

Each chapter is divided into sections 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 list on the next few pages will acquaint you with the services available, where the service is described, and what statement, parameters or subparameters you code to request the service.

The list is divided into the following areas:

Running Your Job
Defining and Describing Data Sets
Special Data Sets
Obtaining Output
Cataloged and In-Stream Procedures
TCAM Services

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/VS1 JCL Reference,* listed in the Preface; services provided by many DCB subparameters are described in greater detail in *OS/VS1 Data Management Services Guide,* listed in the Preface. For example, the *OS/VS1 JCL Reference* tells you the data set organizations you can request in the DSORG subparameter; the *OS/VS1 Data Management Services Guide* describes the different data set organizations in detail.

List of JCL Services

RUNNING YOUR JOB

Service	Publication(s) where Described	JCL Statement, Parameter, or Subparameter Used
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
executing programs contained in libraries	*JCL Reference*, "PGM Parameter"	PGM parameter on EXEC statement
ISSP (Installation specified selection parameters), using	*JCL Services*, "Job Scheduling" and " Obtaining Output"	PROFILE AND MPROFILE on JOB statement; SYSOUT=PROFILE on DD statement
job class, assigning	*JCL Services*, "Job Scheduling"	CLASS or PROFILE parameter on JOB statement
job priority, assigning	*JCL Services*, "Job Scheduling"	PRTY or PROFILE parameter on JOB statement
job scheduling	*JCL Services*, "Job Scheduling"	PRTY or PROFILE, CLASS or PROFILE, and TYPRUN=HOLD parameters on JOB statement
limiting the amount of time a job uses the CPU	*JCL Reference*, "TIME Parameter"	TIME parameter on JOB statement
limiting the amount of time a job step uses the CPU	*JCL Reference*, "TIME Parameter"	TIME parameter on EXEC statement
MSS (Mass Storage System) considerations	*JCL Services*, "Mass Storage System Considerations"	UNIT,VOL=SER,SPACE,MSVGP parameters on DD statement.
passing accounting information to accounting routines	*JCL Reference*, "ACCT Parameter"	ACCT parameter on EXEC statement
passing information to processing program	*JCL Reference*, "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	*JCL Reference*, "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

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,* "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,* "* Parameter", "DATA Parameter", "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,* "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	*JCL Reference,* "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,* "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,* "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	*JCL Services,* "Requesting Units and Volumes for a Data Set"	SEP subparameter of UNIT parameter on DD statement
using MSS (Mass Storage System)	*JCL Reference,* "UNIT Parameter"	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 Services*, "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 Services*, "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 1 of 2)

Service	**Publication(s) where Described**	**JCL Statement, Parameter, or Subparameter Used**
abnormal termination dump, requesting	*JCL Services,* "Controlling the Output Listing of JCL Statements, Messages, and Dumps"	SYSABEND or SYSUDUMP DD statement
alignment of forms, requesting	*JCL Services,* "Printer Form and Character 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 or MPROFILE parameter on JOB statement
assigning an output data set to an output class	*JCL Services,* "Writing Output Data Sets"	SYSOUT parameter on DD statement
bursting of output	*JCL Services,* "Bursting of Output"	BURST parameter on DD statement
character set and arrangement selection	*JCL Services,* "Printer Form and Character Control"	UCS or CHARS parameter on DD statement
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 (or MPROFILE) 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 (or MPROFILE) parameters on JOB statement
controlling output to a workstation	*JCL Services,* "Controlling Output to a Workstation"	SYSOUT, DEST parameter on DD statement
copy modification	*JCL Services,* "Copy Modification"	MODIFY 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 Form and Character Control"	FOLD subparameter of UCS parameter on DD statement
forms control	*JCL Services,* "Printer Form and Character Control"	FCB parameter on DD statement
forms overlay	*JCL Services,* "Forms Overlay"	FLASH parameter on DD statement
holding an output data set	*JCL Services,* "Writing Output Data Sets"	HOLD=YES parameter on DD statement

OBTAINING OUTPUT (Part 2 of 2)

Service	**Publication(s) where Described**	**JCL Statement, Parameter, or Subparameter Used**
multiple copies of an output data set	*JCL Services,* "Requesting Multiple Copies of an Output Data Set"	COPIES parameter on DD statement
operator verification of special character sets on a 3211 or 3203-4 printer, requesting	*JCL Services,* "Printer Form and Character Control"	VERIFY subparameter of UCS parameter on DD statement
operator verification of a specific FCB module request	*JCL Services,* "Printer Form and Character Control"	VERIFY subparameter of FCB parameter on DD statement
printer form and character control	*JCL Services,* "Printer Form and Character Control"	UCS, FCB, or CHARS, and 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 Form and Character Control"	UCS or CHARS parameter on DD statement
special output form, requesting	*JCL Services,* "Printer Form and Character Control"	SYSOUT parameter on DD statement
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 Servic϶s,* "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

TCAM SERVICES

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

Chapter 2: 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 automatically performs 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 schedules your job for execution, but you can influence when your job is selected by coding the CLASS and PRTY (or the PROFILE parameter) parameters on the JOB statement.

This chapter is divided into the following sections:
Job Scheduling
Requesting Storage for Execution of a Program
Conditional Execution of Job Steps
Restarting a Job

Job Scheduling

As the system reads in jobs, it places them in an input queue. The input queue is divided into job class queues where each job is placed according to its 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 where your job is placed in the input queue by assigning a job class and priority to your job.

Although you can influence your job's selection by assigning a job class and priority, 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. For jobs 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. For example, you might submit two jobs, JOBA and JOBB, where JOBA must complete execution before JOBB is initiated --JOBA might create records that JOBB will use. You have to delay JOBB's initiation until JOBA completes execution. You can delay the job's initiation until required resources are available by coding TYPRUN=HOLD on the JOB statement.

Assigning a Job to a Job Class

Each installation establishes job classes 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 (that is, 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 system. The job class itself is a letter from A to Z, or a number from 0 to 9; its meaning is defined by the individual installation.

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

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

If the installation's system programmer has generated the ISSP (installation specified selection parameters) tables, job class may also be assigned by specifying the PROFILE parameter on the JOB statement. An example is shown in this section under "Assigning Job Class and Priority Using ISSP." If you do not code the CLASS parameter or the PROFILE parameter, the reader assigns a default class of A (unless the default class was changed by the installation) to the job. If you code an incorrect job class (other than A-Z, 0-9), 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: Dynamic dispatching and time slicing are two options that can be specified during system generation to provide for more efficient use of the CPU (central processing unit). They are described in the *OS/VS1 Planning and Use Guide,* listed in the Preface. Both options cannot be included in the same system.

Your installation assigns job classes to each partition; specific job classes should be assigned to partitions that include dynamic dispatching or 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 each 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.

Assign a priority to your job by coding the PRTY parameter on the JOB statement:

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

If the installation's system programmer has generated the ISSP (installation specified selection parameters) tables, priority may also be assigned by specifying the PROFILE parameter on the JOB statement. An example is shown in this section under "Assigning Job Class and Priority Using ISSP." 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 begins 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 chooses the job that was read into the system first.

If you do not code a priority or the PROFILE parameter, the system assigns the default established in the reader procedure.

Users with RES RTAM remote workstations can limit the scheduling priority of jobs submitted from a workstation with the PRIORITY field in SYS1.UADS. If the priority limit in UADS is less than the value on the JOB card, the job is assigned the lower UADS value for scheduling. However, all outputs for that job are assigned the priority specified on the JOB card. This allows a remote user the ability to place a priority on his output without disrupting the scheduling scheme set up by the system programmer.

Note: The system programmer can establish a number less than 13 as the highest priority for jobs submitted from a workstation. 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 Job Class and Priority Using ISSP

ISSP (installation specified selection parameters) tables of attributes can be defined by the system programmer at your installation to help assign your job to a job class and a priority to your job. ISSP can also be used to assign a message or a data set to an output class, as described in a later chapter, "Obtaining Output".

Describe the job with the PROFILE keyword on your JOB statement, using the parameters supplied by the system programmer to make up a profile string, and the job class and priority are assigned for you. The general format to follow when using the PROFILE parameter to assign job class and priority is:

```
//PGM JOB ...PROFILE='job profile string'
```

In the following example, the system programmer has provided:

- Type of run (RUN) as production (PROD), or test (TEST)
- Type of language (LANG) used (FORT, COBOL, PL/1, BAL, and *, where asterisk (*) is any character value)
- Estimated execution time (TIME) as (10, 25, 100, *, where asterisk (*) is any number greater than 100 and is the default)
- Jobs requiring setup (SETUP).

If you are running a PL/1 compile job of the shortest duration (10), and requiring setup, code:

```
//PGM JOB ...PROFILE=('RUN=TEST,
//         LANG=PL1','TIME=(10),SETUP')
```

For additional information concerning the use of ISSP, see the section "Installation Specified Selection Parameters" in the *OS/VS1 Planning and Use Guide*, listed in the Preface.

Delaying Job Initiation

To delay a job's initiation, code TYPRUN=HOLD on the JOB statement. The job is held until the operator issues a RELEASE command for the job. 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 does not know that a job has been held. When the operator releases the job, it 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 held 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 issues 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 *virtual storage:*

- Real storage is the storage of System/370 from which the CPU can directly obtain instructions and data and to which it can directly return results.
- Virtual storage is addressable space that appears to the user as real storage, from which instructions and data are mapped into real storage locations.

When a program is selected, it is brought into virtual 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 detail in *Introduction to Virtual Storage in System/370,* listed in the Preface.)

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).
- 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 *virtual=real* storage.

To identify a virtual=real program, specify ADDRSPC=REAL on the JOB or EXEC statement. For the size of the required real storage, specify REGION=nK. This coding on the JOB statement specifies that each step of the job must not be paged during its execution. To specify these requirements for aspecific step, code ADDRSPC=REAL and REGION=nK on the EXEC statement for that step. If the ADDRSPC parameter is coded on the JOB statement, it is ignored on EXEC statements in the job. Unless changed by the installation, the default assumed if you do not code the ADDRSPC parameter is ADDRSPC=VIRT, indicating that the program can be paged during its execution.

How to Request Storage with the REGION Parameter

Code the REGION parameter only for programs that must not be paged during their execution — the REGION parameter is ignored unless you also codeADDRSPC=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 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 depends on the physical size of the nonpageable dynamic area. System activity at the time the request is made may reduce the amount of available contiguous real storage and delay execution of the program.

Note: Main storage hierarchy support and the rollout/rollin feature are not supported in VS1. However, you need not recode statements that contain the ROLL parameter or that contain REGION specifications originally made for hierarchy support. The system checks the ROLL parameter for correct syntax, but otherwise ignores it. In the REGION parameter, the system rounds 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 rounds 11K to 12K, adds the values, and assigns 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 each job step must be entirely 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
```

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/1, 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 (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: 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 fails.) 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 and 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 bypasses one job step or all remaining job steps, depending on where 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 checks 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 is 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. Because 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 is executed or bypassed. You should code COND on the EXEC statement when:

- You want to specify different tests for each job step.
- You specify a true test and you want to skip just that one step, rather than bypass all subsequent steps in the job.
- You want to name a specific step whose return code is to be tested.
- 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 or abnormally terminates, 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)
```

Note: If a job step refers to a data set created in a preceding step, that data set does 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 is not 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 happens if the system encounters JCL errors or is unable to allocate space to a data set.

Note: 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 20 is less than the return code issued by any previous step.
3. Execute STEP5 if the return code tests are not satisfied, 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 to reduce the effects of abnormal termination. When a job step terminates abnormally or when a system failure occurs, checkpoint/restart allows 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/VS1 Checkpoint/Restart,* listed in the Preface.

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/VS1 Data Management Macro Instructions,* listed in the Preface.)

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 is 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 the output from all of the steps up to and including the step that abnormally terminated. You 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 section "Controlling the Output Listing of JCL Statements, Messages, and Dumps.")

Requesting Restart

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. The RD parameter is also ignored for system tasks and generalized start jobs.

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
```

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, 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, 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 begins 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 resumes at the beginning of the first step; if the first step calls a cataloged procedure, execution begins 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/VS1 Checkpoint/Restart,* listed in the Preface.) 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 which volume contains the checkpoint entry you are using.

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 "DCB Parameter" in the *OS/VS1 JCL Reference,* listed in the Preface; details on specifying label type are also included in the *OS/VS1 JCL Reference* under "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:

- 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.
- 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 automatically makes 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 do 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:

- You can alter the DD statements in the restart step, 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 DD (data definition) statement. The *OS/VS1 JCL Reference,* listed in the Preface, describes every parameter you can code on a DD statement and illustrates the parameters necessary to create, retrieve, and extend data sets. This chapter 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. This chapter contains the following sections:

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 a device type, such as 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, such as a reel of magnetic tape, a drum, or a disk pack.

Specifying Volume Information

Volumes must be mounted on direct access and magnetic tape 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. For an existing data set you must make a specific volume request; 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, for example, 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.

Volume State: The system assigns the following attributes to every volume to define the state of the volume:

- The *mount* attribute controls when a volume can be demounted.
- The *use* attribute controls when a volume can be assigned to satisfy a specific or nonspecific volume request.
- The *nonsharable* attribute controls whether the volume can be shared by two or more data sets.

The operator and system programmer at your installation can designate the state of volumes. In a special member of SYS1.PARMLIB called PRESRES, the system programmer can predefine mount and use attributes for any or all of the direct-access volumes at your installation. The operator can define or change a volume's state by issuing the MOUNT and UNLOAD commands.

The Mount Attribute: A volume must be mounted on a unit before you can use it. A volume is temporarily mounted by the operator to satisfy a volume request, or it is permanently mounted. The system assigns a mount attribute to a volume to control when it can be demounted. The mount attributes are:

- Permanently resident
- Reserved
- Removable.

Permanently resident volumes are volumes that can never be demounted. In the PRESRES member of SYS1.PARMLIB you can designate any direct-access volume as permanently resident, but the following volumes are *always* permanently resident:

- All volumes that cannot be physically demounted, such as a 2305 drum storage volume
- The volume from which the system is loaded (the system residence volume)
- Volumes containing system data sets that are used without being allocated.

Reserved volumes remain mounted until the operator issues an UNLOAD or VARY OFFLINE command. You can reserve both direct-access and tape volumes. A volume is usually designated as reserved to avoid repeated mounting and demounting of the volume when it is used by a group of related jobs. For example, a volume containing a private library to be used by a group of jobs can be reserved while those jobs are being run. A volume is reserved when you designate it as reserved in the PRESRES member of SYS1.PARMLIB or when the operator issues a MOUNT command for that volume.

Removable volumes can be demounted after their last use in a job step or when the units on which they are mounted are required for other volumes. All volumes not designated as either permanently resident or reserved are removable. The operator can change a removable volume to a reserved volume by issuing the MOUNT command.

The Use Attribute: The system assigns a use attribute to each volume to indicate that the volume can be assigned to satisfy either specific volume requests only or specific and nonspecific volume requests.

The use attributes are:

- Private
- Public
- Storage
- Scratch.

You can assign the *private* attribute to direct-access and tape volumes. Allocate a private volume only to satisfy a specific request. Demount the volume at the end of the step unless:

- The volume is being used by another job
- The volume is permanently resident or reserved
- The data set on the volume is passed
- You code RETAIN in the VOLUME parameter.

Only direct-access volumes can be assigned public or storage attributes. Allocate a public volume to temporary data sets when no specific volume is requested.

These volumes are commonly referred to as scratch volumes. You can also allocate a public volume to satisfy a specific request. You can change a public volume to private by coding the PRIVATE subparameter of the VOLUME parameter.

Allocate a *storage* volume to temporary or nontemporary data sets when no specific volume serial number is requested. A storage volume can also be allocated to satisfy a specific volume request when you specify the volume serial number of the storage volume.

Assign the *scratch* attribute to tape volumes only. It is similar to the public attribute for direct-access volumes. Scratch volumes are assigned to satisfy nonspecific volume requests for temporary data sets. A scratch volume becomes private if you code the PRIVATE subparameter, a specific request for the volume is made, or the data set residing on the volume is nontemporary.

Combinations of mount and use attributes are summarized in Figure 2-1.

The Nonsharable Attribute: You can assign the nonsharable attribute only to removable direct-access volumes. You can allocate a nonsharable volume to only one job step at a time; other jobs requiring the volume must wait until the job step currently using the volume releases control.

The system assigns the nonsharable attribute to direct-access volumes that might require demounting during execution of the step that requested the volume. The nonsharable attribute is always assigned to a volume when:

- You make a specific volume request and request more volumes than the number of devices allocated.
- You request unit affinity to a data set defined earlier in the job step and the data sets reside on different volumes. (Unit affinity is described under "Sharing a Unit Between Data Sets".)

Never assign the nonsharable attribute to a permanently resident or reserved volume.

Using Private Volumes: If you request a private volume, the system attempts to assure that 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. For example, you are making

Volume State	Temporary Data Set	Nontemporary Data Set	How Assigned	How Demounted
	Type of Volume Request			
Public/Permanently Resident[1]	Nonspecific or Specific	Specific	PRESRES Entry MOUNT command or by default	Always mounted
Private/Permanently Resident[1]	Specific	Specific	PRESRES Entry or MOUNT command	Always mounted
Storage/Permanently Resident[1]	Nonspecific or Specific	Nonspecific or Specific	PRESRES Entry or MOUNT command	Always mounted
Public/Reserved[1]	Nonspecific or Specific	Specific	PRESRES Entry or MOUNT command	UNLOAD or VARY OFFLINE command
Private/Reserved (Tape and direct access)	Specific	Specific	PRESRES Entry or MOUNT command (Only MOUNT command for tape.)	UNLOAD or VARY OFFLINE command
Storage/Reserved[1]	Nonspecific or Specific	Nonspecific or Specific	PRESRES Entry or MOUNT command	UNLOAD or VARY OFFLINE command
Public/Removable[1]	Nonspecific or Specific	Specific	VOLUME=PRIVATE is *not* coded on the DD statement	When unit is required by another volume.
Private/Removable (Tape and direct access)	Specific	Specific	VOLUME=PRIVATE is coded on the DD statement (Specific request or a nontemporary data set for tape also causes this assignment.)	After its use, or at end of job if RETAIN or PASS is coded.
Scratch (Tape only)	Nonspecific or Specific	Nonspecific or Specific	Any tape data set (Scratch volume becomes private if VOLUME=PRIVATE is coded, specific request is made, or data set is nontemporary.)	When unit is required by another volume.

[1]Direct access volumes only.

Figure 2-1. Combinations of Mount and Use Attributes

a specific volume request for a direct-access volume and want the volume to be private:

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

For a tape request, private is implied if:

- A specific request, except for a received data set which was not private when it was passed.
- Disposition of KEEP, CATALOG, UNCATALOG, or, if a nontemporary data set name is specified, PASS.

The system automatically demounts 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 remains mounted unless the system needs to use the drive for a later allocation request. 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 remains mounted unless the system needs to use the drive for a later allocation request. (See "Passing a Data Set" in this section for additional information about a passed data set.)

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; because 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 requires 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,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.

You should not use nonspecific volume references between storage volumes and public volumes. Likewise, within one job step, you should not create a data set and then refer to it on a second DD statement using DISP=OLD. Doing either may cause unsuccessful execution of the job step.

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 user-assigned 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 canceled.

A *device type* corresponds to a particular set of I/O devices and their features. For example, 2400-2 represents 2400-series magnetic tape units with 7-track capability and data conversion. For this device type you specify:

```
UNIT=2400-2
```

The system assigns an available 2400 tape unit with those features. (See *OS/VS1 JCL Reference,* listed in the Preface, for a device type list.)

When you code UNIT=3330V, the system assigns available storage in the Mass Storage System.

Each installation can also define user-assigned *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 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 section "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: When you make nonspecific volume requests for data sets, the system chooses volumes to be assigned to the data sets. An optional 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 for a new direct access data set, it is ignored.

Your installation can include the I/O load balancing feature in the system during system generation. If I/O load balancing is not included, 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, either because of insufficient devices available or because of insufficient space, the request is ignored.

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 affinity 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 setdefined on the DD statement requesting unit affinity is assigned a dummy status.

If all of the following conditions are present, the named data set might write over the data set defined on the DD statement requesting unit affinity:

- The named DD statement requests a scratch tape.
- The data set defined on the DD statement requesting unit affinity is opened before the data set on the named DD statement.
- The tape is not unloaded before the OPEN of the data set defined on the named DD statement.

Note: 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 is abnormally terminated. Also, an old direct access data set should never have unit affinity to a new, non-specific direct access data set.

When You Do Not Have to Code the UNIT Parameter: In a few cases, the system obtains 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, that is, when 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 volumes used by the data set are the same as those of a previous DD statement in the same step, the same unit is used for the common volumes in both DD statements. The unit specification still applies to any noncommon volumes. The system never assigns the same volumes to two different units in the same step.
- When the data set is to share tracks, blocks, or cylinders with an earlier data set, that is, when SUBALLOC or SPLIT is coded. (The SUBALLOC and

SPLIT parameters are described in the section "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 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 section "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 for a single data set, code the SPACE parameter on the DD statement defining the data set. Request space for a group of data sets by coding the SPLIT or SUBALLOC parameters (see the section "Requesting Space for a Group of Data Sets"). Space for VSAM clusters or components is created by the Access Method Services DEFINE command. For a description of DEFINE, see *OS/VS1 Access Method Services,* listed in the Preface.

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. In most applications, however, 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 section "Requesting Space for a Group of Data Sets") are 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 the number of units of measurement your data set requires. Also, 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 must specify only the *unit of measurement* the system should use to allocate space and the quantity of 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, code an integer, indicating how many blocks, tracks, or cylinders you require.

It is easiest to specify an average block length. The system computes the least number of tracks required to contain the number of blocks you specify and allocates 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, that is, 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 that is 8 bytes long. The simplest space request, then, is:

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

The system computes the number of 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 (block size), key length, and device overhead.

These variables, and examples of estimating space requirements for partitioned and indexed sequential data sets, are described in *OS/VS1 Data Management Services Guide*, listed in the Preface, under "Data Set Disposition and Space Allocation." Figures illustrating direct access capacities and track capacities are included in the *OS/VS1 JCL Reference*, listed in the Preface.

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 abnormally terminates the step or searches another volume, depending on the type of volume request you make. Figure 2-2 illustrates system action for determining if enough space is available to satisfy your primary request.

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 as many as five noncontiguous *extents* (blocks) of space. (If you specify user labels by coding 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. Therefore, the system allocates as many as four additional extents 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" later in this section 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 up to 15 times.

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 attempts to allocate additional space contiguous with the primary quantity. If contiguous space is not available, the system allocates five noncontiguous extents of space equaling the secondary quantity.

Secondary space need not be allocated on the same volume as the primary quantity. However, for data sets whose disposition is NEW or MOD, the system allocates all requested space on the same volume until it determines insufficient space is available or until it has allocated sixteen extents of space to the data set on one volume. If either of these conditions occurs, the system allocates space on the next volume specified for a specific request or the system requests that a scratch volume be mounted for a nonspecific request as long as sufficient volumes were specified in the volume count. (See "Multivolume Data Sets" in the section "Requesting Units and Volumes for Data Sets.")

When allocating a secondary quantity for a data set with a disposition of OLD (that is, a data set that is preallocated or is being written over), the system goes to the next volume, if you specified one, and determine if there is already a secondary quantity allocated. If

Type of Volume Request		System Action
Specific volume request (that is, 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 are still used to allocate secondary quantity, if necessary.) (2) Determines none of specified volumes contain sufficient space--job step is abnormally terminated
Nonspecific volume request (that is, system chooses volume)		If space is not available on first volume chosen, system chooses another volume and continues 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. The job step is abnormally terminated.

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

there is, the system will use that space instead of making another allocation. If you didn't specify another volume, the space will be allocated on the current volume. If you did specify another volume and no space is already allocated there for the data set, secondary space will be allocated there.

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. The directory is in the beginning of the primary space, and you cannot lengthen it by requesting a secondary quantity as you can lengthen the data set itself. 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 the number of records to request, see "Processing a Partitioned Data Set" in the *OS/VS1 Data Management Services Guide,* listed in the Preface.

If you are creating an indexed sequential data set that occupies more than one cylinder and are *not* defining the index on a separate DD statement, you may 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 in addition to the primary quantity. (The space request for an indexed sequential data set must be in terms of cylinders.)

For a partitioned data set the directory is allocated from the space you request as the primary quantity. Therefore, you must consider the size of your directory in estimating the primary space request. For an indexed sequential data set the space requested for the index is added to the primary quantity. 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 allocates as many as five extents of noncontiguous space equaling the primary request, as described earlier in this section under "How the System Satisfies Your Primary Request." The efficiency of input/output operations decreases 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 section "Requesting Space for a Group of Data Sets") or certain system data sets (for example, 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 is satisfied with contiguous space, but the secondary quantity will not necessarily be contiguous. The default for MSS (Mass Storage System) is CONTIG if you specify MSVGP without SPACE. (See *OS/VS1 JCL Reference,* listed in the Preface, for additional information about the SPACE parameter.)

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 if you open an existing data set for output (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. Failure to do so can cause unpredictable results and possibly cause lost records or no record found. 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 system abnormally terminates the job. 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 the number of records you want allocated for a directory. If you are defining an indexed sequential data set that occupies more than one cylinder and are *not* defining the index on a separate DD statement, you must request space for an index. 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 (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/VS1 JCL Reference,* listed in the Preface.

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.

IBM 3850 Mass Storage System Considerations

If you are using MSS (Mass Storage System) for new data set requests, these JCL parameters require special consideration:

UNIT

- Set to 3330V.

VOL=SER

- Mutually exclusive of MSVGP.
- You must code SPACE parameter.

- If you do not code VOL=SER, you must code either SPACE or MSVGP (or you may code both).

SPACE

- If VOL=SER is coded, you must code SPACE.
- If MSVGP is coded, SPACE is optional.

MSVGP

- Mutually exclusive of VOL=SER, VOL=REF, and SUBALLOC.
- SPACE parameter is optional except for BPAM, ISAM, and VSAM data sets.
- If you do not code the SPACE parameter, the default is that specified for the group.
- Mutually exclusive of the ABSTR keyword of the SPACE parameter. If you are using MSS enhancements, MSVGP is mutually exclusive of the ABSTR, MXIG, and ALX keywords of the SPACE parameter.
- In MSVGP=(id,ddname) the ddname refers to the name of the DD statement that identifies a Master Out data set that is to be separated from a Master In data set.

If you request a new nonspecific 3330V permanent data set and do not specify the MSVGP parameter, a mounted 3330V STORAGE volume is used. If one does not exist, a volume is selected from SYSGROUP.

If you request a new nonspecific 3330V temporary data set and do not specify the MSVGP parameter, a public or storage volume is used. If one does not exist, a volume is selected from SYSGROUP.

When coding MSVGP=(id,ddname) the ddname must be a nonblank name of a prior DD statement within the same step as the MSVGP DD statement. Only one ddname can be coded and it must be coded with the group id.

For a nonspecific temporary or permanent request, allocation to a SYSGROUP volume is guaranteed by specifying VOL=PRIVATE or MSVGP=SYSGROUP. If you specify VOL=PRIVATE, you must also code the SPACE parameter.

You should use the catalog facilities for all data sets residing in MSS. You should catalog new data sets when they are created. When data sets are processed so that they can be extended to other volumes, the catalog should be updated. When you reference an existing data set, you should locate it with the catalog.

Unless data sets within an MSS group have unique data set names, an abend occurs. When a data set is extended to multiple volumes with a MOUNT from EOV, MSSC (Mass Storage System Communicator) cannot determine that the volume being chosen does not contain a data set with the same name as the one being extended.

To extend multivolume data sets to a nonmounted volume, the unit count on the UNIT parameter must be less than the volume count on the VOLUME parameter.

To guarantee allocation of a nonsharable UNIT, you must specify MSVGP or VOL=PRIVATE.

All volumes containing part of a multivolume data set must be specified with a catalog reference or a volume serial DD list, if the data set can be extended. No data is considered for data reuse if the data is part of a multivolume data set (including GDG ALLs), or has an esoteric volume name. Data reuse is not considered for a mulivolume data set even if parallel mounting or volume count=unit count is specified in the JCL.

MSSC selects nonspecific mass storage volumes for allocation as primary volumes. MSSC selects nonspecific MSS volumes for EOV as secondary volumes. For a new nonspecific data set whose volumes are selected by MSSC, request only one device with JCL so that only the primary volume is requested by allocation.

If an old multivolume data set resides on volumes within a group, specify parallel mount on the UNIT parameter of the DD statement, or specify a unit count equal to the number of volumes containing the data set.

For MSS, parallel mounting, Unit=(3330V,P), is honored for all generations of a Generation Data Group retrieved by coding the Generation Data Group name without a generation number.

You should not specify deferred mounting for volumes belonging to a MSVGP if requests for new data sets are in that job step using MSVGP for the same group.

MSSC provides cylinder increment default space allocation if you specify MSVGP without the SPACE parameter. The SPACE parameter can be used to code track or block size increments. If you want noncontiguous primary space allocation, you must specify the SPACE parameter.

To create a BPAM or ISAM data set with a nonspecific request within a specified MSVGP, you must specify the SPACE and MSVGP parameters. The MSVGP space defaults do not support BPAM or ISAM data sets.

Requesting Space for a Group of Data Sets

You must request space for every data set you create on a direct access volume. This is usually done for each data set by coding the SPACE parameter, or, for VSAM clusters or components, by the DEFINE command. (See the section "Requesting Space for a Single Data Set.") 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 that contains 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. If other data sets are using the volume concurrently, the benefit is lost, so you should request a private volume. (For details on requesting a private volume, see "Using Private Volumes" in a previous section "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.

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 the number of tracks per cylinder are to be allocated to this data set.
- Code unit and volume information.

Optionally, 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 the number of tracks per cylinder 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," later in this section, 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 the number of blocks you expect to write for all the data sets combined. The system computes the number of cylinders required, depending on what device you request in the UNIT parameter.

To indicate the number of tracks per cylinder to be allocated to each data set, specify a percent of the total tracks on a cylinder. The system computes the number of 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 track 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 the type of operation you are performing, for example, reading, writing, modifying records.

Requesting Cylinders: To request cylinders, you specify CYL as the unit of measurement and an integer indicating the number of cylinders to 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 (block size), keylength, and device overhead. These variables are described in *OS/VS1 Data Management Services Guide*, listed in the Preface. Figures illustrating direct access capacities and track capacities are included in the *OS/VS1 JCL Reference,* listed in the Preface.

To indicate the number of tracks per cylinder to 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 the type of operation you are performing, for example, reading, writing, modifying records.

Requesting a Secondary Quantity: You can specify a secondary quantity of blocks or cylinders in the SPLIT parameter on the first DD statement in the sequence. The system allocates additional blocks or cylinders to any data set in the group that runs out of space. (If you request 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 abnormally terminates.

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 section "Requesting Units and Volumes for Data Sets.") You can code the request for an additional device 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 no data and has enough space for all the data sets in the group. Then you suballocate space from the master set for data sets in the group. An installation might reserve blocks of space for different departments, for 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 (that is, 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 are not allocated to a private volume unless they specifically request it by coding its volume serial number.) Access arm movement is decreased, because 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, which 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 section "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 section.

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, which is 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 section "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 section.)

Note: Space suballocated by cylinder starts at the beginning of the free space in the master data set, and is aligned to a cylinder boundary only if the free space begins on a cylinder boundary.

How a Secondary Space Request is Satisfied: Secondary space allocated for 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 extend 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 section "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. That is, 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 *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 whether 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 positions the read/write mechanism after the last record in the data set. If the volume information for the data set is supplied on the DD statement, in the system catalog, or passed with the data set from a previous step and the data set is not there, the system issues an appropriate error message.

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 section "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 (that is, 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.

For VSAM data sets whose volume and unit status have been obtained from the catalog, the only disposition supported is KEEP. The system changes all other normal and conditional dispositions to KEEP.

Disposition processing differs for data sets on direct access volumes and data sets on magnetic tape volumes. A direct access volume contains a VTOC (volume table of contents) 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 public tape volume, the tape is rewound and the volume is available for use by other job steps. If the data set resides on a private volume, the tape rewinds and unloads. 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 is not 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/VS1 JCL Reference,* listed in the Preface, under "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 directly from the catalog (that is, 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 is not removed. (The data set might 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/VS1 Utilities,* listed in the Preface.)

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 "LABELParameter" in *OS/VS1 JCL Reference,* listed in the Preface.)

For data sets on direct access devices, the entry describing the data set in the VTOC and the data set itself are 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. You can group collections of data sets by cataloging them; 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.

To request that a data set be cataloged, code CATLG as the disposition; the system does a CATBX which automatically creates any missing index levels. (See *OS/VS1 Data Management for System Programmers,* listed in the Preface.) The disposition CATLG implies KEEP.

You can specify a disposition of CATLG for an already cataloged data set. Do this when you are adding output to the data set (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 (that is, 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 is a GDG (generation data group). The entire GDG is stored under a single data set name; each data set within the group is called a *generation data set,* and is associated with a generation number that indicates how far removed the data set is from the original generation. When you create a new generation data set (for example, A.B.C(+1)), you may specify a disposition of PASS or CATLG. When you specify PASS be sure to catalog or delete the passed version before you create and catalog an additional new version, A.B.C(+2). For more information on defining and creating generation data groups, see the section "Creating and Retrieving Generation Data Sets."

You must not assign the disposition CATLG to a data set name enclosed in apostrophes.

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, unless the system needs to use the drive for a later allocation request; 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.

However, if a passed data set has not been received and a job step abnormally terminates, the passed data set assumes the conditional disposition specified the last time it was passed.

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/VS1 Data Management Macro Instructions,* listed in the Preface.)

Notes:

- If the status of a data set is NEW and you want to pass it, you can omit the term NEW. However, you must indicate its absence with a comma, for example, DISP=(,PASS).
- If the system finds it necessary to remove the volume containing a passed data set, it ensures through messages to the operator that the volume is remounted before its next use. Therefore, it is unnecessary for you to code RETAIN in the VOLUME parameter of a DD statement that specifies a disposition of PASS.
- If several data sets in a job have the same name, care should be taken when any are to be passed. Only one of these data sets can be passed at a time, that is, a job step must receive and not pass a data set before another data set with the same name can be passed. A data set can not be passed more than once, unless it is received before it is passed again.
- If you are assigning a disposition to a private library, which is defined on a JOBLIB DD statement, refer to the section "Creating a Private Library," later in this publication.

Receiving a Data Set: If you want a subsequent step to receive a passed data set, omit the VOL= parameter on the DD statement receiving the passed data set. This DD statement must identify the data set through the DSNAME parameter, and must contain the DISP= parameter. You should not provide unit information unless you want additional units allocated to the data set.

If a passed data set is not received by a subsequent step within a job, the normal default of KEEP for old data sets and DELETE for new data sets is assumed. If a passed data set has not been received and a job step abnormally terminates, the passed data set assumes the conditional disposition specified the last time it was passed.

A data set may not be received by more than one DD statement within the same step, even if the receiving DD statement passes it again. One way to have a second DD statement within the same step reference a received data set is to code VOL=REF=*.ddname.

Default Disposition Processing

If you either fail to code the DISP parameter, or omit one of the subparameters, the system supplies default values.

- Data set status (first subparameter) default is NEW.
- Normal data set disposition (second subparameter) default is determined by:
  a. the data set status.

     OLD, SHR, or MOD (when volume information is available) specified in the first subparameter causes the system to keep the data set.

     NEW (specified or default) or MOD (when volume information is not available) specified in the first subparameter causes the system to delete the data set.

  b. the PASS disposition specified in a prior step for a passed data set with a status of other than NEW.
- Conditional data set disposition (third subparameter) – if not specified and the step abnormally terminates, the system uses normal data set disposition (second subparameter) specification or default to process data sets.

If the step fails before execution begins (during unit allocation for example), the system keeps existing data sets and deletes new data sets regardless of the disposition specified.

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 the section "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 termination of the job that refers to the data set. A job should have exclusive control of a data set in order to modify, add, or delete records.

When you don't 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/VS1 Data Management Macro Instructions,* listed in the Preface.)

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 overrides 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; otherwise it is treated as the name of a temporary data set. Symbolic parameters and assigning values to symbolic parameters are described in the section "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 receives control of the data set if:

- The data set is not being used by another job.
- 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 does *not* receive control of a data set if:

- The data set is being used by another job and that job has exclusive control.
- 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 that a job requests are not available, the system issues a message to the operator indicating the unavailable data sets.

The operator replies with one of these responses:

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

Special Data Sets

Data sets to satisfy a special purpose are usually defined with a special ddname, a specific data set name, or a specific parameter. Special data sets are described in the following sections:

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 Retrieving Generation Data Sets
Creating and Retrieving Indexed Sequential Data Sets

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 be in a library (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 automatically looks 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 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 is in the system library, simply code the program name (PGM=program name). If the program is 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 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 section 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/VS1 Data Management Services Guide,* listed in the Preface.

Creating a Private Library

Use the JOBLIB or STEPLIB 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. Because 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, assign the library a name in the DSNAME parameter and 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, that is, 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 program named 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=MOD,
//              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 yet have any members. 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 (that is, several steps in your job will be using the library), use the JOBLIB DD statement to define the library: the library is 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 searches 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, that is, 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, 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. Use caution if you code SHR when you add members to the library in your job. (A discussion of sharing a data set is in the section "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, that is, 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 searches the libraries for a program in the order in which the DD statements defining the libraries are coded.

Temporary Libraries

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

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 SYSLMOD were included in PROCSTEP3 of the procedure, you would code PGM=*.STEP2.PROCSTEP3.SYSLMOD.

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 that 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/VS1 JCL Reference,* listed in the Preface.) 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 are described in the section "Requesting Units and Volumes for Data Sets.")

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.

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 are 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/VS1 JCL Reference,* listed in the Preface, 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 BSAM (basic sequential access method) or QSAM (queued sequential access method) 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 the section "Defining a Dummy Data Set."

Using a Dedicated Data Set for Allocating a Temporary Data Set

Temporary data sets 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 2-3.

The system uses 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 is not 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 occurs.

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 uses 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 2-3. Defining a Temporary Data Set in order to Use the Space Allocated to a Dedicated Data Set

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.

Creating and Retrieving Generation Data Sets

A generation data set is one of a collection of successive, historically related, cataloged data sets known as a generation data group. The system keeps track of each data set in a generation data group as it is created so that new data sets can be chronologically ordered and old ones easily retrieved.

To create or retrieve a generation data set, identify the generation data group name in the DSNAME parameter and follow the group name with a relative generation number. When creating a generation data set, the relative generation number tells the system whether this is the first data set being added during the job, the second, the third, etc. When retrieving a generation data set, the relative generation number tells the system how many data sets have been added to the group since this data set was added.

A generation data group can consist of cataloged sequential, partitioned, indexed sequential (if the data set is defined on one DD statement), and direct data sets residing on tape volumes, direct access volumes, or both. Generation data sets can have like or unlike DCB attributes and data set organizations. If the attributes and organizations of all generations in a group are identical, the generations can be retrieved together as a single data set (up to 255 data sets can be retrieved in this way).

Before You Define the First Generation Data Set

Before you define the first generation data set, you must build a generation data group index.This index provides lower-level entries for as many generation data sets (up to 255) as you would like to have in your generation data group. The system uses these lower-level indexes to keep track of the chronological order of the generation data sets. The index must reside on the system residence volume, or an alternate control volume. Use the IEHPROGM utility program to build your index; this program is described in *OS/VS1 Utilities,* listed in the Preface.

Another requirement of generation data groups is that a data set label list exist on the same volume as the index. The system uses this label to refer to DCB attributes when you define a new generation data set. There are two ways to satisfy this requirement: (1) create a model data set label before you define the first generation data set; or (2) use the DCB parameter to refer the system to an existing cataloged data set each time you define a new generation data set.

Creating a Model Data Set Label: To create a model data set label, define a data set and request that it be placed on the same volume as the generation data group index. This ensures that there is always a data set label on the same volume as the index to which the system can refer.

The name you assign to the data set may be the same as or different than the name assigned to the generation data group. (If you assign the same name for both, the data set associated with the model data set label cannot be cataloged.) You may request a space allocation of zero tracks or cylinders. The DCB attributes you can supply are DSORG, OPTCD, BLKSIZE, LRECL, KEYLEN, RKP, and RECFM.

This is an example of creating a model data set label:

```
//DD1 DD DSNAME=PAY.WEEK,
//        DISP=(NEW,KEEP),
//        UNIT=2314,VOLUME=SER=SYSRES,
//        SPACE=(TRK,0),DCB=(RECFM=FB,
//        LRECL=240,BLKSIZE=960)
```

You need not create a model data set label for every generation data group whose indexes reside on the same volume. Instead, you may create one model data set label to be used by any number of generation data groups. If you create only one model, you should not supply any DCB attributes. When you create a generation data set, specify the name of the model in the DCB parameter and follow the name with a list of all the DCB subparameters required for the new generation data set, that is, DCB=(dsname,list of attributes).

Referring the System to a Cataloged Data Set: If there is a cataloged data set that resides on the same volume as your generation data group index and you are sure that data set will exist as long as you are adding data sets to your generation data group, you need not create a model data set label. When you create a generation data set, specify the name of the cataloged data set in the DCB parameter, that is, code DCB=dsname. If all the DCB attributes are not contained in the label of the cataloged data set, or if you want to override certain attributes, follow the data set name with these attributes, that is, code DCB=(dsname,list of attributes).

Creating a Generation Data Set

When defining a new generation data set, always code the DSNAME, DISP, and UNIT parameters. Other parameters you might code are the VOLUME, SPACE, LABEL, and DCB parameters.

DSNAME Parameter: In the DSNAME parameter, code the name of the generation data group followed by a number enclosed in parentheses. This number must be 1 or greater. If this is the first data set you are adding to a particular generation data group during the job, code +1 in parentheses. Each time during the job you add a data set to the same generation data group, increase the number by one.

Any time you refer to this data set later in the job, use the same relative generation number as was used earlier. At the end of the job, the system updates the relative generation numbers of all generations in the group to reflect the additions. Unpredictable results may occur if you use a relative generation number that causes the actual generation number to exceed G9999.

DISP Parameter: Assign new generations a status of NEW and a disposition of PASS or CATLG in the DISP parameter, that is, code DISP=(NEW,PASS) or DISP=(NEW,CATLG). If you specify a disposition of PASS, take care to catalog or delete the passed version before you create and catalog a new version. For example, if you created A.B.C(+1) with a disposition of PASS, then you must not catalog A.B.C(+2) until you delete or catalog A.B.C(+1).

UNIT Parameter: The UNIT parameter is required on any DD statement that defines a new generation data set unless VOLUME=REF=reference is coded. In the UNIT parameter, identify the type of devices you want (tape or direct access).

Another way to request a device is to code UNIT=AFF=ddname, where the named DD statement requests the device or device type you want.

VOLUME Parameter: You may assign a volume in the VOLUME parameter or let the system assign one for you. The VOLUME parameter can also be used to request a private volume (PRIVATE), to retain the private volume (RETAIN), and to indicate that more volumes may be required (volume count).

SPACE Parameter: Code the SPACE parameter only when the generation data set is to reside on a direct access volume. You can code the SPLIT or SUBALLOC parameter in place of the SPACE parameter if the data set's organization permits the use of these parameters.

LABEL Parameter: You can specify label type, password protection (PASSWORD), and a retention period (EXPDT or RETPD) in the LABEL parameter. If the data set will reside on a tape volume and is not the first data set on the volume, specify a data set sequence number.

DCB Parameter: A model data set label may have the same name as the group name. If this is so, and if the label contains all the attributes required to define this generation, you need not code the DCB parameter. If all the attributes are not contained in the label, or if you want to override certain attributes, code these attributes in the DCB parameter, that is, code DCB=(list of attributes).

If a model data set label has a different name than the group name and if the label contains all the attributes required to define this generation data set, only the name of the data set associated with the model data set label need be coded. Code the name in the DCB parameter, that is, DCB=dsname. If all the attributes are not contained in the label, or if you want to override certain attributes, follow the data set name with these attributes, that is, code DCB=(dsname,list of attributes).

If no model data set label exists, you must code the name of a cataloged data set that resides on the same volume as the generation data group index, that is, code DCB=dsname. If all the attributes are not contained in the label for this data set, or if you want to override certain attributes, follow the data set name with these attributes, that is, code DCB=(dsname,list of attributes).

Retrieving a Generation Data Set

To retrieve a generation data set, you always code the DSNAME and DISP parameters. Other parameters you might code are the UNIT, LABEL, and DCB parameters.

If a generation data set is created but not opened, that is, left empty, and DISP=(NEW,CATLG) is specified, the system does not know the data set has not been opened, and catalogs it anyhow. Then, if the level of this unopened generation data set is specified in JCL, it is the data set to be allocated. The system does not allocate the next opened data set before the unopened generation data set.

For example, A.B.C(+1) and A.B.C(+2) are created in that order and only A.B.C(+1) is opened, A.B.C(+2) becomes the "current" generation because it was the most recently created and, for data retrieval, is referenced as A.B.C(+0) in JCL. The unopened data set, and not the one that was opened, is allocated to the job/step.

DSNAME Parameter: In the DSNAME parameter, code the name of the generation data group followed by a number enclosed in parentheses. The number you code depends on how many new generation data sets have been added to the group since this generation data set was added. If none has been added before the job, code a zero (0). If one has been added before the job, code (-1). Decrease the number by 1 until you determine the present relative generation number of the data set, then code this number.

When you refer to this data set later in the job, use the same relative generation number as was used earlier, even if another generation has been added during the job. An exception occurs when you delete one generation during a job, and in a later step of the same job reference a generation earlier than the one deleted. You must now add one to the relative generation number of the data set you reference. You can avoid this situation by deleting a generation data set in the last step of a job.

If you want to retrieve all generations of a generation data group as a single data set, or if your want to retrieve all generations of a generation data group by concatenation, in order, starting with the most recent data set and with unity affinity to the most recent data set, code the generation data group name without a generation number, for example, DSNAME=WEEKLY.PAYROLL. You can retrieve all generations by concatenating them only if the attributes and organization of all generations are identical.

Note: Relative generation numbers are based on the catalog as it existed at the start of the job, plus any changes made by cataloging new members of the data set during the job.

DISP Parameter: The DISP parameter must always be coded. The first subparameter of the DISP parameter must be OLD, SHR, or MOD. You must assign a disposition as the second subparameter. You should avoid coding PASS as the second subparameter when you retrieve a generation data set.

UNIT Parameter: Code the UNIT parameter when you want more than one device assigned to the data set. Code the number of devices you want in the unit count subparameter, or, if the data set resides on more than one volume and you want as many devices as there are volumes, code P in place of the unit count subparameter.

LABEL Parameter: Code the LABEL parameter when the data set has other than standard labels. If the data set resides on a volume and is not the first data set on the volume, specify the data set sequence number.

DCB Parameter: Code DCB=(list of attributes) when the data set has other than standard labels and DCB information is required to complete the data control block. Do not code DCB=dsname when retrieving a generation data set.

Resubmitting a Job for Restart

Certain rules apply when you refer to generation data sets in a resubmitted job (the RESTART parameter is coded on the JOB statement).

For step restart: If step restart is performed, generation data sets that were created and cataloged in steps preceding the restart step must not be referred to in the restart step or in steps following the restart step by means of the same relative generation numbers that were used to create them. Instead, you must refer to a generation data set by its present relative generation number. For example, if the last generation data set created and cataloged was assigned a generation number of +2, it would be referred to a 0 in the restart step and in steps following the restart step. In this case, the generation data set assigned number of +1 would be referred to as -1.

For checkpoint restart: If generation data sets created in the restart step were kept instead of cataloged (DISP=(NEW,CATLG,KEEP) was coded), you can, during checkpoint restart, refer to these data sets and generation data sets created and cataloged in steps preceding the restart step by means of the same relative generation numbers that were used to create them.

Generation data sets can be created and retrieved using utility programs. The method of doing this is described in *OS/VS1 Utilities,* listed in the Preface.

Examples of Creating and Retrieving Generation Data Sets

The following job step includes the DD statements that could be used to add three data sets to a generation data group.

```
//STEPA EXEC PGM=PROCESS
//DD1    DD   DSNAME=A.B.C(+1),
//            DISP=(NEW,CATLG),
//            UNIT=2400,VOL=SER=13846,
//            LABEL=(,SUL)
//DD2    DD   DSNAME=A.B.C(+2),
//            DISP=(NEW,CATLG),
//            UNIT=2314,VOL=SER=10311,
//            SPACE=(480,(150,20))
//DD3    DD   DSNAME=A.B.C(+3),
//            DISP=(NEW,CATLG),
//            UNIT=2305,VOL=SER=28929,
//            SPACE=(480,(150,20)),
//            DCB=(LRECL=120,BLKSIZE=480)
```

The first two DD statements do not include the DCB parameter; therefore, a model data set label must be on the same volume as the generation data group index and must have the same name as the generation data group (A.B.C). Because the DCB parameter is coded on the third DD statement, the attributes LRECL and BLKSIZE, along with the attributes included in the model data set label, are used.

The following job includes the DD statements required to retrieve the generation data sets defined above, when no other data sets have been added to the generation data group.

```
//JWC    JOB  CLASS=B
//STEP1  EXEC PGM=REPORT9
//DDA    DD   DSNAME=A.B.C(-2),DISP=OLD,
//            LABEL=(,SUL)
//DDB    DD   DSNAME=A.B.C(-1),DISP=OLD
//DDC    DD   DSNAME=A.B.C(0),DISP=OLD
```

Creating and Retrieving Indexed Sequential Data Sets

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

1. The *prime area* contains data and related track indexes. It exists for each indexed sequential data set.
2. The *overflow area* contains overflow from the prime area when new data is added. It is optional.
3. The *index area* contains master and cylinder indexes for any indexed sequential data set that has a prime area occupying more than one cylinder.

Indexed sequential data sets must reside on direct access volumes. The data set can reside on more than one volume and the device types of the volumes may in some cases differ.

Creating an Indexed Sequential Data Set

One to three DD statements can define a new indexed sequential data set. When you use three DD statements to define the data set, each DD statement defines a different area and the areas must be defined in the following order:

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

When you use two DD statements to define the data set, the areas must be defined in the following order:

1. Index area.		1. Prime area.
	or	
2. Prime area.		2. Overflow area.

When you define the data set with one DD statement, you are defining the prime area and, optionally, the index area.

When you define the data set with more than one DD statement, assign a ddname only to the first DD statement; the name field of the other statements must be blank.

The only DD statement parameters that can be coded when defining a new indexed sequential data set are the DSNAME, UNIT, VOLUME, LABEL, DCB, DISP, SPACE, SEP, and AFF parameters. When to code each of these parameters and what restrictions apply are described in the following paragraphs.

DSNAME Parameter: The DSNAME parameter is required on any DD statement that defines a new temporary or nontemporary indexed sequential data set. To identify the area you are defining, follow the DSNAME parameter with the area: DSNAME=name (INDEX), DSNAME=name (PRIME), or DSNAME=name (OVFLOW). If you are using only one DD statement to define the data set, code DSNAME=name (PRIME) or DSNAME=name.

When reusing previously allocated space to create an indexed sequential data set, the DSNAME parameter must contain the name of the old data set to be overlaid.

UNIT Parameter: The UNIT parameter is required on any DD statement that defines a new indexed sequential in the UNIT parameter and must *not* request DEFER.

If separate DD statements define the prime and index areas, you must request the same number of direct access devices for the *prime* area as there are volumes specified in the VOLUME parameter. You may request only one direct access volume for an index area and one for an overflow area.

A DD statement for the index area or overflow area can request a device type different than the type requested on the other statements.

Another way to request a device is to code UNIT=AFF=ddname (except for new data sets), where the named DD statement requests the direct access device or device type you want.

VOLUME Parameter: The VOLUME parameter is required only if you want an area of the data set written on a specific volume or the prime area requires use of more than one volume. (If the prime area and index area are defined on the same statement, you cannot request more than one volume on the DD statement.) Either supply the volume serial number or numbers in the VOLUME parameter or code VOLUME=REF=reference. In all cases, the VOLUME parameter can be used to request a private volume (PRIVATE) and to retain the private volume (RETAIN).

Note: If a new ISAM data set is being created with a nonspecific volume request and its DSNAME already exists on a volume eligible for allocation, the job may fail due to duplicate names on the volume. Under these conditions, successful allocation depends on where the old data set resides in relation to eligible devices. Failure of this type can be corrected by either scratching the old data set or renaming the new data set before resubmitting the job.

LABEL Parameter: The LABEL parameter need only be coded to specify a retention period (EXPDT or RETPD) or password protection (PASSWORD).

DCB Parameter: The DCB parameter must be coded on every DD statement that defines an indexed sequential data set. At minimum, the DCB parameter must contain DSORG=IS or DSORG=ISU. Other DCB subparameters can be coded to complete the data control block if it has not been completed by the processing program. When you use more than one DD statement to define the data set, code all the DCB subparameters on the first DD statement. Code DCB=*.ddname on the remaining statement or statements; ddname is the name of the DD statement that contains the DCB subparameters.

When reusing previously allocated space and recreating an ISAM data set, code the desired changes in the DCB parameter on the DD statement. Although you are creating a new data set, some DCB subparameters cannot be changed if you want to use the space the old data set used. The DCB subparameters you *can* change are: BFALN, BLKSIZE, CYLOFL, DSORG, KEYLEN, LRECL, NCP, NTM, OPTCD, RECFM, and RKP.

DISP Parameter: If you are creating a new data set and not reusing preallocated space, the DISP parameter need only be coded if you want to keep, DISP=(,KEEP), catalog, DISP=(,CATLG), or pass, DISP=(,PASS), the data set. If you are reusing previously allocated space and recreating an ISAM data set, code DISP=OLD. The newly created data set overlays the old one.

To catalog the data set when DISP=(,CATLG) is coded, or to pass the data set when DISP=(,PASS) is coded, define the data set on only one DD statement. If the data set was defined on more than one DD statement and the volumes on which the data set now resides correspond to the same device type, you can use the IEHPROGM utility program to catalog the data set. Refer to *OS/VS1 Utilities,* listed in the Preface, for details.

SPACE Parameter: The SPACE parameter is required on any DD statement that defines a new indexed sequential data set. Use either the recommended nonspecific allocation technique or the more restricted absolute track (ABSTR) technique. If you use more than one DD statement to define the data set, all must request space using the same technique.

Nonspecific Allocation Technique: You must request the primary quantity in cylinders (CYL). When the DD statement that defines the prime area requests more than one volume, each volume is assigned the number of cylinders requested in the SPACE parameter.

Use the *index* subparameter of the SPACE parameter to indicate how many cylinders are required for an index. When you use one DD statement to define the prime and index areas and you want to explicitly state the size of the index, code the *index* subparameter.

The CONTIG subparameter can be coded in the SPACE parameter. However, if you code CONTIG on one of the statements, you must code it on all of them.

You cannot request a secondary quantity for an indexed sequential data set. Also, you cannot code the subparameters RLSE, MLXIG, ALX, and ROUND.

Absolute Track Technique: The number of tracks you request must be equal to one or more whole cylinders. The address of the beginning track must correspond with the first track of a cylinder other than the first cylinder on the volume. When the DD statement that defines the prime area requests more than one volume, space is allocated for the prime area beginning at the specified address and continuing through the volume and onto the next volume until the request is satisfied. (This can only be done if the volume table of contents of the second and all succeeding volumes is contained within the first cylinder of each volume.)

The *index* subparameter of the SPACE parameter is used to indicate how many tracks are required for an index. The number of tracks that you specify must be equal to one or more cylinders. When you use one DD statement to define the prime and index areas and you want to explicitly state the size of the index, code the *index* subparameter.

SEP or AFF Parameter: Code the SEP or AFF parameter only if you want channel separation from the area or areas defined on the preceding statement or statements in the group. In order for the areas to be written using separate channels, you must also request devices by their actual address (for example, UNIT=190).

Note: If the indexed sequential data set is to reside on more than one volume and an error is encountered as the volumes are being allocated to the data set, use the IEHPROGM utility program to scratch the data set labels on any of the volumes to which the data set was successfully allocated before you resubmit the job. This utility program is described in *OS/VS1 Utilities,* listed in the Preface.

Area Arrangement of an Indexed Sequential Data Set

When you create an indexed sequential data set, the arrangement of the areas is based on two criteria:

- The number of DD statements used to define the data set.
- What area each DD statement defines.

An additional criterion is used when you do not include a DD statement that defines the index area:

- Is an index size coded in the SPACE parameter on the DD statement that defines the prime area?

For additional information on area arrangement of an indexed sequential data set, see *OS/VS1 JCL Reference,* listed in the Preface.

Retrieving an Indexed Sequential Data Set

If all areas of an existing indexed sequential data set reside on volumes of the same device type, you can retrieve the entire data set with one DD statement. If the index or overflow resides on a volume of a different device type, you must use two DD statements. If the index and overflow reside on volumes of different device types, you must use three DD statements to retrieve the data set. The DD statements are coded in the following order:

1. First DD statement - defines the index area
2. Second DD statement - defines the prime area
3. Third DD statement - defines the overflow area

The only DD statement parameters that you can code when retrieving an indexed sequential data set are the DSNAME, UNIT, VOLUME, DCB, and DISP parameters. When to code each of these parameters and what restrictions apply are described in the following paragraphs.

DSNAME Parameter: The DSNAME parameter is always required. Identify the data set by its name, but do not include the term INDEX, PRIME, or OVFLOW. If the data set was passed from a previous step, identify it by a backward reference.

UNIT Parameter: The UNIT parameter must be coded unless the data set resides on one volume and was passed. You identify in the UNIT parameter the device type and the number of these devices required.

If the data set resides on more than one volume and the volumes correspond to the same device type, you need only one DD statement to retrieve the data set. Request one device in the UNIT parameter per volume. If the index or overflow area of the data set resides on a different type of volume than the other areas, you must use two DD statements to retrieve the data set. On one

DD statement, request the device type required to retrieve the index or overflow area. On the other DD statement, request the device type and the number of devices required to retrieve the prime area and the overflow area if the overflow area resides on the same device type. If the index and the overflow areas reside on different device types from the prime area, a third DD statement is needed.

VOLUME Parameter: The VOLUME parameter must be coded unless the data set resides on one volume and was passed from a previous step. Identify in the VOLUME parameter the serial numbers of the volumes on which the data set resides. Code the serial numbers in the same order as they were coded on the DD statements used to create the data set.

DCB Parameter: You must code the DCB parameter unless the data set was passed from a previous step. The DCB parameter must always contain DSORG=IS or DSORG=ISU. You can code other DCB subparameters to complete the data control block if it has not been completed by the processing program.

DISP Parameter: The DISP parameter must always be coded. The first subparameter of the DISP parameter must be OLD or SHR. You can, optionally, assign a disposition as the second subparameter.

Examples of Creating and Retrieving an Indexed Sequential Data Set

The following job step includes the DD statement that could be used to create an indexed sequential data set. Each area of the indexed sequential data set is defined on a separate DD statement.

```
//OUTPUT4 EXEC PGM=INCLUDE
//GROUP1  DD   DSNAME=PART86(INDEX),
//             DISP=(,KEEP),UNIT=3330,
//             VOLUME=SER=538762,
//             SPACE=(CYL,10,,CONTIG),
//             DCB=(DSORG=IS,RECFM=F,
//             LRECL=80,RKP=1,KEYLEN=8)
//        DD   DSNAME=PART86(PRIME),
//             DISP=(,KEEP),
//             UNIT=(2314,2),
//             VOLUME=SER=(538763,
//             538764),SPACE=(CYL,(25),,
//             CONTIG),DCB=*.GROUP1
//        DD   DSNAME=PART86(OVFLOW),
//             DISP=(,KEEP),UNIT=2314,
//             VOLUME=SER=538765,
//             SPACE=(CYL,15,,CONTIG),
//             DCB=(*.GROUP1,OPTCD=I)
```

The following job step includes the DD statements required to retrieve the indexed sequential data set created above.

```
//INPUT12 EXEC PGM=ADD
//RET4    DD   DSNAME=PART86,
//             DCB=DSORG=IS,UNIT=3330,
//             DISP=OLD,
//             VOLUME=SER=538762
//        DD   DSNAME=PART86,
//             DCB=DSORG=IS,
//             UNIT=(2314,3),DISP=OLD,
//             VOLUME=SER=(538763,
//             538764,538765)
```

Two DD statements are required to retrieve the data set because the index area resides on a volume of a different device type than the volumes on which the prime and overflow areas reside.

Identifying Associated Data Sets

Associated data sets reside on diskette, are separate from the in-stream data, and are to be spooled as SYSIN data sets. Identify associated data sets by specifying a data set identifier (DSID) and, optionally, a volume identifier on the DD* or DD DATA statement in the job stream. For additional information about associated data sets, see *OS/VS1 IBM 3540 Programmer's Reference,* as listed in the Preface.

Subsystem Data Sets

There are no special requirements for subsystem data sets. They are defined by application programmers using the DD statement keyword SUBSYS. For further information on subsystem data sets see the publication, *OS/VS1 Planning and Use Guide*, as listed in the Preface.

Note: Checkpoint restart is not supported for job steps that contain DD statements specifying SUBSYS. The action of the CHKPT macro is suppressed for such job steps.

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 chapter includes the following sections:

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

Writing Output Data Sets

Requesting Multiple Copies of an Output Data Set Copy Modification

Printer Forms and Print Chain Control

Forms Overlay

Bursting of Output

Controlling Output to a Workstation

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.

If a step abnormally terminates, you can also request a *dump,* containing the contents of parts of virtual storage. You must include a DD statement, defining 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
: The JOB statement only.

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 section "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 the installation's system programmer generated the ISSP (installation specified selection parameters) tables, a message may also be assigned to an output class by specifying the MPROFILE parameter on the JOB statement. An example showing the use of ISSP to assign a message to an output class is shown in the section "Writing Output Data Sets."

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

Ordinarily, you 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 section "Writing Output Data Sets.")

Requesting an Abnormal Termination Dump

To obtain a dump if a job step abnormally terminates, 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, the dump contains only the contents of the processing program's virtual storage area.
- 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).

Complete descriptions of dumps, including dumps in high-density format available with the IBM 3800 Printing Subsystem, and information on reading dumps are included in the *OS/VS1 Debugging Guide,* listed in the Preface.

To have the dump printed, 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 section "Writing Output Data Sets." (If you do not receive the dump you expect, it may be because the dump was canceled by the operator.) 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 does 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
```

Specifying Dump Format on the IBM 3800 Printing Subsystem

The IBM 3800 Printing Subsystem allows you to optionally request system-formatted high-density problem program dumps. You specify high-density format options with CHARS=DUMP (204 characters per line) and/or FCB=STD3 (8 lines per inch) parameters on the SYSABEND or SYSUDUMP DD statement. Parameter specification for controlling the dump formatting options is:

CHARS=DUMP Specified	FCB=STD3 Specified	Characters Per Line	Lines Per Inch
NO	NO	120	6
YES	NO	204	6
NO	YES	120	8
YES	YES	204	8

You should submit only one dump formatting DD statement per job step.

The following DD statement requests the system to produce a dump that is 204 characters per line and 8 lines per inch.

```
//SYSABEND DD ...,FCB=STD3,CHARS=DUMP
```

Writing Output Data Sets

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

- Assign the data set to an output class.
- Specify 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, that include the system output writer and the DSO (direct 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. The 36 possible output classes are 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, code the SYSOUT parameter on the DD statement defining the data set:

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

If the system programmer has generated the ISSP (installation specified selection parameters) tables, you can assign an output data set to an output class by using the PROFILE keyword parameter instead of the CLASSNAME positional parameter on the SYSOUT DD statement.

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 the output classes are selected through ISSP, describe the output using the same profile string with both the PROFILE (on the DD statement) and MPROFILE (on the job statement) keywords. Additional information and an example showing the use of ISSP to assign output classes is shown in this section under "Assigning Output Classes Using ISSP." If you omitted the MSGCLASS and MPROFILE parameters, code the default output class for messages in the SYSOUT parameter. (For details on coding the MSGCLASS parameter, see the section "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 *system output writer* 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 that 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, including printers, punches, and magnetic tape.

The DSO (direct system output 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: 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, for example, a data set containing payroll information requires 5,000 payroll checks, the data set is not printed until the operator supplies those forms; if you are routing the data set to another destination, the data set is not printed until that destination requests it.

To delay writing the output, code HOLD=YES on the DD statement defining the data set:

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

The data set is placed in the held status until the operator releases it by issuing a ROUTE or RELEASE command. Notify the operator when you code HOLD=YES for a data set: when a data set is held, 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 section "Controlling Output to Workstations."

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.

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

Assigning Output Class Using ISSP

ISSP (installation specified selection parameters) tables of attributes can be defined by the system programmer at your installation to help assign a message or data set to an output class. The parameters supplied by the system programmer are used to make up profile strings, which are used to describe the job's output.

A message is assigned to an output class by coding the MPROFILE parameter with the message profile string on the JOB statement, using the following format:

```
//PGM JOB ...,MPROFILE=
//          'message profile string'
```

Assign a data set to an output class by coding the PROFILE parameter with the sysout profile string on the SYSOUT DD statement. The following example, showing the general format, omits the SYSOUT classname positional parameter, because the classname is specified by the PROFILE keyword parameter.

```
//DATASET DD SYSOUT=PROFILE=
//            'sysout profile string'
```

In the following example, the system programmer has provided:

a. Type of form (FORM) as (ENVELOPE,CHECK,MULTI,SINGLE), and

b. Number of pages (PAGES) up to 10 (10), 11 to 50 (50), or over 50 (*), the default value.

If you wish to print a job on single-part paper, with an estimated length of 35 pages, and you require the message output class to be the same as the data set output class, code:

```
//PGM      JOB ...,MPROFILE='FORM=SINGLE,
//             PAGES=50'
//DATASET DD   SYSOUT=(...,PROFILE=
//             ('FORM=SINGLE','PAGES=50'))
```

For additional information concerning the use of ISSP, see the section "Installation Specified Selection Parameters" in the *OS/VS1 Planning and Use Guide,* listed in the Preface.

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 allocates 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. As a result, no job identification is written with the output.

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 BSAM (basic sequential access method) or QSAM (queued sequential access method) 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 section "Defining a Dummy Data Set."

Requesting Multiple Copies of an Output Data Set

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 section "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.

With a 3800 Printing Subsystem, you can also specify multiple copies of each page, followed by the same number of copies of each successive page. Specification of a value greater than 255 causes the system to cancel the job. For the coding format and examples, see "COPIES Parameter" in the *OS/VS1 JCL Reference,* listed in the Preface.

For information and restrictions on coding COPIES when you specify the device with the UNIT parameter, see "Rules for Coding" under "COPIES Parameter" in the *OS/VS1 JCL Reference,* listed in the Preface.

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 additional copies of the data set currently being processed are 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 additional 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 all the data sets for the job; in this example, four copies of each data set are printed. The command must be issued while the writer is processing the job's output.

A maximum of 254 additional 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. For further information, including copy control for the IBM 3800 Printing Subsystem, see *OS/VS1 JCL Reference,* listed in the Preface.

Copy Modification

With the IBM 3800 Printing Subsystem, copy modification allows the printing of predefined data on all pages of a specific copy or copies of a data set. For example, column headings or explanatory remarks could vary from copy to copy of the same printed page of data. Copies might also be personalized with the recipient's name, address, and other desired information. Blanks or printable graphic characters could be used to suppress the printing of variable data on particular copies of a page. This is a function done in other printers by using short or spot carbon in the forms set.

The predefined data is constructed, and stored as a copy modification module by using the IEBIMAGE utility program (see "The IEBIMAGE Utility Program" in the *IBM 3800 Printing Subsystem Programmer's Guide).* The module is then identified and used by means of the MODIFY parameter, coded on the output DD statement. See "MODIFY parameter" in the *OS/VS1 JCL Reference* for the format specification.

Printer Form and Character 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 or arrangement, when output is being printed by a 3203-4 or 3211 Printer with the universal character set feature or by a 3800 Printing Subsystem.
- A specific FCB (forms control buffer) module that controls the number of lines per inch printed and the length of the form, when the data set is written to a 3203-4 or 3211 Printer or a 3800 Printing Subsystem.
- Printing of predefined information on output printed on the 3800 Printing Subsystem.
- Overlay of printed output with a specified forms image.

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 issues a message to the operator instructing him to supply form 1014 to the printer. The system does not issue a message telling the operator to mount special forms on the 3800 Printing Subsystem if

$$\text{FORMDEF=}\begin{Bmatrix}\text{HOLD}\\ \text{BYPASS}\end{Bmatrix}$$

is specified at system generation or by the operator via the SETPRT command.

Requesting a Special Character Set

The UCS (universal character set) feature allows for different sets of characters to be printed for commercial and scientific applications. This feature can be requested for 3203-4, 3211, or 1403 Printers during system generation.

To request a special character set for a 3203-4, 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:

3211	Characteristics
A11	Arrangement A, standard EBCDIC character set, 48 characters
H11	Arrangement H, EBCDIC character set for FORTRAN and COBOL, 48 characters
G11	ASCII character set
P11	PL/1 alphameric character set
T11	Character set for text printing, 120 characters

1403 or 3203-4	Characteristics
AN	Arrangement A, standard EBCDIC character set, 48 graphics.
HN	Arrangement H, FORTRAN/COBOL EBCDIC character set, 48 graphics.
PCAN	Arrangement A, preferred character set, 48 graphics.
PCHN	Arrangement H, preferred character set, 48 graphics.
PN	PL/1 character set, 60 graphics.
QNC	PL/1 commercially preferred character set, 60 graphics.
QN	PL/1 scientifically preferred character set, 60 graphics.
RN	FORTRAN/COBOL commercial, 52 graphics.
SN	Text printing, commercial, 84 graphics.
TN	Text printing, scientific, 120 graphics.
XN	High speed alphameric, 1403 model 2, 40 graphics.
YN	High speed alphameric, 1403 model N1, 42 graphics.

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 print train corresponding to the character set you request. If you do not code the UCS parameter, the operator supplies a default.

You can code the UCS parameter with the UNIT parameter or with the SYSOUT parameter:

```
//OUTPUT DD UNIT=3211,UCS=T11
//OUTPUT DD SYSOUT=C,UCS=A11
```

Note: UCS is ignored on the SYSOUT DD statement if the printer is a 1403 or if CHARS is specified for the 3800.

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 Character Arrangements With a 3800 Printing Subsystem

The CHARS parameter allows the specification of a character arrangement table name or names to be used when printing with the 3800. The table names supplied for the 3800 include all the print train names that are standard for the 1403-N1 and 3211 Printers, and others. Groups of these table names are selected at system generation. See your system programmer to learn the names available at your installation.

Up to four character arrangement tables can be used to print a data set. For details on specifying them, see "CHARS Parameter" in the *OS/VS1 JCL Reference,* listed in the Preface. When more than one character arrangement table is specified, you can dynamically select the one you want by coding OPTCD=J as a DCB subparameter and coding the first byte of each output data line (following the print control character) as a table reference character. For considerations before doing this, see the discussion of OPTCD=J in the *IBM 3800 Printing Subsystem Programmer's Guide,* listed in the Preface.

Further support for the 3800 is provided by the IEBIMAGE utility program. With it you can modify or construct character arrangement tables and graphic character modification modules, to allow substitution of existing or user-designed characters. For details on using this program, and for planning for the 3800, see the *IBM 3800 Printing Subsystem Programmer's Guide,* listed in the Preface.

The UCS (universal character set) parameter may be specified on the same output DD statement with the CHARS parameter to permit redirection of output between the 3800 and other printers. Also, if CHARS is not specified, the 3800 recognizes a valid character arrangement specified with UCS. The FOLD keyword is not recognized by the 3800, but using the GF10, GF12, or GF15 character arrangement table provides the folding effect.

Requesting Forms Control

You request a specific forms control image for a 3203-4 or 3211 Printer by coding an *image identifier* in the FCB (forms control buffer) parameter. For the 3800 Printing Subsystem, you specify the FCB *module name* with the FCB parameter. Although the FCB image for the 3203-4 or 3211 and the FCB module for the 3800 serve the same purpose, they are constructed in different manners and are not interchangable between the two printers.

For either the 3203-4/3211 or the 3800, the FCB image or module is stored on and retrieved for use from SYS1.IMAGELIB. For the 3203-4 or 3211 Printer, information on IBM-supplied FCB images and on defining and adding user images to SYS1.IMAGELIB is in *OS/VS1 Data Management for System Programmers,* listed in the Preface. For the 3800, analogous information on FCB modules is in the *IBM 3800 Printing Subsystem Programmer's Guide,* listed in the Preface.

Requesting Alignment of Forms for a 3203-4 or 3211 Printer: 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)
```

The ALIGN subparameter is not used by the 3800.

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. 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 3203-4, 3211, or a 3800, the FCB image or module named STD2 is loaded into the printer and operator verification is requested.

Forms Overlay

With the IBM 3800 Printing Subsystem, the forms overlay feature allows printing the image from a forms overlay negative together with the data being printed. This reduces the need for preprinted forms, and for changing of forms.

The FLASH parameter on the 3800 DD statement identifies the overlay to be used and the number of copies on which that overlay is to be printed. See "FLASH Parameter" in the *OS/VS1 JCL Reference,* listed in the Preface for details on specifying this parameter. For information on designing and making or obtaining forms overlay negatives, see the *Forms Design Reference Guide for the IBM 3800 Printing Subsystem,* listed in the Preface.

Bursting of Output

With the optional burster-trimmer-stacker added, the IBM 3800 Printing Subsystem can separate continuous form paper into individual sheets. The BURST parameter is used to specify to the operator whether the output is to go to the burster-trimmer-stacker or to the continuous forms stacker. For further information, and examples, see "BURST Parameter" in the *OS/VS1 JCL Reference.*

Controlling Output to a Workstation

With RES (remote entry services) you can submit jobs to a central computing center from a workstation and 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. You enter jobs 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 section "Writing Output Data Sets").
- A special output form by specifying a form number in the SYSOUT parameter (see the section "Printer Forms and Print Chain Control").
- Multiple copies of the data set (see the section "Requesting Multiple Copies of an Output Data Set").

RES 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, code the DEST parameter, as described in this chapter. A data set can also be routed to another destination by use of the operator command ROUTE, which is described in *Operators Library: OS/VS1 Reference,* listed in the Preface.

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 later in this section 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 addresses 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 in the held status.

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

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 is not executed and no message is 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/VS1 JCL Reference,* listed in the Preface.)

For example, you are routing a data set containing bank records to the destination identified as DEPT58 and are placing the data set in the held status:

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

This DD statement is included in the job named COMPUTE — when a data set is placed in the held status, it is identified by the name of the job that produced it.

You want to send a message to DEPT58, notifying them that the data set is in the held status:

```
// SEND  'data set from job COMPUTE held',
// USER=DEPT58,LOGON
```

To remove this data set from the held status, the operator at DEPT58 issues a ROUTE command, specifying the name of the job that 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 placed in a partitioned data set known as the procedure library. An *in-stream procedure* is a set of job control statements 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 chapter describes how to write and use cataloged and in-stream procedures; it is divided into the following sections:

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 library 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 section, "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 section "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 and in-stream procedures is that in-stream procedures are placed within the job that calls them, and 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/VS1*

Utilities, listed in the Preface. 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. Never use the PEND statement. 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 must 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, identify the procedure on the EXEC statement of the step calling the procedure by coding as the first operand on the EXEC statement:

- The procedure name, or
- PROC= the procedure name.

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; that priority is replaced by the default.
- 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 section "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. A PARM parameter that appears in a later EXEC statement in the called procedure is nullified.
- If the TIME parameter is coded, it applies to the total procedure. The TIME parameter is nullified if it appears on any of the EXEC statements in the called procedure.
- 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:

- Assign a time limit of 4 minutes to the entire procedure; TIME parameters on individual EXEC statements in the procedure are nullified.
- Allow automatic step restart for each step of the job by coding RD=R. The RD parameter is added to the first step of the job and overrides 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 to the system 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 (that is, 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/VS1 JCL Reference,* listed in the Preface.) 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 should not attempt 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.

Coding mutually exclusive parameters on a DD statement in a procedure normally causes the system to issue an error message. If you override that statement the system does not issue the error message, even if the overriding statement does not affect the mutually exclusive parameters. In this case the first coded of the mutually exclusive parameters takes precedence.

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))
```

and you want to make the following changes:

- Change the disposition on the DD statement named DD1A to CATLG.
- Change the unit on the DD statement named DD1B to TAPE.
- 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 the section, "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
```

and you want to make the following changes:

- Change the UNIT parameter on the statement named DDN to UNIT=180.
- 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 the section "Controlling the Output Listing of JCL Statements, Messages, and Dumps.") Procedure statements are identified on the output listing as illustrated in Figures 2-4 and 2-5. The output listing also shows the symbolic parameters and the values assigned to them.

Using Symbolic Parameters

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 in this section under "Assigning Values to and Nullifying Symbolic Parameters." A description of how to include symbolic parameters when writing a cataloged or in-stream procedure follows.

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, (for example, PARM=&ALLVALS or PARM=&DECK&CODE).

The symbolic parameter itself is one to seven alphameric and national (#, @, $) characters preceded

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 2-4. 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 2-5. Identification of In-Stream Procedure Statements on the Output Listing

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.

Keyword parameters that you can code on the EXEC statement (ACCT, ADDRSPC, COND, PARM, RD, REGION, and TIME) cannot be used as the name of a symbolic parameter. For example, you cannot code &REGION=200K or REGION=&REGION on the EXEC statement, but you can code REGION=&SIZE.

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.

Note: Symbolic parameters are not acceptable in the DSID, VOLUME, and DLM keywords on a DD* or DD DATA statement. Doing this may cause unpredictable results.

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. For details see "Assigning Values to and Nullifying Symbolic Parameters," later in this section.

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", later in this section.) 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 causes 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 appears as:

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

However, if you code:

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

and &SERNO is nullified, a leading comma remains and causes 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 remains at the beginning of the operand field if &POSPARM is nullified and again causes 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, that is, 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/VS1 JCL Reference,* listed in the Preface.)

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 one of these:

- Alphabetic
- Numeric
- Period
- One of the three national characters (#, @, or $).

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 appears 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=&DOCNO..TXT

  The period is required because a period follows the symbolic parameter. A single period appears 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 Values to and Nullifying Symbolic Parameters

When a procedure containing symbolic parameters 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 assigning it a value or nullifying it.
- The programmer who writes the procedure assigns a default value to or nullifies the symbolic parameter 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 symbolic 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 a 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).

The rules for assigning values to symbolic parameters are:

- 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.
- 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.
- 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.
- 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 does cause a JCL syntax error. (For a discussion of this, see "Caution Concerning Leading and Trailing Commas," earlier in this section.)

In these cases, you must include a comma when you assign a value to the symbolic parameter, that is,

```
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 DDDSN=DATA,DISP=(OLD,&DISP,KEEP)
```

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

```
//TIMES PROC DISP=
```

If you are calling the procedure, and no default was assigned to &DISP, or if &DISP 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,DISP=
```

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*, listed in the *Preface.*

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 C 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 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 abnormally 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. 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.

folding. a technique used with the universal character set (UCS) feature on an impact printer to allow each of the 256 possible character codes to print some character on a chain or train with fewer graphics. For example, it allows the printing of uppercase graphic characters when lowercase are not available in the character array on the chain or train.

forms control buffer (FCB). a buffer that is used to store vertical formatting information for printing, each position corresponding to a line on the form.

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.

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.

mass storage system (MSS). a storage system made up of a library of tape cartridges, a set of direct-access volumes, and a mechanism to transfer data between the two.

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 contains information such 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 (that is, 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. 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.

table reference character. a numeric character (0, 1, 2, or 3) corresponding to the order in which the character arrangement table names have been specified with the CHARS keyword.

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 slicing. 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. 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

GC24-5100-4

OS/VS1 JCL Services (File No. S370-36) Printed in U.S.A. GC24-5100-4

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, N. Y. 10604

IBM World Trade Americas/Far East Corporation
Town of Mount Pleasant, Route 9, North Tarrytown, N. Y., U. S. A. 10591

IBM World Trade Europe/Middle East/Africa Corporation
360 Hamilton Avenue, White Plains, N. Y., U. S. A. 10601