JCL

Reference Manual

DPS7000/XTA
NOVASCALE 7000

Job Control and IOF

REFERENCE
47 A2 11UJ 04

DPS7000/XTA NOVASCALE 7000

JCL

Reference Manual

Job Control and IOF

December 1997

BULL CEDOC
357 AVENUE PATTON
B.P.20845
49008 ANGERS CEDEX 01
FRANCE

REFERENCE
47 A2 11UJ 04

Printed in France

Suggestions and criticisms concerning the form, content, and presentation of this book are invited. A form is provided at the end of this book for this purpose.

To order additional copies of this book or other Bull Technical Publications, you are invited to use the Ordering Form also provided at the end of this book.

Trademarks and Acknowledgements

Preface

Objectives

This manual describes the Job Control Language (JCL) for the GCOS 7 (General Comprehensive Operating System) operating system.

Intended Readers

You should have an elementary overall knowledge of GCOS, and be familiar with the principles of job management. The information given in this manual is intended for all users of DPS 7000 systems with the GCOS 7 operating system (Release V8).

Structure of This Manual

Section 1	contains a general description of the management of jobs and of the role of the JCL within GCOS. The distinction is made between the fundamental JCL statements of the operating system (Basic JCL Statements) and those statements used in conjunction with various system utilities (Extended JCL Statements).
Section 2	describes the rules for writing JCL statements.
Section 3	describes how to name files, volumes and devices within JCL statements.
Section 4	contains detailed descriptions of all the Basic JCL Statements. It is organized in alphabetical order of statement name, for easy reference.
Section 5	lists the extended JCL statements and tells you where to find more information about them.
Appendix A	lists the names of JCL statements that have abbreviations.
Appendix B	gives ASCII - EBCDIC character conversion information.

Format of JCL Statements

The format model for each statement is written in a typeface different from the body of the text and adheres to the following notation conventions:

Syntax appears in the following font: `ASSIGN internal-file-name`

UPPER CASE	indicates a keyword item to be entered exactly as shown.
`[item]`	An item within square brackets is optional.

`{item1}`
`{item2}`
`{item3}`
	A column of items within braces means that one value must be selected if the associated parameter is specified. If the parameter is not specified, the underlined item is taken as the default value.

`( )`	Parentheses must be coded if they enclose more than one positional item.
`...`	An ellipsis indicates that the preceding item may be repeated one or more times.

Associated Documents

You are advised to refer to *Error Messages and Return Codes* Guide *47 A2 10UJ* in conjunction with this manual. The *JCL User's Guide 47 A2 12UJ* is recommended as a basis for using the GCOS Job Control Language.

The following manuals (mentioned throughout this manual), where appropriate, will be of particular interest to those readers who require more information about the GCOS 7 operating system:

Communications

GCOS 7 MCS User's Guide 47 A2 32UC
Remote Facilities DPS6 to DPS7 ..... 47 A2 11UC
UFT User's Guide 47 A2 13UC
DJP User's Guide 47 A2 14UC

File Management and Utilities

UFAS-EXTENDED User's Guide 47 A2 04UF
Data Management Utilities User's Guide 47 A2 26UF
GAC-EXTENDED User's Guide 47 A2 11UF
Catalog Management Manual 47 A2 35UF
Cartridge Tape Library Unix Server User's Guide 47 A2 63UU

Interactive Facilities

Languages

Program Preparation

System Administration

Transaction Driven Subsystem (TDS)

Table of Contents

Appendices

Illustrations

Figures

Tables

1. Introduction

This section describes, in general terms, the structure of a user's job under GCOS 7, and traces its progress through the system. The following paragraphs give only a brief explanation of how to handle jobs. The *JCL User's Guide* contains a detailed and more illustrative description.

1.1 JOB DESCRIPTION

1.1.1 General Terminology

Work is normally submitted to the operating system as an **input stream**, which consists of one or more **job descriptions**. Each job description defines a unit of work, known as a **job**, that a user of the computer system wishes to run. Job descriptions are composed of Job Control Language, **JCL statements**, which supply information about the jobs and direct how they are run. The task of loading input data into a library may be done by a **data enclosure**, which does not constitute a job, but is at the same level as a job. The syntax of JCL statements is described in Section 2.

1.1.2 Job Enclosure

A **job enclosure** is the complete set of JCL statements that make up a **job description**, starting with a $JOB statement and finishing with a $ENDJOB statement. A typical job enclosure is made up of one or more **step enclosures** and possibly one or more **input enclosures** (Figure 1-1).

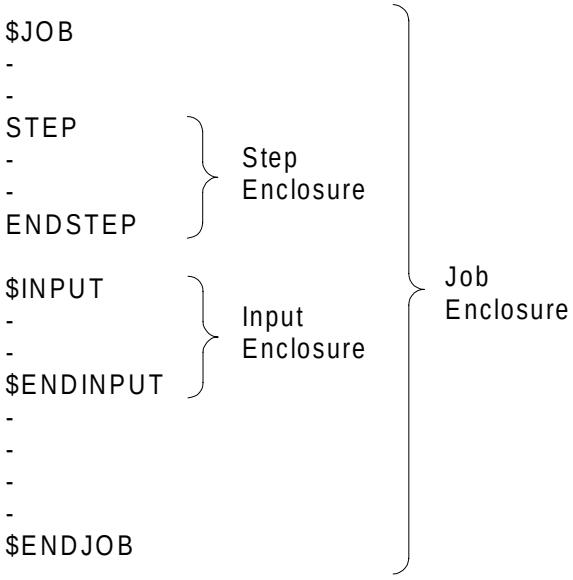


Figure 1-1. Job Enclosure

1.1.3 Step Enclosure

A job **step** specifies the loading and execution of a program, known at execution time as a **load module**. The set of JCL statements that specifies the characteristics and requirements of a job step is known as a **step enclosure**, and several of these enclosures may appear in the description of a job. As shown in Figure 1-1, each step enclosure starts with a STEP statement and finishes with an ENDSTEP statement.

1.1.4 Input Enclosure

An **input enclosure**, beginning with a $INPUT statement and ending (normally) with a $ENDINPUT statement, contains data records to be associated with one or more steps in the job. These records may be, for example, data for a user load module or a source program to be read by a compiler.

1.1.5 Data Enclosure

A **data enclosure**, beginning with a $DATA statement and ending (normally) with a $ENDDATA statement, contains data to be loaded into a specified library. It is at the same level as a job enclosure.

1.1.6 Sample Input Stream

Each user job in an input stream is delimited by JCL $JOB ..... $ENDJOB statements. The system resource requirements are defined by JCL statements and enclosed by STEP ..... ENDSTEP statements. Data in the input stream is delimited by $INPUT ..... $ENDINPUT statements. The structure of an input stream therefore consists of three levels of enclosure (see Figure 1-2). Their functions are to:

- Make the job known to the system: JOB enclosure.
- Describe the handling of each step to the operating system: STEP enclosure.
- Define data input in a job stream: INPUT enclosure.

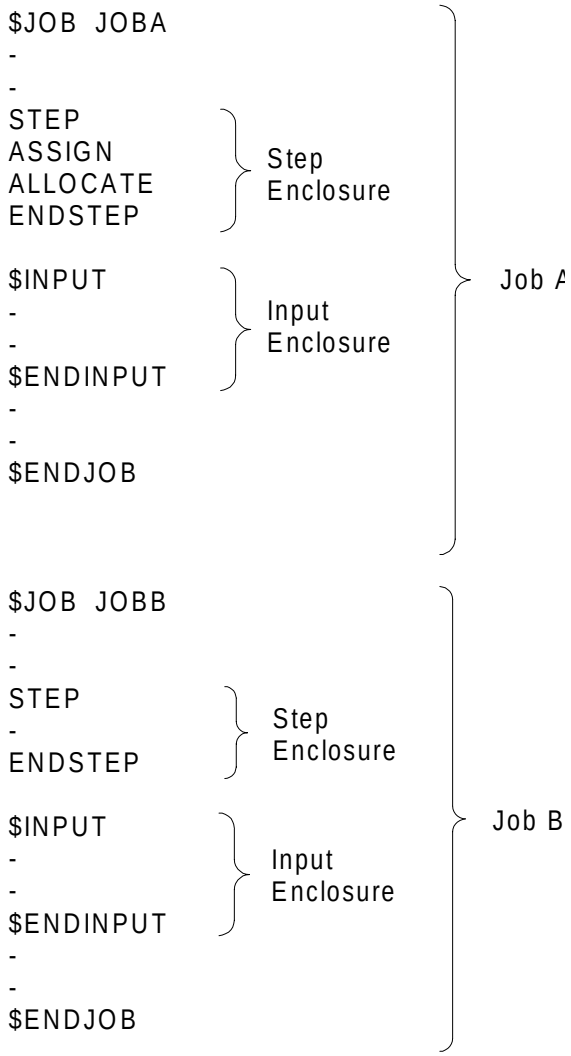


Figure 1-2. Sample Input Stream

1.2 TYPES OF JCL STATEMENTS

Most of the statements of a job description are related to a step description; that is, they name the load module to be executed, and define the conditions and resources required for the step.

However, if a step involves the execution of a system program, for example a compiler or utility program, then you are not required to include all the JCL statements that define the step. Instead, only one statement is necessary (for example: COBOL, LIBMAINT), with the appropriate parameter information that is associated with the statement. The specified statement is equivalent to a set of basic statements that fully define the requirements of the step. When the job description is translated to an internal form, the single statement originally written by the programmer is expanded into this set of statements. Therefore, the Job Control Language consists of two types of statements:

- **Basic JCL Statements**, each of which requests a specific system operation; examples are STEP, ASSIGN.

- **Extended JCL Statements**, each of which expands into a set of Basic JCL Statements and is equivalent in itself to a complete step description; examples are COBOL, SORT, PREALLOC.

Section 4 of this manual contains detailed descriptions of the Basic JCL statements. Although Extended JCL Statements are fully documented in other publications, Section 5 contains a brief definition of each extended JCL statement with a reference to the manual that contains the full statement description.

NOTE: Despite their differences in application, both types of JCL statements are written using identical syntax rules (as described in Section 2).

Note the following important points with respect to the documentation and handling of the two types of JCL statement:

- Although each of the statements, RUN and EXECUTE, expands into a step, neither is a utility statement (that is, both are considered to be part of the basic operating system). Consequently, they are treated as Basic JCL Statements and are described in this manual.

- All Extended JCL Statements, except SORTWORK and GSORTWRK, must appear outside step enclosures.

1.3 SUBMITTING JOBS

You can submit jobs to the operating system in the following ways:

- locally at your site, from magnetic files (on disks, tapes, cartridges, ...);
- from a remote station, under RBF (Remote Batch Facility) and/or DJP (Distributed Job Processing);

NOTE: A user of IOF (Interactive Operation Facility) can enter certain JCL statements directly to the system (that is, outside a job enclosure). Full details are given in the *Interactive Operation Facility Manual*. In particular, an IOF user terminal may issue a RUN statement to submit a stored job stream to the system.

1.4 JOB RUN AND JOB STATES

Job Run

The whole life cycle of a job, in contrast to the period of this cycle during which the job is actually being executed, is called the job run.

During its run, a job is uniquely identified by a Run Occurrence Number (). The RON is simply a number of up to four digits that is always preceded by the letter X. For example, X12, X27, X118 are possible RONs.

Steps of a job are referenced by their physical order in the job description; that is, the Step Number.

Job States

A job run can be thought of as occurring in stages. Each stage of the job run is associated with a job state. These states are:

```
INTRODUCED, IN SCHEDULING, HOLD, EXECUTING, SUSPENDED, OUTPUT
```

Job states are outlined in Figure 1-3. First, the job is introduced to GCOS: That is, a request to execute this job is sent to GCOS with the job description; thus the job is "known" to the system. It is at this time that a RON is assigned to the job run. The job description is then translated to an internal format more suitable for execution. In this process all JCL syntax errors are detected.

From the time a job is introduced until the end of JCL translation, the job is in an INTRODUCED state. The Input Reader is used to introduce and translate job descriptions. Once translated, the job goes to the IN SCHEDULING state. The Job Scheduler, following various rules, chooses from all jobs IN SCHEDULING, the next job that will be executed. This job then enters the EXECUTING state.

Eventually the job executes and it goes to the OUTPUT state and stays there until all its output has been printed by the Output Writer, after which the job is no longer known to GCOS.

Two other job states exist, the HOLD state and the SUSPENDED state. These states are used less frequently. Jobs in the HOLD state, although at the same stage as jobs IN SCHEDULING, are removed from the scheduler queue and ignored by the Job Scheduler. A job can be put in the HOLD state by a specific parameter (HOLD) in the $JOB statement or by an operator command, HOLD_JOB (HJ).

A job in the HOLD state can be put IN SCHEDULING by an operator command, RELEASE_JOB (RJ), or a RELEASE statement in a job other than the job that is in the HOLD state. The HOLD state can be used when a job that is introduced should not be executed before a certain event has occurred.

You can use the HOLD_JOB (HJ) () operator command to change the order in which jobs are scheduled.

EXECUTING jobs can be suspended by the HJ command.

They are then in the SUSPENDED state. This can prove useful to handle recurrent resource conflicts between two jobs.

You can reactivate a suspended job by means of the RELEASE_JOB command (RJ). In this case, the job's execution priority (see below) determines when the job is started again.

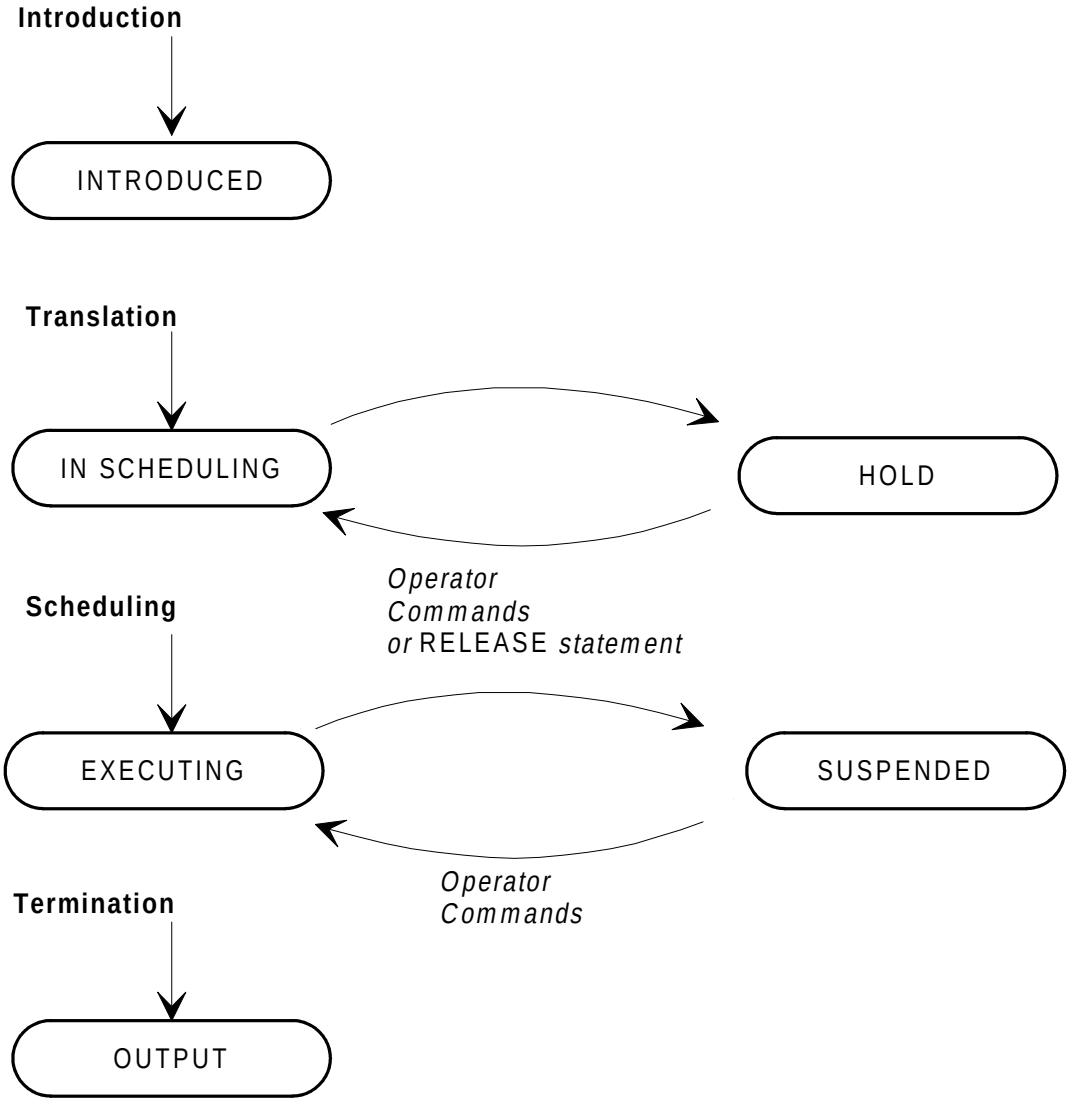


Figure 1-3. Relationship between the Stages and States of a Job Run

1.5 INPUT READER

The Input Reader is activated by an operator command, or by loading a specific diskette. The Input Reader is in charge of reading a stream of job descriptions, analyzing each job and translating the JCL to a format recognizable by GCOS, and storing the corresponding data and JCL statements separately for scheduling and executing the job later.

The Input Reader has two successive load modules: the Stream Reader and the JCL Translator. The Stream Reader and the JCL Translator are separate load modules in a specific job (see "Service Jobs" below).

1.5.1 Stream Reader

When the Stream Reader is requested to read an input (see Figure 1-4), it assigns the input device and awaits the first record. For each job, the JCL statements in the input stream are stored (in "source" form) as a file in the backing store. Each input enclosure is stored separately in a system file known as SYS.IN. In this way, the Stream Reader separates input enclosures from the job description, giving one stored JCL file and "n" SYSIN sub-files for "n" input enclosures.

If the Stream Reader encounters a $SWINPUT statement, it switches reading from the current stream to the file indicated by $SWINPUT. When the $ENDJOB statement is encountered, an entry is made in a queue that is accessible by the Translator. The Stream Reader then repeats its activity for the next job, until the end of the input stream is reached. At this point, an end of input notification is sent to the Translator and translation is requested.

NOTE: Once the Stream Reader has separated and stored the JCL file, the $JOB and $ENDJOB statements are no longer part of the job and are not passed to the JCL Translator. After the "n" input enclosures have been stored, none of the $INPUT and $ENDINPUT statements exist in the job description. Similarly, after the Stream Reader has finished switching, the $SWINPUT statements no longer exist in the job description.

1.5.2 Input Stream

Figure 1-4 illustrates the input stream.

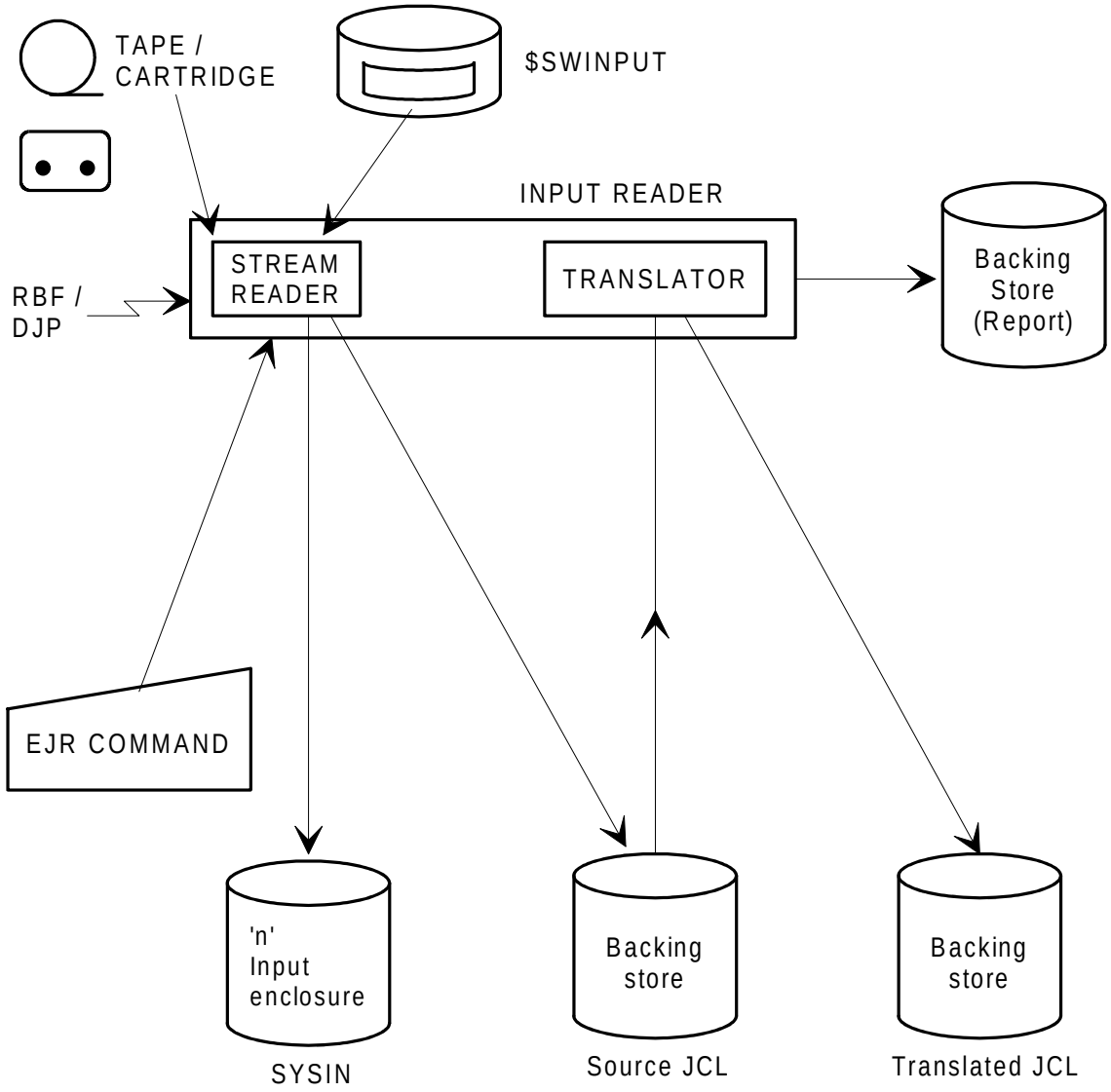


Figure 1-4. Input Reader and Associated Files

1.5.3 JCL Translator

When the JCL Translator is notified by the Stream Reader that the end of an input stream has been reached, it starts to translate the JCL statements stored in the backing store files, into a format that is executable by the Command Interpreter (described below). For each JCL file produced by the Stream Reader's activity, the translator opens a file in the backing store, to which the "translated" JCL statements of a job description are written. In addition, a listing of the JCL and Translator error messages is sent to the (Job Occurrence Report).

Although the jobs are introduced into the system (by the Input Reader) in the order in which they appeared in the job stream, the jobs in the input stream are selected for translation according to their scheduling priority (as specified in the $JOB statement).

1.6 JOB SCHEDULING

When the translation of a job description is complete, if the Input Reader has found no errors, the Job Scheduler determines when the job can be started.

1.6.1 Job Scheduler

The Job Scheduler controls the system load at the job level, and allows you to organize and plan the workload to optimize the throughput of the system. In addition, it provides you with a series of commands (both JCL statements and OPERATOR commands) to control the job flow.

The Scheduler Queue

There is a scheduling priority associated with each job, indicating its priority of scheduling relative to other jobs. Scheduling priorities range from 0 (the highest) to 7 (the lowest). Jobs IN SCHEDULING are stored in one queue (the Scheduler Queue), which they enter at the end of the JCL translation and leave when they start to execute.

Scheduling priority can optionally be specified for a job (PRIORITY parameter in $JOB statement). If it is not, a default value that is associated with the job class is used.

NOTE: You can modify the scheduling and execution priorities of a job as well as its state.

1.6.2 Control of Jobs

There are practical reasons for restricting the number of jobs that are executing simultaneously. For example, if three tape drives are available, it is useless to try to execute two jobs simultaneously if each requires three tape drives. For such reasons GCOS provides a set of functions to control the number and type of jobs executing concurrently as well as the order in which they are executed.

The number of jobs that are in the EXECUTING and SUSPENDED states at a given time is known as the System Load. There is an overall limit, known as the System Multiprogramming Limit (or maximum system load) that controls the number of such jobs.

At any time, the jobs that make up the System Load are divided among one or several job classes.

1.6.3 Job Class and Class Attributes

1.6.3.1 Job Classes

Job classes are a means of organizing your workload, to optimize the throughput of jobs in a multiprogramming environment. For example:

- Some jobs need to be executed serially (one after another) because of the installation dependencies or constraints.
- Some jobs must be executed at given periods of time (night shifts, etc.).
- To take the best advantage of multiprogramming, some job mixes (running simultaneously) are more favorable than others. For example, central processor bound and I/O bound jobs are grouped together, instead of groups of all central processor bound jobs.
- If several jobs use the same resources (for example tape units or other peripherals) they should not be scheduled simultaneously: this will minimize the amount of time spent waiting for a resource to become available.

Each user job entering the system belongs to a certain class. You can associate a series of attributes with each class and thus control the job mixes and workload planning. The class or classes under which a project can run a job are stored in the description of the project in the Site Catalog. The job classes are assigned to a project by the System Administrator using the MAINTAIN_CATALOG commands CRP and MDP.

A job class is identified by one or two letters:

A...Z, AA...AZ, BA...BZ,...,ZA...ZZ

This provides a theoretical limit of 702 different job classes, however, a maximum of 256 can actually be supported by the job scheduler.

The one-character classes, A to Z, are initialized at configuration time. Their default attributes are summarized in Table 1-1. Under certain conditions these attributes can be modified, but the class cannot be deleted.

The two-character classes must be created and then initialized using the START_LOAD command. The two_character classes can also be created at CONFIG time. They can be deleted using the TERMINATE_LOAD command. Refer to the *System Operator's Guide* for details.

- Classes A to Q and AA to QZ are for user jobs.
- Classes R to Z and RA to ZZ are reserved for service jobs. They are not available to user jobs.
- Classes A to P and AA to PZ are for batch jobs, with class P as the default.
- Classes A to Q and AA to QZ are for IOF jobs, with class Q as the default.

A class multiprogramming limit (or maximum class load) controls the number of jobs that can execute simultaneously in a given class (the actual number at any time is called the class load).

Default job scheduling and step execution priorities, and default multiprogramming limits are associated with each job class.

In addition, you can suspend or reactivate a job class. This provides further capabilities to select job processing and manage the serial execution of jobs.

1.6.3.2 Class Attributes

A job is scheduled and then executed according to rules imposed by the job class to which the job is assigned. These rules are called job class attributes. Three attributes are attached to each class and used to optimize your work load as long as the job itself exists. These attributes are never erased by a system shutdown or system crash, and are unaltered unless the permanent backing store is destroyed. You can modify them with the MODIFY_LOAD (MDLD) command, if necessary.

The attributes are:

Scheduling Priority

The scheduling priority; (PRIORITY) determines where in the queue and for how long the job waits to run. The scheduling priority of a job may be restricted for this project by the authorized classes in the site catalog. The project default priority is used for jobs whose priority violates this restriction or that have no priority specified in the $JOB or RUN statements. If there is no default value based on project, then the default priority is based on job class. You can also override it with the MODIFY_JOB (MDJ) command.

Execution Priority

The execution priority (XPRTY) determines the job's competitiveness for memory and CPU resources. The default value is allocated to all steps of all jobs in a class for which the execution (or dispatching) priority (XPRTY) is not specified in the STEP statement. Again, you can override it with the MDJ command.

Multiprogramming Limit

The multiprogramming limit (MAXLOAD) determines the maximum number of jobs in execution or suspended within a given class.

The default values for these attributes are defined for job classes A to Z when the system is configured. The system is delivered with the preset Job Classes, Scheduling Priorities and Execution Priorities shown in Table 1-1. For the two-character job classes, the default values are set when the class is created.

Table 1-1. Job Class Attributes and Default Values (1/2)

Job Class	Default Scheduling Priority (PRIORITY)	Default Execution Priority (XPRTY)	Default Multiprog. Limit (MAXLOAD)	Recommended Usage	Comment
USER JOBS					
A	7	9	1	batch jobs	
B	7	9	1	batch jobs	
C	7	9	1	batch jobs	
D	1	5	1	batch jobs	
E	2	4	1	batch jobs	
F	3	7	1	batch jobs	
G	4	9	1	batch jobs	
H	6	0	1	communication	
I	7	9	4	batch jobs	
J	6	0	1	TDS	
K	7	9	1	batch jobs	
L	7	9	1	batch jobs	
M	7	9	1	batch jobs	
N	7	9	1	batch jobs	
O	7	9	1	batch jobs	
P	7	9	40	batch (default)	

Table 1-1. Job Class Attributes and Default Values (2/2)

Job Class	Default Scheduling Priority (PRIORITY)	Default Execution Priority (XPRTY)	Default Multiprog. Limit (MAXLOAD)	Recommended Usage	Comment
SERVICE JOBS					
Q	0	1	100	IOF jobs	
R	0	1	6	Readers	(1)
RA				OLTD (Job TD-MVH)	defined by OLTD
RB				Mirror (Job REFFIL)	defined by mirror
S	0	0	12	Telecom (QMON, FECM, FEPS, TNS, RAEH, , OCS) IOSER ARS	(1)
T	7	2	6	Telecom Services (Job SERVER2)	
U	0	1	6	RBF	
V	7	9	1	System Trace (Job TRCCL)	
W	0	1	8	Output Writer (Job WRITER)	(1), (2)
X	0	2	4	JCL Translator (Job JTRA) Tds-HA Services (Jobs CMSR, RECOV, JRU)	(1), (2)
Y	7	2	6	Telecom (Jobs VCAM, GTP) Segment Server for TDS (Job JPPC)	(1), (2)
Z	0	0	4	Main Operator (Job IOF) Local Message Handler (Job LAEH)	(1)

(1) Class always started

(2) Use IDLE state

1.7 JOB EXECUTION

1.7.1 Job Selection

Whenever the system load is less than the system multiprogramming limit and the scheduler queue contains at least one job whose class load is less than the multiprogramming limit of its class, a job will be selected for execution from the scheduler queue. The selection is based on the job classes of those jobs already executing, with the job classes of the members of the queue.

1.7.2 Command Interpreter

When the Scheduler selects a job from the queue, it notifies the Command Interpreter, which then accesses the job's translated JCL from the backing store. For each JCL statement, the function of the Command Interpreter initiates the appropriate operating system action that is requested by that statement.

Before a step can execute, its requests in the JCL for resources (user buffer space, files, volumes, devices) must be analyzed and fulfilled, or the step must queue for resources.

1.7.3 Step Execution

Each time a STEP statement is encountered, the Command Interpreter calls the "Initiation" routine. The step initiation routine reads all of the statements of the step enclosure and allocates system resources to the step.

If the resource cannot be allocated, the step must wait until the resource is available. When all necessary resources have been allocated to the step, the appropriate load module is loaded from the load module library and step execution begins when ENDSTEP is encountered.

1.7.4 Step Termination

At step termination, allocated resources are released and the Command Interpreter resumes control. If the job contains another step, the cycle of resource allocation, step execution and resource release is repeated. When the end of the translated JCL is reached, job execution is terminated.

Figure 1-5 illustrates the processing of a one-step job. Certain errors or abnormal conditions during step initiation or execution may cause the step to be aborted. The JUMP statement can be used to anticipate the error situation at the termination of a step and prevent the abnormal termination of an entire job. The LET statement (SEV parameter) can set status values to simulate a normal or abnormal step termination.

1.7.5 Job Termination

When the Command Interpreter encounters the end of the job, certain job termination procedures are started, to delete temporary files and SYSIN subfiles, to produce final accounting information, and to inform the operator. At this point, the output for the job is available to be handled by the Output Writer.

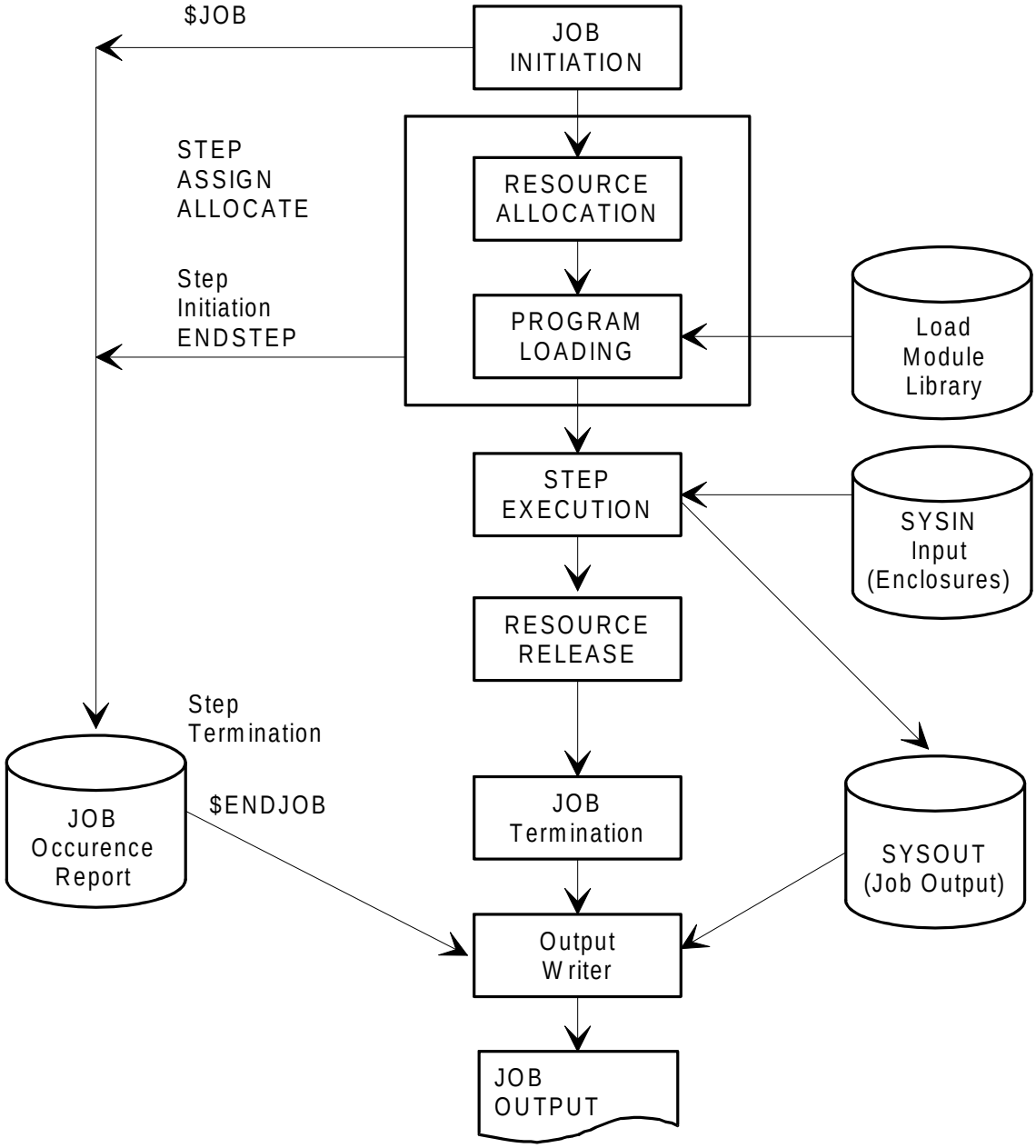


Figure 1-5. Step Execution for a Single-Step Job

1.8 SERVICE JOBS

A service job is a job that the system runs in order to execute certain system functions.

Service jobs are not always running, because they provide temporary functions or because they need to be active for only certain types of work.

The following are service jobs:

- Interactive Operation facility (IOF)
- Input Stream Reader (Reader)
- Disk Move Head T & D (TD_MVH)
- Automatic refilling of mirrored disks (REFILL)
- Queue Monitor (QMON)
- Front End Processing Support (FEPS)
- Front End Control and Management System (FECM)
- Transport and Network Subsystem (TNS)
- Networking - Remote Administrative Exchange Handler (RAEH)
- Automatic Restore and Save (ARS)
- Open Communication Subsystem (OCS)
- Input/Output Server for either Large Memory Cache or Mirror Disks or both (IOSER)
- DJP File Transfer Server (SERVER2)
- Remote Batch Facility (RBF)
- System Trace (TRCCL)
- Output Writer (WRITER)
- JCL Translator (JTRA)
- HA Complex Management Server (CMSR)
- HA Takeover File Recovery (RECOV)
- HA Journal Recovery Utility (JRU)
- Virtual Communications Access Method (VCAM)
- Generalized Transfer Processor (GTP)
- TDS Segment Management (JPPC)
- Local Administrative Exchange Handler (LAEH)

NOTES:

1. Service jobs are distinct from those components of the system known as system processors (for example, compilers, data management and library utilities) in the sense that system processors are called from within user jobs (by Extended JCL statements). Exceptions to this are the RUN and EXECUTE statements, which can call service jobs from within user jobs.

2. All service jobs are assigned a ron (run ocurrence number) when introduced to the system. The job class is exclusive to that particular service job (See Table 1-1, earlier in this chapter).

3. There can be only one JTRA job executing at any one time.

4. There is one READER job for each device on which Input Reader has been started.

5. Service jobs do not display log messages concerning run states, except for error messages.

6. Service jobs pass through the same states as user jobs, but with certain differences.

   Some service jobs use a specific state called IDLE. They enter the IDLE state when they run out of work. However, as soon as work is available (for example, transfers to be made, jobs to be translated, or output to be printed), they automatically enter the IN SCHEDULING state. The IDLE state economizes the use of system resources and removes the need for operator intervention.

1.9 FACILITIES AVAILABLE THROUGH JCL

The following paragraphs outline the features of the operating system that you can access with Basic JCL Statements. For more information on a feature, refer to the specified statement or to the appropriate user's manual.

1.9.1 Spooling of a Unit Record Device

Spooling is a system feature that makes it appear to the user that a program is using a unit record device directly while in fact it is using intermediate storage on system disk files to eliminate user device assignment problems. In a general sense the device is equivalent to a sequential disk file with certain processing restrictions (for example, a printer cannot be "read"). Full details of the use of unit record devices are given in the *Unit Record Devices User's Guide*.

1.10 HANDLING INPUT

1.10.1 The SYSIN Mechanism

The SYSIN mechanism allows programs that read data from a unit record input device to execute concurrently without having to deal with device assignment problems. The SYSIN mechanism provides buffering of input files on mass storage devices, thereby speeding up job execution time. While some jobs are executing, input for another job can be read concurrently.

1.10.2 Use of SYSIN

The SYSIN mechanism uses the SYSIN system file (SYS.IN) that is created at system generation and is stored on a resident disk device. For each input enclosure, one subfile is assigned. The Stream Reader removes each input enclosure from the job description and stores it in the assigned subfile. The format given to the input data is determined by the TYPE parameter specified in the $INPUT statement.

The data is transferred to a subfile of the SYSIN file. To access the SYSIN subfile, you request the SYSIN mechanism and the assignment of the subfile with the following ASSIGN statement:

```
ASSIGN internal-file-name, *input-enclosure-name;
```

Each member of the SYSIN file is a temporary subfile. When the job terminates, the system deletes all the SYSIN subfiles assigned to the job.

1.11 HANDLING OUTPUT

1.11.1 Output Writer

The Output Writer mechanism allows programs that send data to a unit record output device (printer or cassette) to execute concurrently without being concerned with device assignment problems. The Output Writer mechanism provides buffering of printer and output files on secondary storage devices, thereby speeding up the execution of the job. While some jobs are executing, the output from jobs that have finished executing can be printed concurrently. A file to be printed by the Output Writer is known as a SYSOUT file. There are two types of SYSOUT file: standard SYSOUT and permanent SYSOUT.

1.11.2 Standard SYSOUT

The standard SYSOUT file (named SYS.OUT) is a system file that is set up during system generation and is stored on the resident disk device. In a given step, the system assigns a subfile of the standard SYSOUT file for each output file produced in the program that is destined for standard SYSOUT. Standard SYSOUT subfiles are automatically printed by the system (by default after the end of the job). When the processing of output is finished, the subfiles are deleted.

A standard SYSOUT subfile is created when the following two conditions apply:

- You have activated the SYSOUT mechanism (see below) in the program or in the JCL.
- There is no assignment (ASSIGN) in the JCL for the given file. The Job Occurrence Report (JOR) is stored in the backing store and retrieved by the Output Writer as if it were a SYSOUT subfile.

1.11.3 Permanent SYSOUT Files

A permanent SYSOUT file is used when you require a SYSOUT file that is not to be deleted after Output Writer activity, or whose contents will not be output until after the current step, or that requires a SYSOUT file on magnetic tape (useful for large volumes of output). If you wish to print a permanent SYSOUT file after the step associated with the file's creation, a WRITER statement must appear in the appropriate job description.

1.11.4 The SYSOUT Mechanism and SYSOUT Format

The SYSOUT mechanism is a method of writing data to a SYSOUT file. You can activate the SYSOUT mechanism by declaring -SYSOUT in the SELECT .... ASSIGN clause of a COBOL program (INPUT-OUTPUT SECTION) or by using the JCL statement SYSOUT. The resulting SYSOUT file is said to be in SYSOUT format. This format incorporates into the file all the output editing functions (for example, margin setting, print density specification, form length) that apply by default (specified at system generation time) or that you have specified (for example, in the DEFINE statement). This format leads to the concept of an "edited" file. The use of the SYSOUT mechanism to produce edited SYSOUT files at job execution time results in a saving of Output Writer activity later when the file is sent to the output device.

If no file assignment appears for a file that is to be written in SYSOUT format, the system automatically assigns a standard SYSOUT subfile. To create a permanent SYSOUT file, you must assign the file (ASSIGN statement) for the step in which the file is written.

The SYSOUT mechanism is always used for writing to the standard SYSOUT file. Whether or not the SYSOUT mechanism is used to create a permanent SYSOUT file depends on the declared record size and record format of the file (see the SYSOUT statement description in Section 3, and the *JCL User's Guide* for more details).

1.12 STORED JCL

One or more job descriptions can be stored as a job stream in a source library. The stored job stream can be maintained in the same way as any source library member, using the LIBMAINT SL facilities (see the *Library Maintenance Manual*). A stored job stream can be introduced to the system by means of the RUN statement or the operator command, EJR. With the $SWINPUT statement, you can introduce stored JCL and/or data records into a job description. This takes place at Stream Reader time.

With the INVOKE statement, you can introduce a stored sequence of step enclosures and/or Extended JCL statements into a job description at JCL translation time. The stored sequence of statements, known as an invoked sequence, replaces the INVOKE statement in the job description.

The EXECUTE statement allows you, during job execution, to translate and then execute a stored sequence of step enclosures and/or Extended JCL statements.

For further details, refer to the *JCL User's Guide*.

1.12.1 Mini-Editor

The Mini-Editor is an updating facility that allows you to modify a sequence of stored JCL statements, or a sequence of JCL statements in an input enclosure, at JCL translation time. A set of special editing commands is provided. The commands required for a particular update must be stored in an input enclosure. The UPDATE parameter of the INVOKE statement gives the name of the input enclosure containing the editing commands, and activates the Mini-Editor to update the contents of the invoked sequence. Refer to the *JCL User's Guide* for more information about using the Mini-Editor.

1.12.2 JCL Parameter Substitution

The same job description (or stored sequence of JCL statements) may have to be used several times with the modification of a few parameters, such as file names or volume names. One way of doing this is to modify every statement in which the parameters to be changed appear. However in large jobs this would require the modification of many statements. To simplify matters, GCOS provides JCL parameter substitution.

The principle of JCL parameter substitution allows you to substitute a value reference that appears in a JCL statement at translation time by a value that you specified elsewhere in the current job. The statements that can supply such values are VALUES, MODVL, RUN, INVOKE and EXECUTE (and the EJR operator command).

1.13 FILE SHARING AND PASSING

In some cases it is useful to access the same physical file (identified by external-file-name) concurrently in one step by means of several logical files (identified by internal-file-name). The same situation can occur when several jobs are running simultaneously. Unless a control of file sharing exists to ensure file security, this situation could result in a file modification that is incompatible with the requirements of another job. The ASSIGN statement allows the SHARE and ACCESS parameters to control the sharing of files.

The concept of file sharing is explained further in the *JCL User's Guide*.

The contents of a temporary file can be preserved from one step to another by means of the END=PASS parameter (or, when a step terminates abnormally, the ABEND=PASS parameter) of the ASSIGN statement. This means that more than one step within a job can access the same temporary file. The mechanism is known as file passing.

For a permanent file, file passing means that access to the file can be reserved from one step to another. While a file is being passed, no other job can assign it.

The concept of file passing is illustrated in the *JCL User's Guide*.

1.14 USE OF OPERATOR COMMANDS

You can include operator commands in a job description, subject to any restrictions imposed on your project. The special JCL statement EXDIR is used for this purpose.

1.15 CHECKPOINT/RESTART MECHANISM

For various reasons, such as a system crash, power failure, device fault or program abort due to an operator error or an I/O error, the processing of a user program may abnormally terminate. The checkpoint and restart mechanisms provide a means of recovery from an abnormal program termination. The checkpoint mechanism is activated by the REPEAT parameter of the STEP statement, or of the $JOB statement, with calls to the checkpoint mechanism from your program. For further details, refer to the *JCL User's Guide,* the *System Administrator's Manual,* and the *GAC-Extended User's Guide*.

1.16 JOURNALIZATION

The checkpoint/restart mechanisms are complemented by journal files.

The BEFORE journal is used in the restart process to return (that is, to roll back) files that were being updated before the abnormal termination to their exact contents at the time the checkpoint process was executed. The journal is required when the repositioning of the file is not sufficient to return the file to the state before abnormal termination.

The AFTER journal is used to return (that is, to roll forward) files that were being updated at the abnormal termination to their state at the time the incident occurred.

You can use the BEFORE, AFTER, BOTH or NONE parameters to specify which journals are to be recorded. The information on the type of journal to be used with a file can be stored in the catalog. Alternatively, you can supply this information via the DEFINE statement.

The DEFINE statement is associated with the relevant file by means of the internal-file-name. The corresponding step may or may not use the checkpoint/restart mechanisms.

2. Rules for Writing JCL Statements

A JCL statement can consist of the following elements:

- a label;
- a statement name;
- one or more parameters.

Each statement has the following format:

```
[label:] ... [$]statement-name [positional-parameter]
[positional-parameter] ... [keyword-parameter] ...
[self-identifying-parameter] ...;
```

2.1 GENERAL RULES

1. The dollar sign ($) is mandatory only for the statements that are handled by the Stream Reader (i.e., $JOB, $ENDJOB, $INPUT, $ENDINPUT, $SWINPUT, $DATA and $ENDDATA), and for any JCL statement that indicates the end of an input enclosure for which END=DOLLAR has been specified (see $INPUT).

NOTES:

1. The dollar sign can be omitted from the first $JOB statement that appears in a stored job stream (handled by the RUN statement or the EJR operator command). However, the recommended practice is never to omit the dollar sign from $JOB.

2. Throughout this manual, and in GCOS 7 documentation in general, the $ is normally omitted wherever a JCL statement name (except for $JOB, $ENDJOB, $INPUT,$ENDINPUT, $SWINPUT, $DATA and $ENDDATA) appears.

2. If a $ is specified, the statement name must immediately follow it.

3. The semicolon (;) is a mandatory terminator for all JCL statements.

4. Most statements contain parameters between the statement name and the terminator. There must be at least one space between the statement name and the first parameter. Each parameter must be separated from the following parameter, if any, either by a comma or by one or more blanks. If a comma is used as a separator, one or more blanks may appear on either side of the comma.

NOTE: In this manual, blanks are used as separators in statement formats. However, in examples where two or more parameters appear on the same line, commas may be used as separators to improve legibility.

5. Except for the Stream Reader statements $JOB, $ENDJOB, $INPUT, $ENDINPUT, $SWINPUT, $DATA and $ENDDATA, any two or more JCL statements can share a record. A Stream Reader statement may be spanned over more than one record; however, it may not share a record with another Stream Reader statement, a JCL statement or an input enclosure record.

6. The statements for which the dollar sign is mandatory must not be labelled (although $ENDJOB can have a label on the previous record).

7. When the dollar sign is mandatory, it must appear as the first character of a record.

8. A statement for which the dollar sign is not mandatory (even if the dollar appears) may start anywhere in a record and may end anywhere in the same or subsequent record.

NOTE: It is often useful to indent certain JCL statements in order to improve the legibility of a job description.

9. Any parameter, keyword or value in a JCL statement may start on a new record. The system recognizes the end of a statement by the semicolon; an end-of-record in a job description is not considered as a separator.

10. Where a statement is continued from one record to the next one, a keyword name or a parameter value can be split across the record boundary. However, in this case:

    - the last character of the first part of the name or value must appear in the last character position of the record;

NOTE: If the parameter value is a protected string, it can also be split by means of a hyphen character (see the paragraph entitled "Protected Strings" below);

    - the first character of the second part of the name or value must appear in the first position of the next record.

11. Parameter substitution is not allowed on the statements for which the dollar sign is mandatory (i.e., $JOB, $ENDJOB, $INPUT, $ENDINPUT, $SWINPUT, $DATA and $ENDDATA).

2.2 COMPONENTS OF JCL STATEMENTS

The components of a JCL statement are:

- the label
- the statement name
- parameters

These components are described below.

2.2.1 Label

Any statement, except $JOB, $ENDJOB, $INPUT, $ENDINPUT, $SWINPUT, $DATA and $ENDDATA, can have one or more labels. Each label can contain up to eight alphanumeric (A-Z, 0-9), hyphen, and underscore characters, and must be followed by a colon (:). The colon can optionally be preceded or followed by one or more blanks.

NOTE: The label CONTINUE, is reserved for special use only as permitted by the rules stated in the description of JUMP (see Section 4). An $ENDJOB statement can have a label, but this label must appear in the preceding record. For example:

```
FINISH:
$ENDJOB;
```

is valid, but

```
FINISH: $ENDJOB;
```

is not valid.

2.2.2 Statement Name

The statement name identifies the action the system takes. A dollar sign is only necessary in the circumstances described in General Rule 1 above.

You must give the statement name exactly as it appears in the format for the statement (see Section 4). At least one blank character must follow the statement name, except where no parameters are given and ";" immediately follows the statement name. When the dollar sign is mandatory, it must be in the first position of the record that contains the statement name. Examples of statement names are:

`$JOB`	identifies a job;
`STEP`	names a load module (for step execution);
`ASSIGN`	assigns a file to a step.

2.2.3 Parameters

Parameters control how the system interprets a particular command. Some statements, such as ENDSTEP, have no parameters. This is an exception rather than the rule.

2.2.3.1 Parameter Types

There are three types of parameter:

- positional parameters;
- keyword parameters;
- self-identifying parameters.

Positional parameters, where used, must be the first parameters that follow the blank(s) after the statement name.

If a comma is used as a separator between two parameters, one or more blanks can follow and/or precede the comma separator. Certain statements can contain a group of parameters enclosed in parentheses, in place of a single parameter. Within a group, parameters must be separated by commas or blanks and positional parameters must be the first parameters that follow the opening parenthesis.

Positional Parameters

The JCL Translator recognizes a positional parameter by its position in the statement. Positional parameters appear immediately after the blank(s) that follow the statement name (or after the opening parenthesis of a parameter group). For example, the following statement has two parameters:

```
ASSIGN INFILE, PAYROLL.MAST;
```

The first value, INFILE, is an internal-file-name, and the second value, PAYROLL.MAST, is an external-file-name. Both these values are positional parameters in the ASSIGN statement, and each must always be specified in the position shown, since that is how the JCL Translator recognizes it.

If a positional parameter value is omitted from a statement or parameter group, two commas must separate the previous parameter from the next parameter (one comma if the first positional parameter of a statement or group is omitted). Alternatively, if blanks are used as separators, a "#" symbol may be used to signify the missing parameter. For example, in the statement:

```
VALUES , P2,, P4;
```

is equivalent to:

```
VALUES # P2 # P4;
```

In the above examples, P2 is the value of the second positional parameter and P4 is the value of the fourth positional parameter; the first and third parameters are missing.

In the statement:

```
VALUES P2 , P4;
```

is equivalent to:

```
VALUES P2 P4;
```

In the above example, P2 is the value of the first positional parameter and P4 is the value of the second one.

If all positional parameters that follow a particular positional parameter are omitted, the corresponding commas or # signs can also be omitted. For example, the statement:

```
VALUES P1,P2,,P4,,,,;
```

or

```
VALUES P1 P2 # P4 # #;
```

can be written:

```
VALUES P1, P2,,P4;
```

or

```
VALUES P1 P2 # P4;
```

A positional parameter can be a single parameter value (e.g., external-file-name), a protected string (see below) or a parameter group.

If only the first positional parameter in a parameter group is required, the parentheses can be omitted provided that this positional parameter is not also a parameter group. For example, the statement:

```
INVOKE BJC, (BIG.YIN);
```

can be written:

```
INVOKE BJC, BIG.YIN;
```

The keyword and self-identifying parameters (described below) must always appear after positional parameters, if they are specified.

Keyword Parameters

A keyword parameter is recognized by its name. Consequently, the position of keyword parameters with respect to other parameters in a statement is unimportant (provided they always follow positional parameters); keyword parameters may appear in any order that is convenient.

A keyword parameter always takes one of the following forms:

- keyword-name = value,
- keyword-name = 'protected-string',
- keyword-name = (parameter-group),

The keyword must appear exactly as given in the appropriate statement format (see Section 4).

One or more blanks can precede or follow the equal sign (). As with positional parameters, a keyword parameter value may be a single parameter value, a protected string or a parameter group. Where only the first positional parameter of a group is required, the parentheses can be omitted provided that this positional parameter is not also a parameter group.

Examples:

```
ASSIGN MFIL1, PAYROLL.MAST, DEVCLASS = MS/M500,
                          MEDIA = VOLA1;
$INPUT MYFILE, TYPE = COBOL, END = ENDINPUT;
```

In the above statements, the following are positional parameter values:

```
MFIL1, PAYROLL.MAST, MYFILE
```

The keyword parameter names are:

```
DEVCLASS, MEDIA, TYPE, END
```

The keyword parameter values are:

```
MS/M500, VOLA1, COBOL, ENDINPUT
```

Self-Identifying Parameters

A self-identifying parameter is recognized by its name. Its position in a statement is unimportant as long as positional parameters, if any, appear first. Unlike a keyword parameter, a self-identifying parameter has no associated value. The name itself indicates to the system the required action. Each self-identifying parameter must be specified exactly as shown in the corresponding statement format in Section 4.

Example:

```
$JOB JB2A, USER = XACC, HOLD;
```

In the statement above, JB2A is a positional parameter value, USER is a keyword parameter whose value is XACC, and HOLD is a self-identifying parameter.

NOTE: In general, the system does not recognize a self-identifying parameter as such, if it appears in place of a positional parameter. Instead, its name is taken as the value of the positional parameter. However, you can avoid this confusing situation, either by placing a keyword parameter before the first self-identifying parameter, or by entering as many commas or # signs as there are missing positional parameters.

For example, the following statements are not equivalent:

```
ASSIGN IFN1, TEMPRY, RESIDENT;

ASSIGN IFN1, RESIDENT, TEMPRY;
```

Neither are the following two statements equivalent:

```
ASSIGN FIL1, DEFER, DEVCLASS = MT/T9, MEDIA = VOLA;

ASSIGN , FIL1, DEFER, DEVCLASS = MT/T9, MEDIA = VOLA;
```

Note that the above statement could be written:

```
ASSIGN # FIL1, DEFER, DEVCLASS = MT/T9, MEDIA = VOLA;
```

2.2.3.2 Default Values

In most cases it is possible to omit one or more parameters from a given JCL statement. Most positional parameters are mandatory, whereas a keyword or self-identifying parameter can often be omitted. Where appropriate, the system automatically assigns a default value for each missing parameter. For example, in the $INPUT statement, the keyword parameters TYPE and END have the default values DATA and ENDINPUT respectively. Therefore, the following two statements are equivalent:

```
$INPUT MYFILE;

$INPUT MYFILE, TYPE = DATA, END = ENDINPUT;
```

2.2.3.3 Protected Strings

A protected string is a string of characters enclosed in single quotes.

The contents of the string are "protected" in the sense that they are not analyzed by the Stream Reader nor by the JCL Translator. Therefore a protected string may contain any character, including the blank character, of the current character set. The quotes are not considered to be part of the protected string.

Examples:

```
'A+B'

'A B'

'SMITH;'
```

If a single quote is required within a protected string, it must appear twice in succession.

Example:

The string

```
ABC'123+;
```

must be specified as:

```
'ABC''123+;'
```

A protected string can be split across record boundaries (if, for example, it is too long to fit on the current record). The split can occur anywhere within a record (that is, not necessarily at the last character position), but the last character of the first part of the string must be followed by a single hyphen (-), and the remainder of the record after the hyphen must consist entirely of blank characters. The first character of the second part of the string must start in the first character position of the subsequent record.

Example:

```
REPORT 'THIS IS AN-
EXAMPLE OF A LONG PROTECTED STRING';
```

3. File Identification and Standard Parameter Groups

This section is divided into two parts. The first part, File Identification, discusses the rules for naming files and, where appropriate, the volumes and devices on which files are stored. The rules apply to all files specified in JCL statements, but particular reference is made here to the Basic JCL Statement ASSIGN. ASSIGN associates the name of a file in a program with the name that the system uses to refer to the file. The second part, Standard Parameter Groups, gives the formats for parameter groups that occur in several extended JCL statements.

3.1 FILE IDENTIFICATION

To uniquely identify a particular file, the system must have the following information:

- the file name;
- the file status;
- the volume name;
- the device identification.

3.1.1 Types of Files

The system supports both permanent and temporary user files. It also supports library files, which may be temporary or permanent, and SYSIN files.

The amount of information that you need to specify, so that the system can find a particular file, depends on whether the file is temporary or permanent, and if permanent, whether it is cataloged or uncataloged.

3.1.1.1 Cataloged, Uncataloged and Temporary Files

Temporary files and uncataloged permanent files are uniquely identified by a combination of the file name, file status, volume identification and device identification. If any one of these components is different from an otherwise identical file identification, then the two files are considered to be different. Therefore, to correctly identify a file, you must specify all the information to the system.

Examples:

```
ASSIGN MYFILE, PAYFILE.TRANS, DEVCLASS=MS/M500, MEDIA=B069;
ASSIGN MYFILE, PAYFILE.TRANS, DEVCLASS=MS/M500, MEDIA=B068;
ASSIGN MYFILE, PAYFILE.TRANS, DEVCLASS=MS/M350, MEDIA=B069;
```

The above statements all refer to different files, because one part of the file identification is different in each case.

However, for cataloged files (which are always permanent), the volume name and device identification (known as "residency parameters") are permanently stored in a special file known as a CATALOG. Therefore, you only need to specify the name of a cataloged file.

The CATALOG also gives the user of a file more control over the means of accessing and updating the file contents. Refer to the *Catalog Management Manual* for a detailed description of catalogs and their use.

When the file is to be allocated as a temporary or cataloged file, the volset syntax may be used instead of supplying the volume name and the device identification. (See Chapter 3 of the *JCL User Guide* for more information.) However, the volset syntax can also be used for a temporary input file.

3.1.1.2 Library Files

A library is a file that has several members, also known as subfiles. Each member is identified by its member name. Libraries can be permanent (cataloged or uncataloged) or temporary.

3.1.1.3 SYSIN Files

The system uses a file called SYS.IN for the temporary storage of the input enclosures of user jobs. Each input enclosure, stored as a temporary "subfile" of SYS.IN, is identified by an input-enclosure-name.

3.1.2 File Names

A file name as discussed in this section is the name by which a file is known to the system (that is, as recorded in file labels). It is known as the "external-file-name".

NOTE: Within a COBOL program, the SELECT . . . ASSIGN clause defines the "internal-file-name". This name, a maximum of eight characters in length, is associated with the external-file-name by means of the ASSIGN statement and is described under ASSIGN.

The rules for naming the following types of file are discussed below:

- Permanent files;
- Temporary files;
- Library files;
- Subfile (member)s;
- Input enclosures;
- Non-standard files.

3.1.2.1 Permanent File Names

The following paragraphs apply in particular to cataloged files, although an uncataloged file also can be named according to the following rules.

The *Catalog Management Manual* gives a more detailed description of the rules for naming cataloged files.

General Rules

A permanent-file-name is a combination of component names and separators. A permanent-file-name for a cataloged file must have the following structure:

```
compound-name.simple-name
```

where compound-name has the form:

```
master-directory-name [.directory-name] . . .
```

GCOS will accept at most 16 alphanumeric, hyphen and underscore characters for a component name (where a component name may be a simple-name, directory-name or master-directory-name). For compatibility with other DPS and Series 60 systems, a component name should have at the most 12 characters, and the underscore character should not be used.

The maximum total length of a permanent-file-name is 33 characters, including all separator characters (period) (). If a file has a generation number, which has a maximum length of 11 characters, the effective maximum length is 44 characters.

The name of a file indicates a hierarchy of directories and files. The master directory is at the top of the hierarchy, which is organized in a tree structure. Each entry at each level of the tree below the master directory is either a directory or a file. Each level of a particular branch of the tree is identified by successive component names in the file name.

For example, if a payroll master file is being processed for the transactions of Section 1 of Department 61 in Plant A, its permanent-file-name could be PLANTA.DEPT61.SECT1.PAYMAST and the tree structure could be of the form shown in Figure 3-1. Here PAYMAST is the simple-name for the lowest entry (that is, the file itself), and SECT1, DEPT61 and PLANTA indicate higher levels of identification of the file.

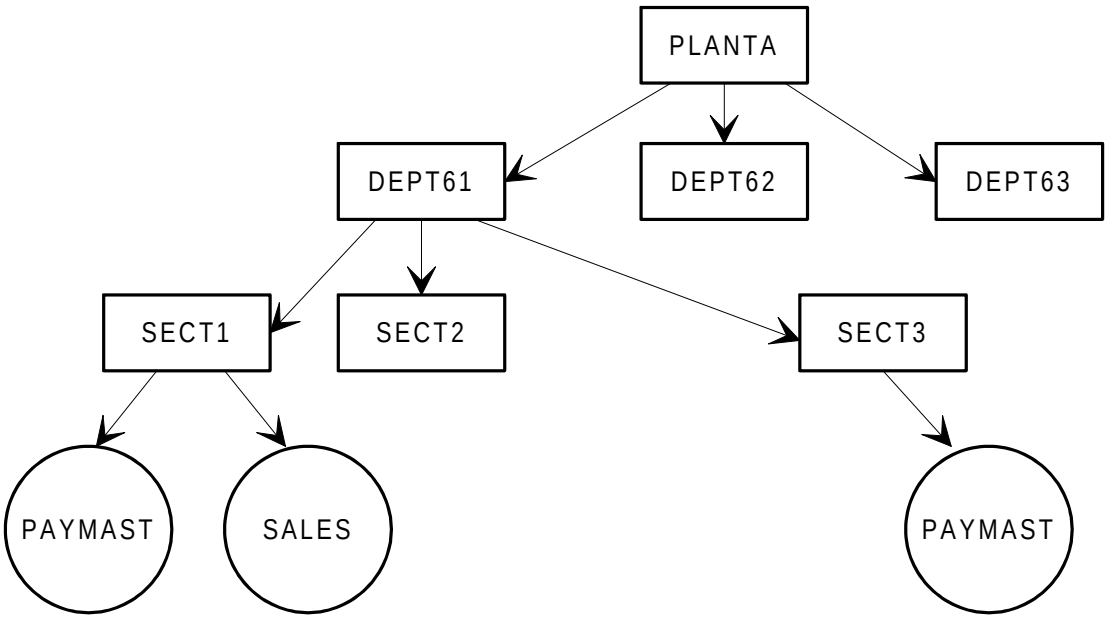


Figure 3-1. Part of a Catalog Tree Structure

A component name must be unique within the immediately higher level component name that points to it. Therefore, two identical names at the same level below a particular component name are not acceptable. However, if the two identical names were below two different component names or at different levels, the names would be accepted. An example of these rules is given in Figure 3-2.

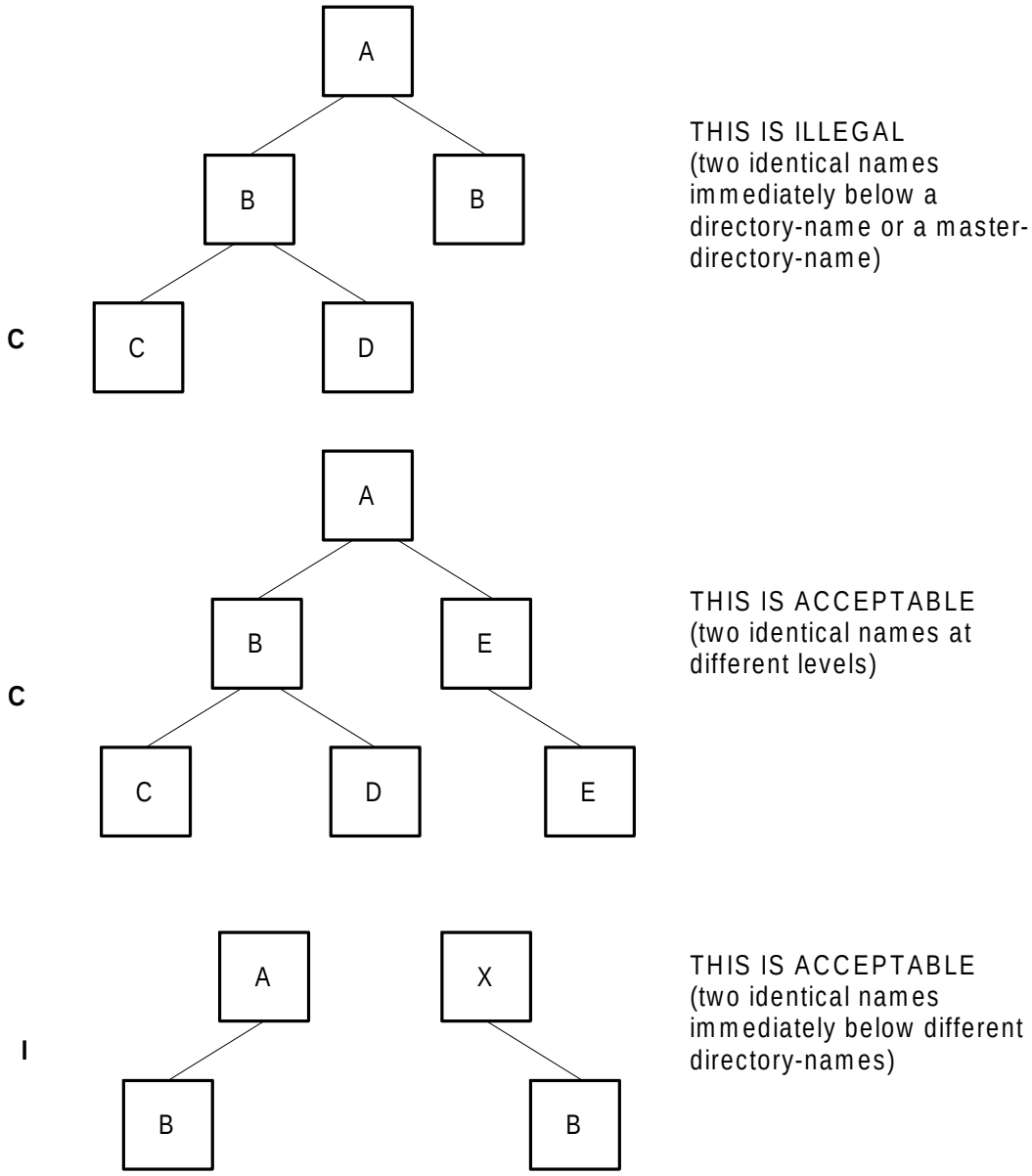


Figure 3-2. Valid and Invalid Construction of tree

The DPS 7000 compatible rules for naming a cataloged permanent file are summarized below:

- The maximum number of characters for a component name is 12.
- A component name must start with a letter (A-Z) and the remaining characters can be any of the following: alphabetic (A-Z), numeric (0-9), hyphen (-).
- The period character (.) is used as a separator between the component names of a permanent file name.
- The maximum number of characters allowed for a permanent file name is 33 (including periods), plus 11 for the generation number (if present).

Tape Files

The maximum length of the name for a permanent ANSI tape file is 17 characters (plus 11 characters for the generation number if the file is cataloged and has a generation number). The other rules for permanent-file-names apply as described above.

Automatic Prefixing

Generally, the master-directory-name is the project-name specified by the users of a file for jobs that access the file. By means of a mechanism called automatic prefixing, you can omit the project identification from an external-file-name of a cataloged file in JCL statements. The system automatically adds the project name for the current job (or the name specified in a previous PREFIX statement) to the beginning of all external-file-names that start with a period character.

For example, if a job runs under the project name AREA3, the ASSIGN statement

```
ASSIGN INFILE, .INTL.STOCK;
```

assigns the cataloged file AREA3.INTL.STOCK to internal-file-name INFILE.

Note that a combination of several component names may be defined as a prefix by means of the PREFIX statement.

Notes on Uncataloged File Names

An uncataloged file exists independently of any catalog or hierarchical structure; it has a unique identification in terms of its name and its residency (volume name and device identification). Therefore it is not important if the name of an uncataloged file does not reflect the tree structure described above for cataloged files. However, it is often convenient to name an uncataloged file as if it were part of a hierarchical structure, because, for example, it may be incorporated into the site catalog or a user catalog later.

Examples:

```
XTRAFILE1A

MY.FILE

HQ.ARCH.MY-FILE
```

Note that the second and third names above are also legal cataloged file names.

Uncataloged files cannot have generation numbers. This means that the length of an uncataloged file name can be up to 44 characters (alphabetic, numeric, hyphen, period characters). However, the recommended practice is to use a maximum of 33 characters, corresponding to the maximum length of a cataloged file name without a generation number.

Multifile Tapes

A standard tape volume that contains more than one file is a multifile tape. The FSN parameter of the ASSIGN statement can be used to give the rank (or relative position) of the file on the tape volume (for example, FSN=3, for the 3rd. file). A multifile tape volume cannot contain a cataloged file. Consequently, a cataloged tape file must reside on a mono-file tape volume. Non-standard multifile tapes are supported with LABEL=NONE and only in INPUT with LABEL=NSTD.

3.1.2.2 Temporary File Names

A temporary file is one that exists for the duration of a step or, at most, a job. At step (or job) termination, the file is automatically deleted by the system.

NOTE: The name of a temporary file must follow the rules of a simple-name (16 characters maximum), except that the maximum file name length of a temporary ANSI tape file is 9 characters.

3.1.2.3 Library File Names and Member Names

The naming rules for library files are identical with those for non-library files, as described above.

The name of a member can contain up to 31 characters (A-Z, 0-9, the underscore and hyphen) and must start with a letter or a digit.

3.1.2.4 Input Enclosure Names

The maximum length of an input-enclosure-name is 16 characters. The name must start with a letter or a digit, and can contain letters, digits, hyphen and underscore characters.

3.1.2.5 Non-standard File Names

File names that do not conform to the above DPS7000 rules are considered as non-standard. Non-standard file names must be enclosed in single quotes (') (for example, 'A+B').

3.1.2.6 Summary of Limits for Name Lengths

Table 3-1 contains a list of various types of file names (and file name components) with the respective maximum permissible lengths, in terms of characters; the recommended maximum length for each name, for reasons of compatibility, is also given. Note that the maximum length for permanent-file-names includes all period separators.

Table 3-1. File Naming - Maximum Lengths

Category	Maximum Length	Recommended Maximum Length
permanent-file-name (cataloged file)	33 (+11 for generation number)	33 (+11 for genereration number)
simple-name	16	12
permanent-file-name (uncataloged file)	44	33
temporary-file-name	16	16
permanent-file-name (ANSI tape file)	17 (+11 for generation number)	17 (+11 for generation number)
temporary-file-name (ANSI tape file)	9	9
member-name (library subfile)	31	31
input-enclosure-name	16	16

3.1.3 File Status

The file status is the term that describes whether a file is cataloged, uncataloged or temporary; you can indicate the file status in the ASSIGN statement as follows:

FILESTAT=CAT for a cataloged file

FILESTAT=UNCAT for an uncataloged file

FILESTAT=TEMPRY for a temporary file

In general, the FILESTAT keyword parameter need not be specified in ASSIGN, since:

- FILESTAT=UNCAT applies by default if residency parameters (DEVCLASS (or DVIDLIST) and MEDIA, or RESIDENT) are specified;
- FILESTAT=CAT applies by default if residency parameters do not appear in the ASSIGN statement, and an entry for the file exists in an attached catalog;
- the self-identifying parameter TEMPRY can always appear in place of the keyword parameter FILESTAT=TEMPRY in ASSIGN.

However, to avoid ambiguity for a file that has been passed from a previous step (since residency parameters are not required here even for an uncataloged or temporary file), it is recommended that the file status should appear in the ASSIGN statement for a file passed from a previous step.

3.1.4 Volume Identification

The following paragraphs apply to uncataloged permanent files and to temporary files. Volume identification is not used on JCL statements for cataloged files except in certain situations involving catalog management (see the *Catalog Management Manual*).

A distinction is made between a volume that is permanently on-line to the system (resident) and one that is loaded on a device only when it is needed (non-resident).

3.1.4.1 Resident Volume

A resident volume is always a disk, and is identified by the RESIDENT parameter. No device identification is necessary.

Examples:

```
ASSIGN PAY, PAY.MASTER, RESIDENT;
```

The uncataloged permanent file PAY.MASTER resides on a resident disk volume. Note that FILESTAT=UNCAT applies by default where RESIDENT appears.

3.1.4.2 Non-resident Volume

For a non-resident disk volume, cartridge volume or a tape volume, the volume name (which is recorded in the volume label) must be specified using the MEDIA parameter.

Example:

```
MEDIA=B016
```

The volume name can contain alphanumeric, hyphen and underscore characters and has a maximum of six characters. Note that the keyword WORK may be specified to indicate that work tapes or cartridges are to be used.

Example:

```
MEDIA=WORK
```

Work tapes or cartridges are prepared using the VOLPREP or TAPEPREP utility (see Section 5).

In certain cases, the identification of a volume may include the LABEL parameter to identify the type of volume label. It can be written, for example, as follows:

```
LABEL=NATIVE
```

for a volume with standard GCOS 7 label organization (default value).

```
LABEL=NONE
```

for a non-labelled volume.

(See ASSIGN for other values for LABEL.)

Note that the MEDIA and LABEL parameters may only be specified with a device identification parameter (DEVCLASS or DVIDLIST, described below).

The LABEL parameter of the ASSIGN statement is further described in Section 4.

3.1.4.3 Non-standard Volume Names

Volume names that do not conform to the DPS 7000 rules above are considered non-standard. Non-standard volume names must be protected, that is, enclosed in single quotes ('); (for example, 'A+B').

Non-standard volume names containing non-printing characters (for example, binary zeros) cannot be referenced. To use such a device, you must assign it with:

```
MEDIA=*,
DVIDLIST=(device_name)
```

For example, to mount a tape with a non-standard name on drive MT02, the following statement is used:

```
ASSIGN MYIFN, * , MEDIA=*, DVIDLIST=(MT02);
```

For further details of the ASSIGN parameters, refer to Section 4.

3.1.5 Device Identification

The following paragraphs apply to uncataloged permanent files and to temporary files. Device identification is not used in JCL statements for cataloged files except in certain situations involving catalog management (see the *Catalog Management Manual*).

A device is identified by one of the following:

- Device class and attributes (DEVCLASS parameter);
- Device name (DVIDLIST) parameter.

3.1.5.1 Device Class and Attributes

A device class designates a device, such as a disk drive or tape or cartridge drive. It may be qualified by device attributes that further specify a particular type of device within a class, or request some specific technical option or setup.

The general form for coding the device class and its attributes is:

```
XX[/ attribute] . . .
```

where XX are two alphabetic characters that identify the type of device, as follows:

CT	cartridge tape device
MS	disk device (mass storage)
MT	magnetic tape device
PR	printer
TN	terminal

When XX is used alone, it refers to any device within that class, despite its attributes.

WARNING

The device class MS must be specified with its corresponding attributes. Otherwise, the system will abnormally terminate the step.

For other cases, additional attributes can be omitted if there is no ambiguity, and if no specific characteristics are requested.

Note the following for cartridge devices:

For 8mm cartridges - M6 can be omitted if all cartridge devices use 8mm cartridges.

For half-inch cartridges - For a device that does not belong to a library, M5 or 36T can be omitted if all cartridge devices use half-inch cartridges, with the dame recording format (18- or 36-track), and if there is no library.

For a device that belongs to a library, M5 or 36T can be omitted if all devices that belong to the library use the same recording format (18- or 36-track).

Nevertheless, it is recommended to use the full device class, as this will help to avoid the necessity of modifying jobs or catalogs if new devices are introduced on the site, or if the job is launched on another site.

A list of the valid device classes is given in Paragraph 3.1.5.2.

Examples:

```
ASSIGN INF3, PAYDATA, TEMPRY, DEVCLASS=MS/D500, MEDIA=1234;
```

The internal file INF3 is associated with the temporary file PAYDATA, which is stored on a 500MB disk volume with volume-name 1234.

```
ASSIGN INC, MY.TAC, DEVCLASS=MT/T9/D1600, MEDIA=BO16;
```

This requests file assignment for an uncataloged file MY.TAC stored on a 9-track, 1600 b.p.i. tape volume, with volume-name BO16. FILESTAT=UNCAT applies by default where DEVCLASS and MEDIA appear.

3.1.5.2 Valid Device Classes

Cartridge Tape Devices (not belonging to a cartridge library)

Note that no attributes distinguish a device with a sequential cartridge loader.

For 8mm cartridges:

```
CT [/M6] [/{S35|S75}]
```

where:

S35	specifies a recording density of 35 Mb/sq. inch
S75	specifies a recording density of 75 Mb/sq. inch.

For half-inch cartridges:

```
CT [/{M5|36T}]  [/C]
```

where:

M5	specifies that the device uses the 18-track recording format (only standard length tapes are supported). Note that the use of a 36-track formatted cartridge on such a device must specify M5. The only possible use is volume preparation of a standard length tape (which always becomes a 18-track formatted cartridge).
36T	specifies that the device uses the 36-track recording format. No attributes distinguish standard length tapes from E-tapes (both are supported). Note that the use of a 18-track formatted cartridge on such a device must specify 36T. Possible users are reading, volume preparation (the cartridge always becomes a 36-track formatted cartridge) or complete rewriting (always with the 36-track recording format).
C	specifies compaction. Note that: - all devices that use the 36-track recording format have compaction capability. - to read a cartridge written in compaction mode, C must be specified (except when on the site, all devices of the specified type support compaction). - to write on a cartridge in compacted mode, C must be specified. - to read a cartridge written in non-compacted mode, C can be specified or not (if C is specified, the operation is performed on a device with the compaction capability; if C is not specified, the device may or may not have the compaction capability). - to write on a cartridge in non-compacted mode, C must be omitted (but the operation can be performed on a device with the compaction capability).

Examples:

`CT`	Cartridge Tape
`CT/M6`	8mm Cartridge Tape
`CT/M6/S35`	8mm Cartridge Tape with 35 Mb/sq. inch recording density
`CT/M5`	Half-inch Cartridge Tape (device using 18-track recording format).
`CT/36T/C`	Half-inch Cartridge Tape (device using 36-track recording format), with compaction.

Cartridge Library (half-inch cartridges only)

```
CT/LIB [/ {M5 | 36T}] [/C]
```

where:

M5, 36T, and C have the same meaning as for half-inch cartridges not belonging to a cartridge library.

Examples:

`CT/LIB`	Cartridge Tape belonging to a library
`CT/LIB/C`	Cartridge Tape belonging to a library, with compaction.
`CT/LIB/M5/C`	Cartridge Tape belonging to a library (device using 18-track recording format), with compaction.
`CT/LIB/36T`	Cartridge Tape belonging to a library (device using 36-track recording format).

Mass Storage Devices

```
MS/{FSA|B10|D500}
```

Examples:

`MS/FSA`	Fixed Sector Architecture disk drive.
`MS/B10`	1 Gigabyte disk drive.
`MS/D500`	500MB disk drive.

Magnetic Tape Devices

`MT[/T9] [/Dnnnn] [/S]`

`T9`	indicates the number of tracks (9).
`Dnnnn`	specifies the density (D1600 or D6250).
`S`	specifies that the tape is to be processed in "streaming" mode. Streaming leads to faster processing if the treatment is performed without stopping the tape. It is a processing mode, not a characteristic of the tape volume concerned. S is meaningful only if the concerned command is FILREST, FILSAVE, VOLREST or VOLSAVE. In all other cases, start/stop mode is used.

Examples:

`MT/T9`	9-track unspecified density Magnetic Tape.
`MT/D1600`	9-track, 1600 bpi Magnetic Tape.
`MT/T9/D6250`	9-track, 6250 bpi Magnetic Tape.
`MT/T9/D6250/S`	9-track, 6250 bpi Magnetic Tape in streaming mode.
`MT/S`	9-track unspecified density Magnetic Tape in streaming mode.

Printers

`PR [/speed] [/Hnnn] [/model]`

`speed`	speed attribute (FI, A or SI). See note 2 below.
`Hnnn`	Printer device supporting at least "nnn" print positions (for example, PR/H125 requests a printer with 132, or 136, or 160 hammers, but not one with 120 hammers).
`model`	Printer model (PR54, PR90).

Examples

`PR` - any printer.

`PR/H123` - a printer with at least 123 hammers.

NOTES:

1. If the Hnnn attribute is missing, any available printer is requested (conforming to any other attributes specified - see below);

2. A speed attribute can be specified: SI indicates the slowest printer available with the specified hammer requirement, if any; FI indicates the fastest printer available with the specified hammer requirements, if any; A specifies any speed.

3. A particular device type can be specified (for example, PR54).

Terminals

```
TN (used in ::TN for file terminal)
```

3.1.5.3 Device Name

The device-name is the name of one device. Device names are assigned to devices at system generation. The general form of a device-name is:

XXYY

where XX is the device type (MT, MS, etc.), and YY is a two-character (alphanumeric) identifier

Device-names must not be specified for resident volumes or mirror disks. Also, no device reconfiguration is possible if a device-name has been specified.

Example:

```
CT01
```

NOTE: The device-name parameter is used to request a particular device. If that device is in use by another job (that is, not currently available) the request for the device is enqueued until the device becomes available. The use of the DEVCLASS parameter (described above) is therefore recommended as standard practice.

3.2 STANDARD PARAMETER GROUPS

Certain groups of parameters occur in several extended JCL statements.

Rather than repeat them in each statement format, they are listed in full in this section and given a standard parameter group name. This name is used in each subsequent reference to the group both in statement formats and in the text. Other manuals that discuss JCL statements also use these names to refer to the parameter group. The notation defined in Table 3-2 is used to describe the value(s) that individual parameters may take, and it is used throughout this manual.

Table 3-2. Parameter Value Notation Parameter

Notation	Description
alphanum	string of alphanumeric characters (A...Z, 0...9, hyphen, underscore) The first character cannot be a hyphen or an underscore.
alphanumN	alphanum of maximum length N
alphanum_N	alphanum of length N
bit	string of binary digits (0, 1)
bitN	bit of maximum length N
bit_N	bit of length N
digits	string of decimal characters (0...9)
digitsN	digits of maximum length N
digits_N	digits of length N
hexa	string of hexadecimal characters (0...9, A...F)
hexaN	hexa of maximum length N
hexa_N	hexa of length N
identifier	string of alphanum, beginning with a letter (A...Z)
identifierN	identifier of maximum length N
identifier_N	identifier of length N
octal	string of octal characters (0...7)
octalN	octal of maximum lenght N
octal_N	octal of length N
string	string of any characters
stringN	string of maximum length N
string_N	string of length N

Using the notation defined in Table 3-2, the syntax of names used in JCL statements is summarized in Table 3-3.

Table 3-3. Naming Conventions for JCL Names

Notation	Description
billing-name	alphanum12
catalog-name	simple-name [.simple-name]. CATALOG
external-file-name	qualified-name (length < = 44) protected string
input-enclosure-name	alphanum16
internal-file-name	alphanum8
job-name	alphanum8
label-name	alphanum8
load-module-name	alphanum31
master-directory-name	simple-name
member-name	alphanum31
project-name	alphanum12
qualified-name	simple-name [.simple-name]...
simple-name	alphanum16
star-name	alphanum31 where unspecified parts are replaced by the * character.
temporary-file-name	simple-name
user-name	alphanum12
volume-name	alphanum6 (length < = 6) protected string

The standard parameter-group names and their formats are listed in the following paragraphs.

3.2.1 Sequential-input-file-description

Format

```
{input-enclosure-name}    [{FILESTAT  =  {CAT     }}]
{external-file-name  }    [{             {UNCAT   }}]
{         *          }    [{             {TEMPRY  }}]
                          [{TEMPRY                 }]

[{CATALOG  =  digit 1                                              }]
[{RESIDENT                                                         }]
[{DEVCLASS =  device-class MEDIA = {WORK                         }}]
[{                                 {*                            }}]
[{                                 {(volume-name [volume-name]...) }}]
[{VOLSET = {volset-name|DFLT}                                      }]

[SUBFILE  =  member-name]

[SHARE    =  {NORMAL  }]     [ACCESS   = {WRITE   }]
[            {ONEWRITE}]     [           {SPWRITE }]
[            {FREE    }]     [           {READ    }]
[            {MONITOR }]     [           {SPREAD  }]
[            {DIR     }]     [           {RECOVERY}]
[                     ]     [           {ALLREAD }]

[FSN      =  {digits3}]
[            {ANY    }]
[            {NEXT   }]

[NBEFN    =  {digits3}]   [FIRSTVOL  =  {digits3}]
[            {ALL    }]   [              {EOF    }]

[LASTVOL  =  {digits3}]   [END       =  {DEASSIGN}]
[            {EOF    }]   [              {PASS    }]
                          [              {LEAVE   }]
                          [              {UNLOAD  }]

[ABEND    =  {DEASSIGN}]
[            {PASS    }]   [MOUNT     =   digits2]
[            {LEAVE   }]
[            {UNLOAD  }]

[POOL     [{FIRST}]]   [LABEL     =  {NATIVE }]
[         [{NEXT }]]   [             {NONE   }]
                       [             {NSTD   }]
```

Comments and Restrictions

SYSOUT or DUMMY cannot be external-file-names. The file referenced must be a sequential file. If SUBFILE appears, the default for SHARE is DIR. The default value of ACCESS is READ (compare with the ASSIGN statement where the default is WRITE). Otherwise, the parameters and their values are as described under the ASSIGN statement.

The external-file-name may be TEMP, TEMP1 or TEMP2 but in such a case the SUBFILE parameter must be specified and no other parameter should be given.

3.2.2 Input-file-description

This parameter group is identical with sequential-input-file-description except that the file referenced is not restricted to being a sequential file.

3.2.3 Sequential-output-file-description

Format

```
{                   }        [{FILESTAT  = { CAT  }}]
{external-file-name}         [{            {UNCAT }}]
{DUMMY              }        [{            {TEMPRY}}]
                             [{TEMPRY                }]

[{CATALOG   =  digit1                                                 }]
[{RESIDENT                                                            }]
[{DEVCLASS  =  device-class  MEDIA  =  {WORK                        }}]
[{                                     {*                           }}]
[{                                     {(volume-name [volume-name]...)}}]
[{VOLSET  =  {volset-name | DFLT }                                    }]

                             [FSN      = {digits3}]
                             [           {ANY    }]
[SUBFILE    =  member-name]  [           {NEXT   }]

[SHARE      =  {NORMAL  }]   [ACCESS   = {WRITE   }]
[              {ONEWRITE}]   [           {SPWRITE }]
[              {FREE    }}   [           {READ    }]
[              {DIR     }]   [           {SPREAD  }]
[              {MONITOR }]   [           {RECOVERY}]
[                       ]    [           {ALLREAD }]

[END        = {DEASSIGN}]    [ABEND    ={DEASSIGN}]
[             {PASS    }]    [          {PASS    }]
[             {LEAVE   }]    [          {LEAVE   }]
[             {UNLOAD  }]    [          {UNLOAD  }]

[LABEL      = {NATIVE }]     [MOUNT    = digits2]
[             {NONE   }]
[             {NSTD   }]     [FIRSTVOL ={digits3}]
                             [           {EOF    }]

                             [LASTVOL  ={digits3}]
                             [           {EOF    }]

[                       ]    [DENSITY  ={D1600}]
[EXPDATE    ={ddd     }]     [          {D6250}]
[             {yy/ddd  }]    [          { S35 }]
[             {yy/mm/dd}]    [          { S75 }]

                              [POOL [{FIRST}]]
[SIZE       = digits3]        [     [{NEXT }]]
                              [              ]

[CATNOW]  [CATALOG=digit1]
```

Comments and Restrictions

The meaning of each parameter in the group, except SIZE, is given in the ASSIGN statement. The default value of ACCESS is WRITE. SIZE specifies the number of disk cylinders to be allocated to the file and is permitted for temporary files only.

3.2.4 Output-file-description

This parameter group is identical to sequential-output-file-description except that the file referenced is not restricted to being a sequential file.

3.2.5 Print-file-description

The parameters in this group are identical to those in sequential-output-file description except that SYS.OUT may be used as an external-file-name whereas TEMP, TEMP1 or TEMP2 may not be used.

3.2.6 Work-file-description

Format

```
                        [{FILESTAT = {CAT   }}]
                        [{           {UNCAT }}]
external-file-name      [{           {TEMPRY}}]
                        [{TEMPRY             }]


[{CATALOG  = digit1                                                  }]
[{RESIDENT                                                           }]
[{DEVCLASS = device-class MEDIA = {WORK                            }}]
[{                                {(volume-name [volume-name]...)}}]
[{VOLSET = {volset-name | DFLT}                                      }]


[SHARE     = {NORMAL  }] [ACCESS  = {WRITE  }]
[            {ONEWRITE}] [          {SPWRITE}]
[            {FREE    }] [          {READ   }]
[            {DIR     }] [          {SPREAD }]
[            {MONITOR }] [          {ALLREAD}]

[LABEL    =   NATIVE]     [SIZE   =  digits3]

[END       = {DEASSIGN}]  [ABEND   = {DEASSIGN}]
[            {PASS    }]  [          {PASS    }]

[EXPDATE   =  {ddd     }]
[             {yy/ddd  }]
[             {yy/mm/dd}]
```

Comments and Restrictions

The SIZE parameter can be specified only if the file is temporary and it specifies the number of disk cylinders to be allocated to the file.

Otherwise the work file must already have been preallocated (for example, using the PREALLOC utility). The other parameters are described under the ASSIGN statement.

3.2.7 Input-library-description

Format

```
{                  }    [{FILESTAT = {CAT   }}]
{external-file-name}    [{           {UNCAT }}]
{TEMP              }    [{           {TEMPRY}}]
{TEMP1             }    [{                   }]
{TEMP2             }    [{TEMPRY             }]

[{CATALOG =  digit1                                              }]
[{RESIDENT                                                       }]
[{DEVCLASS=  device-class  MEDIA=(volume-name [volume-name]...)}]
[{VOLSET = {volset-name | DFLT}                                  }]

[SHARE    = {NORMAL  }]  [ACCESS   = {WRITE  }]
[           {ONEWRITE}]  [           {SPWRITE}]
[           {FREE    }]  [           {READ   }]
[           {DIR     }]  [           {SPREAD }]
[                    ]  [           {ALLREAD}]

[END      = {DEASSIGN}]  [ABEND    = {DEASSIGN}]
[           {PASS    }]  [           {PASS    }]

[LABEL  = NATIVE]
```

Comments and Restrictions

SYSOUT, DUMMY and input enclosures may not be specified. If the external-file-name is TEMP, TEMP1 or TEMP2 then FILESTAT=TEMPRY is forced, and END, ABEND both default to PASS instead of DEASSIGN. The other parameters are as described under ASSIGN.

3.2.8 Output-library-description

Format

```
                          [{FILESTAT = {CAT   }}]
{external-file-name}      [{           {UNCAT }}]
{TEMP               }     [{           {TEMPRY}}]
{TEMP1              }     [{                   }]
{TEMP2              }     [{TEMPRY             }]

[{CATALOG = digit1                                              }]
[{RESIDENT                                                      }]
[{DEVCLASS=device-class   MEDIA=(volume-name [volume-name]...)}]
[{VOLSET = {volset-name | DFLT }                                }]

[SHARE   = {NORMAL  }]   [ACCESS    = {WRITE   }]
[          {ONEWRITE}]   [             {SPWRITE }]
[          {FREE    }]   [             {READ    }]
[          {DIR     }]   [             {SPREAD  }]
[                    ]   [             {RECOVERY}]

[END     = {DEASSIGN}]   [ABEND     = {DEASSIGN}]
[          {PASS    }]   [             {PASS    }]

[LABEL  = NATIVE]         [SIZE     =  digit3]

[EXPDATE  = {ddd     }]
[           {yy/ddd  }]
[           {yy/mm/dd}]
```

Comments and Restrictions

SYSOUT or DUMMY may not be specified. If the external-file-name is TEMP, TEMP1 or TEMP2 then FILESTAT=TEMPRY is forced, and both END and ABEND default to PASS instead of DEASSIGN. The default of ACCESS is WRITE. SIZE gives the number of disk cylinders to be allocated, for temporary files only. The other parameters are as described under ASSIGN.

3.2.9 Print-library-description

This parameter group is identical with output-library-description. For interactive processing, TEMP TEMP1 and TEMP2 can be specified; if no value is specified, TEMP applies by default.

3.2.10 Define-parameters

These are the parameters of the basic JCL statement, DEFINE. For the format and description, refer to the DEFINE statement.

3.2.11 Sysout-parameters

These are the parameters of the basic JCL statement, SYSOUT. For the format and description, refer to the SYSOUT statement.

3.2.12 Size-parameters

These are the parameters of the basic JCL statement, SIZE. For the format and description, refer to the SIZE statement.

3.2.13 Step-parameters

These are parameters of the STEP basic JCL statement. For the format and description, refer to the STEP statement. The parameters OPTIONS, DEBUG and DUMP=ALL are not valid step-parameters.

4. Basic JCL Statements

4.1 HOW STATEMENTS ARE PRESENTED

This chapter describes each of the Job Control Language Statements, in alphabetical order. Information about each statement is organized under the following six headings:

1. Function.
2. Enclosure.
3. Format.
4. Description of Statement.
5. Parameters (when they exist).
6. Examples.

4.1.1 Function

Briefly describes what the statement does.

4.1.2 Enclosure

Tells you what enclosure level the statement belongs to. This can be job, step, or input.

4.1.3 Format

Contains the full specification for the basic JCL statement. The format model for each statement is written in a typeface different from the body of the text, and the following notation conventions are used:

`UPPERCASE`	The keyword item is coded exactly as shown.
`lowercase`	Indicates a value which the user must supply.
`[item]`	An item within square brackets is optional.

`{item1}`
`{item2}`
`{item3}`
	A column of items within braces indicates that one value must be selected if the associated parameter is specified. If the parameter is not specified, the underlined item is taken as the default value.

`( )`	Parentheses must be coded if they enclose more than one positional item.
`...`	An ellipsis indicates that the preceding item may be repeated one or more times.

4.1.4 Description of Statement

Gives a more detailed description of the statement. It may also include any pertinent comments and restrictions, which are listed under the sub-heading: Comments and Restrictions.

4.1.5 Parameters

Where parameters exist, there is a description of each one. Mandatory parameters are described first, under the sub-heading: Mandatory Parameters.

Optional parameters follow, under the sub-heading: Optional Parameters.

4.1.6 Examples

Finally, some examples are normally given to illustrate how the statement should be written.

4.2 ALLOCATE

4.2.1 Function

Requests space allocation for a disk or diskette file and/or space extension whenever the existing space is used up.

4.2.2 Enclosure

Step.

4.2.3 Format

```
ALLOCATE     internal-file-name

            {SIZE        =   digits10                     }
            {                                             }
            {INCRSIZE    =   digits5                      }
            {                                             }
            {SIZE        =   digits10  INCRSIZE  =  digits5}

            [UNIT       = {TRACK   }]
            [             {TRK     }]
            [             {CYLINDER}]
            [             {CYL     }]
            [             {RECORD  }]
            [             {BLOCK   }]
            [             {100KB   }]

            [CHECK]

            [KEEP] ;
```

4.2.4 Description of Statement

The ALLOCATE statement requests disk space allocation for a temporary or permanent uncataloged file that is assigned to a step. In addition, for UFAS sequential and indexed files, and library files, whenever the allocated space becomes exhausted, ALLOCATE may request file extension. If all the space currently allocated to the file is occupied and more is required in the current step, the file extension mechanism provides more space. Each ALLOCATE statement must always be associated (by the internal-file-name) with an ASSIGN statement in the same step enclosure.

If an ASSIGN statement exists for a temporary file which is to be created, but no corresponding ALLOCATE statement exists, a default ALLOCATE statement with one cylinder of space is assumed. Where appropriate, a one-cylinder extension space is also assumed in this statement.

If several ASSIGN statements in a step refer to the same permanent-file-name, the ALLOCATE statement is only valid when associated with the internal-file-name of the first ASSIGN. See the ASSIGN statement for further details of multiple assignments of the same file in a given step.

An ALLOCATE is necessary when the CATNOW parameter is used in ASSIGN and when the cataloged file to be created resides on disk. (Not applicable to tape).

Space allocation becomes effective only when the file named in ALLOCATE (internal-file-name) is opened, and for diskette files only, the file must be opened in output mode. If no attempt is made to open the file during the step execution, no space allocation is made. Attempts to access the file before its effective allocation will result in an abnormal return code.

Space allocation for permanent files should normally be requested with the PREALLOC utility (see Section 5). In particular, ALLOCATE cannot be used for IDS files. Space allocation for permanent library files should normally be requested with the LIBALLOC statement.

When INCRSIZE is specified, files are dynamically extended, if necessary, during write operations (that is, extension is not done at file open time). This parameter does not apply to diskette files.

An attempt to allocate space for a file which already exists will normally produce the return code ALREADY when the file is opened, and the ALLOCATE will be ignored. However, if the CHECK parameter has been specified in the ALLOCATE statement, the step will be aborted.

4.2.5 Parameters

4.2.5.1 Mandatory Parameter

INTERNAL-FILE-NAME

The name of the file for which space is allocated. It matches the file specified in the related ASSIGN statement.

The name must not exceed eight alphanumeric, hyphen and underscore characters. It must start with either a number or a letter.

4.2.5.2 Optional Parameters

SIZE

Specifies the size of the file in terms of the number of units of space allocation (see UNIT, below). Its default value is 0. This parameter must appear if INCRSIZE is not specified.

INCRSIZE

Specifies the number of units of space allocation (see UNIT below) that are added to an output file every time automatic space extension is performed on it in the current step. It must appear if SIZE is not specified. The default value is 0; that is, there is no automatic extension if INCRSIZE is not specified. This parameter does not apply to diskette files.

UNIT

Specifies the unit of space allocation or extension as either a TRACK, CYLINDER, RECORD, BLOCK, or 100KB. The default unit is a TRACK.

The following abbreviations may also be used:

TRK (for track)
CYL (for cylinder)

CHECK

If a file with the same external file name as the one specified in the associated ASSIGN statement already exists, the use of the CHECK parameter will cause the current step to abort at file open time. If CHECK is not specified in such a case the return code ALREADY is set.

KEEP

If KEEP is specified, the allocation parameters are kept and used again for any file that is assigned using the same internal-file-name.

Examples:

```
ALLOCATE BAC, SIZE=20, UNIT=CYL;

ALLOCATE FULL, INCRSIZE=2;
```

4.3 ASSIGN

4.3.1 Function

Makes a file known and accessible to a step and allocates the resources associated with the file to the step.

4.3.2 Enclosure

Step.

4.3.3 Format

```
ASSIGN      internal-file-name

            {external-file-name   }
            {temporary-file-name  }
            {*input-enclosure-name}
            {DUMMY                }
            {*                    }

            [{FILESTAT  = {CAT   }}]
            [{            {UNCAT }}]
            [{            {TEMPRY}}]
            [{                    }]
            [{TEMPRY              }]

            [CATALOG  =  digit1]

[CATNOW ]

[EXPDATE   =  {ddd     }]
[             {yy/ddd  }]
[             {yy/mm/dd}]

[{VOLWR }]
[{NVOLWR}]

[OPTIONAL]

[DEFER]

[{RESIDENT                  }]
[{device-class-description  }]
[{device-identification-list }]
[{VOLSET = {volset-name|DFLT}}]

[SUBFILE  =  member-name]
```

```
[SHARE     =  {NORMAL  }]
[             {ONEWRITE}]
[             {FREE    }]
[             {MONITOR }]
[             {DIR     }]

[ACCESS    =  {WRITE   }]
[             {READ    }]
[             {SPWRITE }]
[             {SPREAD  }]
[             {RECOVERY}]
[             {ALLREAD }]

[END       =  {PASS    }]
[             {DEASSIGN}]
[             {LEAVE   }]
[             {UNLOAD  }]

[ABEND     =  {PASS    }]
[             {DEASSIGN}]
[             {LEAVE   }]
[             {UNLOAD  }]

[ENDVOL    =  {UNLOAD  }]
[             {REWIND  }]
[             {LEAVE   }]

[MOUNT     =   digits2]

[POOL         [{FIRST}]]
[             [{NEXT }]]

[LABEL     =  {NATIVE}]
[             {NONE  }]
[             {NSTD  }]

[FIRSTVOL  =  {digits3}]
[             {EOF    }]

[LASTVOL   =  {digits3}]
[             {EOF    }]

[FSN       =  {digits3}]
[             {ANY    }]
[             {NEXT   }]

[NBEFN     =  {digits3}]
[             {ALL    }]

[DENSITY   =  {D1600 }]
[             {D6250 }]
[             {S35   }]
[             {S75   }];
```

Where the device-class-description has the format:

```
DEVCLASS   =   device-class

MEDIA      =  {WORK                                }
              {(volume-name   [volume-name]  ...)}
              {*                                  }
```

and the device-identification-list has the format:

```
DVIDLIST   =  (device-name   [device-name] ... )

MEDIA      =  {WORK                                 }
              {(volume-name [volume-name] ... )}
              {*                                   }
```

4.3.4 Description of Statement

The first function of the ASSIGN statement is to make a file known and accessible to a step. This is done by relating the file's "internal-file-name" (defined in the program) to its "external-file-name", which is the name by which the file is known to the file system.

The second function of this statement is to allocate to the step the resources (device, media) that are associated with the file. Devices may be optionally allocated from a device pool (see POOL statement).

The file to be assigned may be an input enclosure (SYSIN subfile, deleted at job termination) or it may be a permanent file (cataloged or uncataloged) or a temporary file. A cataloged file may only be accessed through the catalog mechanism (see the description of the ATTACH statement in the Catalog Management User's Guide). In this case, the residency parameters (RESIDENT, DEVCLASS or DVIDLIST, and MEDIA) must not be specified. For an uncataloged or temporary file, you must declare the associated volume and device by means of the residency parameters.

If the external-file-name begins with the characters SYS. then the assignment is to a system file; in this case, or if the value of the second positional parameter begins with an asterisk (for an input-enclosure-name) or is DUMMY, residency parameters must not appear. One or more residency parameters (for example RESIDENT, or DEVCLASS and MEDIA) indicate that the file is permanent and uncataloged, unless, in addition, the parameter TEMPRY (or FILESTAT=TEMPRY) appears, in which case the file is taken to be temporary. The CATALOG parameter indicates a permanent cataloged file.

If the file name does not correspond to a system file, an input-enclosure, nor a dummy file, and if the residency, FILESTAT, TEMPRY and CATALOG parameters are missing from an ASSIGN statement, the following occurs:

- the system first of all assumes that the file is permanent and cataloged, and it applies the search rules of the catalog mechanism in the following order:

  1. the catalogs which are explicitly attached to the job,
  2. the site catalog (SITE.CATALOG),
  3. the system catalog (SYS.CATALOG),
  4. the catalogs which are registered system-wide as auto-attachable;

- if the search fails, file management then assumes that it is an uncataloged permanent file that has been passed from a previous step (see the PASS parameter below);

- if there is no passed uncataloged file with the same external-file-name, the file is assumed to be resident and uncataloged.

For identification purposes, the system automatically adds a string of characters (which contain the Run Occurrence Number of the current job) to the beginning of each temporary-file-name. Although this compound name may appear in the Job Occurrence Report, you must only use the simple temporary-file-name.

Normally the space for a permanent disk file will already have been preallocated by means of the data management utilities (PREALLOC, FILALLOC, LIBALLOC, ...). However, if a new disk file is being created in the current step, an ALLOCATE statement must be included in the same step enclosure. For a temporary file, the ALLOCATE statement may be omitted (see the ALLOCATE statement). For a tape file, no space allocation request is necessary although PREALLOC is required for the catalog entry of a cataloged tape file.

The ASSIGN statement also enables a unit record device to be directly assigned to a step. For example, if a printer is assigned, then printing takes place during step execution, thus, the programmer decides not to use the SYSOUT mechanism (see the SYSOUT statement). The file system does not allow simultaneous requests for a given volume using both the device-name (DVIDLIST) and the device-class (DEVCLASS).

NOTE: The system automatically adds a prefix to the beginning of all external-file-names that begin with a period (see Section 3).

Multiple Assignments of the Same File in a Given Step

Several ASSIGN statements may exist in the same step for a given permanent-file-name, but each with a different internal-file-name. In this case, the external file identification must be unique and specify (or accept the default value for): FILESTAT, LABEL, RESIDENT, DEVCLASS and MEDIA.

The POOL, END and ABEND parameters must be identical, if they are specified.

The MOUNT option must not be specified when multiple assignments of the same file occur in a given step. All volumes must be mounted.

If SHARE and ACCESS are specified, they must be compatible with the specifications of the corresponding ASSIGN statements. So, too, must be the EXPDATE parameter.

When space allocation for the file is required, the associated ALLOCATE statement must specify the corresponding internal-file-name of the first ASSIGN statement in the step description. The space allocation becomes effective only when a file is opened using this internal-file-name. Finally, a DEFINE statement, if specified, applies only to a specific internal-file-name and not to the related permanent-file-name.

File Concatenation

Several standard tape, disk, and diskette files may be read in sequence by means of file concatenation. The files are treated by the program as if they are one logical sequential file. Files may be concatenated by specifying the respective ASSIGN statements in the required sequence, with the omission of the internal-file-names on all but the first ASSIGN statement.

Example:

```
ASSIGN ifn, MY.FILEA, DEVCLASS=MT/T9, MEDIA=A1;

ASSIGN    , MY.FILEB, DEVCLASS=MT/T9, MEDIA=A2;

ASSIGN    , MY.FILEC, CATALOG=2;
```

In the above example, the three files are considered as a single sequential file, starting at MY.FILEA and finishing at MY.FILEC (assuming that MY.FILEC is a cataloged tape file).

Restrictions

All the files must be sequential. There may also be restrictions on the difference in attributes (for example, record size, record format) between the files concerned. These restrictions are dependent on the programming language (for example, COBOL, FORTRAN) being used.

If a UFAS file with SHARE=MONITOR (specified in the catalog or in the ASSIGN statement) is included in the concatenation process then either:

```
ACCESS=SPREAD (ASSIGN statement)
```
or
```
READLOCK=STAT (DEFINE statement)
```

must also be specified, in cases where the step is not repeatable.

Uncataloged tape files and diskette files may be concatenated by using the NBEFN parameter. This parameter gives the number of files to be concatenated starting from the "first" file which is specified by its external file name or by FSN. On multi-file tapes the sequence of concatenation is the physical order in which the files are on the volume. On diskettes the sequence of concatenation is the order in which the file labels appear in the Volume Table of Contents, VTOC.

If a multi-extent diskette file is included in the concatenation sequence, then the file processed after it will be the one whose label follows the label of the last extent of the multi-extent file. As a diskette file may have only one extent per volume, multi-extent implies multi-volume as well. Because of the concatenation rule above, files whose labels appear in VTOCs between the first extent label and the last extent label of a multi-extent diskette file, will be omitted from the concatenation sequence.

Example 1:

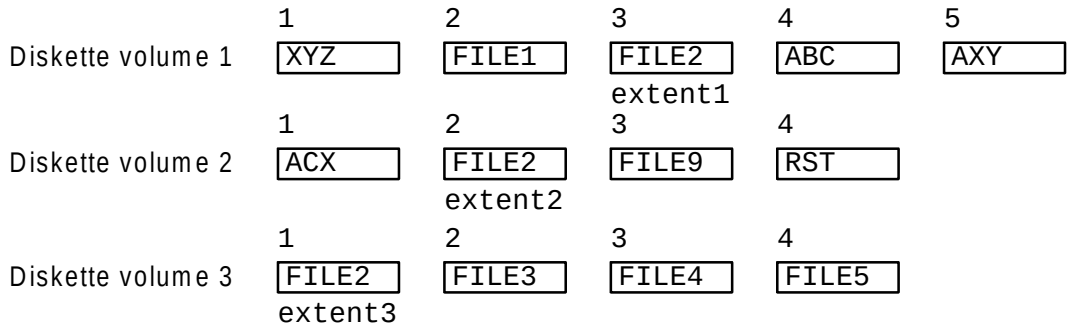
```
ASSIGN INT1, FILE3, NBEFN=5, FSN=3,........ ;
```

Concatenates 5 files on a multi-file tape, starting with the 3rd file on the tape.

Example 2:

```
ASSIGN INT1, FILE1, NBEFN=4,.........;
```

Concatenates 4 diskette files. The first file has the external file name FILE1. The second file is the one whose file label follows that of FILE 1 in the VTOC. If the second file has 3 extents then the concatenation proceeds as follows:



After the 3 extents of FILE2 have been processed, FILE3 and FILE4 are then concatenated. The files ABC, AXY, ACX, FILE9 and RST are not concatenated.

Notes on Suffixes for Internal-file-names

When a COBOL program accesses certain types of files, as described below, the internal-file-name in the appropriate ASSIGN statement must contain a numbered suffix, consisting of a hyphen and one or two digits. The composite internal-file-name must not exceed the normal maximum length of internal-file-names (eight characters). Refer to the *COBOL User's Guide* for more details on the handling of multiple file tapes and multiple logical unit files.

The rules for ASSIGN are:

1. When a COBOL program accesses a file within a multiple file tape, the internal-file-name in the ASSIGN statement must consist of the COBOL internal-file-name followed by a suffix containing the sequence number of the file within the tape.

Examples:

```
ASSIGN SUBFIL-1, CA.MTAPE, DEVCLASS=MT/T9, MEDIA=V66, FSN=1;
ASSIGN SUBFIL-2, CA.MTAPE, DEVCLASS=MT/T9, MEDIA=V66, FSN=2;
ASSIGN SUBFIL-6, CA.MTAPE, DEVCLASS=MT/T9, MEDIA=V66, FSN=6;
```

Note that SUBFIL-14 would not be a valid internal-file-name since it contains more than eight characters.

2. When a COBOL program accesses a file as a logical unit of a multiple logical unit file, the internal-file-name in the ASSIGN statement must consist of the COBOL internal-file-name followed by a suffix containing the sequence number of the logical unit within the multiple logical unit file.

Example:

```
ASSIGN MLNFIL-1, MY.FILEA;

ASSIGN MLNFIL-2, MY.FILEB;
```

Comments and Restrictions

If the COBOL program contains a statement to ACCEPT from SYSIN (see the *COBOL Language Reference Manual*), the ASSIGN statement must be specified as follows:

```
ASSIGN H_RD, *input-enclosure-name;
```

The sum of the ASSIGN, DEFINE, and ALLOCATE statements witihin a step enclosure must not exceed 1200. This maximum value includes all multiple assignments and concatenated files. In an interactive environment, it also includes consigned files. (See the GCL Commands MWINLIB, MWLIB).

4.3.5 Parameters

4.3.5.1 Mandatory Parameters

internal-file-name

The name declared in the source program. It must not exceed eight alphanumeric, hyphen and underscore characters.

This is the first positional parameter of the ASSIGN statement. It may have a null value only in the case of file concatenation (see above).

Examples:

```
INFILE

IFN4

MY-OUT
```

external-file-name

This positional parameter is the name of a permanent file known to the system. The rules for naming files are given in Section 3. This parameter may be omitted if DEVCLASS is the printer (that is, if a unit record device is assigned directly. For further information, see the *Unit Record Devices User's Guide*).

input-enclosure-name

This positional parameter identifies the named input enclosure data to be assigned to the step. This data is held in a SYSIN file and is deleted at job termination. The name must not exceed 16 alphanumeric, hyphen and underscore characters. In the ASSIGN statement, the input-enclosure-name must always be preceded by the * character; when it is specified, the only other valid parameter is the internal-file-name.

temporary-file-name

This positional parameter identifies a temporary file. The name must not exceed 16 alphanumeric and hyphen characters and must start with a letter or a digit. The parameter TEMPRY or FILESTAT=TEMPRY must always be specified for a temporary file. For tape files in ANSI format, the maximum length for a temporary-file-name is nine characters.

DUMMY

This positional parameter specifies that no volume supporting the file is required. All operations on the file are ignored except read operations which produce an end-of-file condition. For certain programming languages (for example, COBOL) DUMMY is restricted to input sequential files; refer to the *appropriate User's Guide*.

*****

This parameter may only be used with non-cataloged tapes and diskettes. It enables a file to be assigned without reference to its external-file-name.

For tapes, it refers to the file whose rank on the tape is given by the File Serial Number (FSN), or the first file on the tape if FSN has not been specified. For diskettes, it means the file whose label appears first in the VTOC.

Restriction

A file specified by EFN=* cannot be passed, and it cannot be repeatable from a checkpoint because this leads to an abort with the return code CONFLICT at restart time.

4.3.5.2 Optional Parameters

FILESTAT

FILESTAT can be used as part of the identification of a file (see Section 3). This specifies the file status, according to the following values:

CAT	Permanent cataloged file. If no residency parameter (RESIDENT, VOLSET, DEVCLASS or DVIDLIST and MEDIA) is specified, then this is the default value for FILESTAT.
UNCAT	Permanent uncataloged file. If FILESTAT is omitted if RESIDENT or DEVCLASS or DVIDLIST is specified, then FILESTAT=UNCAT applies by default.
TEMPRY	Temporary file; in this case, FILESTAT may be omitted (see below).

NOTE: Since default values of FILESTAT exist for cataloged and uncataloged files and since the self-identifying-value TEMPRY can be used for temporary files (see below), the FILESTAT parameter can generally be omitted from the ASSIGN statement. However, FILESTAT must appear in cases where its absence causes ambiguity; for example, in the case of an ASSIGN statement for a file that has been passed from a previous step (and therefore does not require residency parameters - see PASS below).

TEMPRY

This is equivalent to writing FILESTAT=TEMPRY. One of these two parameter forms must be specified when assigning a temporary file. If no residency parameter is specified (and the file is not one that was retained from a previous step using the PASS parameter), the file is located on the job submitter's default volset, provided that the volset facility is active. If it is not, the file is located on the resident volume.

CATALOG

For an existing cataloged file, (i.e. if CATNOW is not specified), this parameter is used to identify a particular catalog in the current catalog search path, for locating the specified file. (See the description of the ATTACH statement in the *Catalog Management User's Guide*.) The value gives the position of the required catalog specification in the corresponding ATTACH statement. If an entry for the file is not found in the specified catalog, the step is aborted. If the CATALOG parameter is omitted, the catalog search rules are applied (all the explicitly attached catalogs, then the site catalog, the system catalog, and the "auto-attachable" catalogs) until the file entry is found.

When CATNOW is simultaneously specified, CATALOG identifies the catalog where the file is to be cataloged.

CATNOW

Dynamically creates an entry for the assigned file in a catalog. The catalog may be directly specified by the optional parameter CATALOG=digit1. Otherwise, the catalog search rules are applied to reference the appropriate catalog.

CATNOW must be specified if the file is to be cataloged. The residency parameter must be specified. This may be done by the RESIDENT parameter, indicating that the file is on a resident disk, or by a volset identification, or by a device class descriptor. A device identification list is not accepted.

If the CATNOW parameter is used, an ALLOCATE statement must be associated with the ASSIGN statement, except for tape files. For further details of this parameter see the *Catalog Management User's Guide*.

EXPDATE

Specifies the expiration date for a file. For a magnetic tape file, EXPDATE provides protection against an accidental attempt to overwrite the file name; for a disk file allocated (with ALLOCATE) in the current step, it prevents any accidental overwrite of the file name and any unintentional deallocation of the file in the future.

NOTES:

1. The date a disk file expires is specified when the space is allocated using the PREALLOC utility (see Section 5) or in ASSIGN when the file is first used in conjunction with an associated ALLOCATE statement. This date is stored in the file label.

2. A disk file can be deallocated (deleted) using the DEALLOC utility (see Section 5). If the expiration date has not been reached, the file may only be deallocated if the BYPASS parameter is specified in the DEALLOC statement.

3. Even if a retention period is specified, a disk file can be opened in output mode and its previous contents overwritten (that is, the file name is protected, but the file contents are not).

4. For uncataloged disk files, the EXPDATE parameter in ASSIGN is only meaningful if the file space is allocated in the current step (with ALLOCATE).

5. For cataloged files (disk and tape), the EXPDATE parameter is only meaningful when a new generation is created; in this case the specified value overwrites the expiration value that is stored in the catalog.

6. For uncataloged tape files, the retention period of a file may be specified each time the file is written, that is, opened in output mode. If no date is provided in this circumstance then the current date is taken as the expiration date.

7. Any EXPDATE value specified for a temporary file is ignored.

8. The VOLPREP utility can delete every existing file on the disk being processed. For this to happen, BYPASS must be specified and, if a file exists for which the expiration date has not been reached, the operator is asked, by a console message, whether all files are to be deleted, regardless of the status of their expiration dates.

9. When the expiration date of a tape file has not been reached, the system protects the file from being overwritten by another file of a different file name. If the expiration date of the tape file has not been reached and a new file of a different name is to be written then the volume must first be returned to the scratch state using the VOLPREP utility (see Section 5). The presence of a retention period does not prevent the file from being opened in output mode and reloaded, but the file name used must be the same as that currently in the file label.

The expiration date in the EXPDATE parameter is specified in one of the following ways:

- Retention period in number of days (for example, 12).

- Date of expiration in the form: year (2 digits)/day within the year (3 digits). For example, 91/046 means the 46th day of the year 1991, or February 15th, 1991.

- Date of expiration in the form: year (2 digits)/month(2 digits)/day(2 digits). For example, 91/02/15 means February 15th, 1991.

VOLWR (for tape and cartridge volumes only)

The tape or cartridge must be mounted with the Write-protection mechanism OFF (with write ring for a tape, or in write-permit position for a cartridge).

NVOLWR (for tape volumes only)

The tape or cartridge must be mounted with the Write-protection mechanism ON (without write ring for a tape, or in write-protect position for a cartridge).

OPTIONAL

The file is considered to be a DUMMY file if either the operator indicates to the system that the file volumes are not available or if the volumes are available but do not, in fact, contain the file.

DEFER

The step can start even if the specified volume is not mounted. In this case, the volume is mounted when the file associated with the internal-file-name is opened. However, a device resource is allocated in order to guarantee that the volume can be mounted at open time, otherwise the job would be aborted if no device were free at that time. Between assign and open, the visible device display is:

```
MSxx ALLOCATED TO RON
```

with no volume name.

RESIDENT

This specifies that the file is stored on resident disk volumes.

DEVCLASS or device-class-description

Specifies the device class (see Section 3) for the device associated with an existing uncataloged file, or a file to be automatically allocated (temporary or to be cataloged if CATNOW is specified). Within this class all devices are equivalent and the specific device used is determined by GCOS.

DVIDLIST or device-identification-list

For an uncataloged file, this specifies a list of one or more devices (see Section 3) by the device names on which the volumes are to be mounted. It is recommended that the DEVCLASS parameter normally be used instead of DVIDLIST.

NOTES:

1. Device names cannot be specified for resident disk volumes.
2. Files that are assigned with the use of the DVIDLIST parameter are not sharable.

MEDIA

This is specified with either DEVCLASS or DVIDLIST. You specify the names of non-resident volumes supporting the file (uncataloged, temporary, or to be allocated and cataloged when CATNOW is specified). The volume-name is composed of up to six alphanumeric characters. Unless the volume(s) containing the file is premounted or resident, at execution time a mount message for the required volume(s) is generated on the operator's console. The maximum number of volume names is 10.

For the printer, the MEDIA parameter gives the print belt and paper identification as a six-character value, as follows:

- Characters 1-2 = print belt identifier.
- Characters 3-6 = form number.

WORK — This parameter value can only be specified with the MEDIA parameter for output tapes. It requests the use of a number (specified by the value of MOUNT) of native labelled work tapes.

* — The volume names are not known when a job is submitted, but will be supplied by the operator at run time. This value may be given for input or output files. It is applicable to catalogued or uncataloged tape files, and diskette files. Its use prohibits device reconfiguration and pre-mounting of volumes. When using MEDIA=*, the volumes and their sequence of mounting is under the complete control of the operator, so care must be taken to ensure that he has a list of volumes in the correct sequence and that he can readily identify each volume.

NOTES:

1. A file in a work tape may be used in several steps of a job if END=PASS appears in each step that uses it.
2. A work tape can be protected from use by other users if END=PASS appears in the ASSIGN in each step which accesses a file in the tape.
3. The VOLPREP or TAPEPREP utilities can be used to prepare a work tape.
4. If TEMPRY (or FILESTAT=TEMPRY) is not specified in ASSIGN when a program writes to a file in a work tape, the tape loses its work status.
5. If the list of volumes specified (for example, in MEDIA) for an ordinary tape file in output mode is not sufficient (that is, if more tapes are required), the system automatically assigns work tapes to the file, until the limit of 10 volume names is reached.

VOLSET

This specifies the volset name where the file is to reside. This parameter can only be used when the volset facility is active on the system and when the file is temporary or is to be cataloged (CATNOW parameter is also specified). It can be used for non-existent files, and for files that are already assigned (for example, a temporary file retained from a previous step by the PASS parameter). Other residency parameters (RESIDENT, DEVCLASS or DVIDLIST and MEDIA) cannot be used if VOLSET is specified. The volset name consists of up to six alphanumeric characters, chosen by the System Administrator.

The name DFLT is a symbolic name reserved for naming the default volset:

- with the basic function, the site volset is the default volset for all the projects. It is the only volset available.
- with the full function, a project's default volset is defined by the System Administrator. If a project does not have a default volset, the site volset is its default.

For further details, refer to the *Administering the Storage Manager Manual*.

SUBFILE

Applicable only to a member of a library (identified by external-file-name above). This parameter permits the named member to be accessed sequentially. The member-name can contain up to 31 alphanumeric, hyphen, and underscore characters and must begin with a letter or a digit. For example, a subfile can be a source library member which contains a communications network description. A subfile can also be specified for output being sent to SYSOUT.

SHARE

Specifies the rules for the common sharing of the file, as required within the step (due to multiple assignments of the same file) or as requested with respect to other concurrent steps. If the SHARE parameter is omitted, the default values apply. (See the note below).

NORMAL	One writer (output processor) or several readers (input processors). Two or more concurrent assignments with different SHARE values are not permitted except between SHARE=NORMAL and SHARE=ONEWRITE. For example, if a job is running with the values SHARE=FREE, ACCESS=WRITE (see ACCESS below), a request for SHARE=ONEWRITE, ACCESS=READ will not be granted until the first job has released the file. Note that after an assignment with SHARE=NORMAL and ACCESS=READ, a request for SHARE=ONEWRITE and ACCESS=WRITE will not be granted.
ONEWRITE	One writer and several readers.

FREE	No restrictions.
MONITOR	This applies to UFAS and IDS/II disk files only. It is used to indicate that file sharing is under GAC (General Access Control). A file may be assigned simultaneously to several readers and writers. Sharing is controlled at access time at the Control Interval (CI) level rather than at assign time at the file level.
DIR	Applies only to libraries. Each member can be concurrently accessed by several readers or one writer.

NOTE: The default value of SHARE for uncataloged files is NORMAL, except when the parameter SUBFILE is specified; in this case default value for SHARE, assuming it is a library, is DIR. The default value for cataloged files is the value given in the catalog entry for the file; however, if a value that differs from the catalog entry appears for SHARE, the catalog entry overrides the specified value and the step has exclusive access to the file.

ACCESS

Specifies the allowable processing modes for the step.

WRITE	Allows input, output and append modes.
READ	Allows input mode only.
SPWRITE	Same as WRITE. Additionally, the current step has exclusive file control, regardless of the SHARE value.
SPREAD	Same as READ. Additionally, the current step has exclusive file control, regardless of the SHARE value.
RECOVERY	Applies only to cataloged files (see the *Catalog Management Guide*); the step has exclusive access for file recovery purposes.
ALLREAD	Same as READ. Additionally, access by others is restricted to READ only, regardless of the SHARE value.

NOTE: When an ALLOCATE statement is associated with the file, that file is only accessible from within the step (that is, ACCESS=READ is treated as ACCESS=SPREAD and ACCESS=WRITE is treated as ACCESS=SPWRITE).

END

Indicates to the system the state in which the file should be left after normal step termination, as follows:

PASS

If the file is temporary it is retained for use by a subsequent step in the same job, until either job termination or a subsequent step assigns the file without a PASS option, in which case the file is deleted at termination of that step. If the file is permanent, then volume and device identification associated with the file is retained. For a temporary or permanent file that is assigned to a later step in this job, the DEVCLASS (or DVIDLIST), MEDIA, RESIDENT, VOLSET and LABEL parameters do not need to be specified in the related ASSIGN statement.

An ASSIGN in a subsequent step must specify internal-file-name, external-file-name and FILESTAT (or TEMPRY) in order to access a passed file. If DEVCLASS (or DVIDLIST), MEDIA, RESIDENT, VOLSET or LABEL are also stated, they must be consistent with the values specified in the passing step.

The stated values, if any, for MOUNT, END, ABEND, SHARE and ACCESS apply only to the step in which they are specified. They are not passed automatically.

NOTE: A file specified by EFN=* cannot be passed.

DEASSIGN

A temporary file is deleted at the end of the step or when a CLOSE WITH LOCK is executed in the (COBOL) program. For a permanent uncataloged file that is assigned to a later step in this job, the subsequent ASSIGN statement must include full volume and device identification (RESIDENT, or DEVCLASS (or DVIDLIST) and MEDIA).

LEAVE

Applies to tape or cartridge files with the following results:

At CLOSE time, the media is left in its current position.

At volume switching time, the media is rewound, except if:

- ENDVOL parameter is specified.
- the ASSIGN was performed with MD=* (any media), in which case the media is unloaded;
- the ASSIGN was performed with FSN=NEXT and the access was INPUT (case of ACCESS=READ, or INSET with utilities), in which case the media is left in its current position (at EOT);

- the file is a cartridge file on a cartridge driver equipped with a Sequential Cartridge Loader (SCL) in AUTOMATIC mode, in which case the media is unloaded;

- the program has a request for LEAVE in the VOLUME SWITCHING primitive.

At the end of the step, the last processed media is left in its current position.

In any case, the volume remains reserved until the end of the step, even if the tape is deassigned or the device is deallocated.

WARNING

LEAVE is efficient if the number of tape drives is greater than or equal to the number of volumes in the media list.

UNLOAD

Applies to tape or cartridge volumes only, with the following results:

At volume switching time, the media is unloaded, except if:

a) ENDVOL parameter is specified.

b) files are processed by utilities as an INSET, in which case the media is left in its current position (at EOT);

c) files are processed by utilities as an OUTSET, in which case the media is rewound;

d) files are cartridge files on a cartridge driver equipped with a Sequential Cartridge Loader (SCL), in MANUAL mode, in which case either b) or c) above applies, depending on the processing type.

NOTES:

1. In the case of cartridge files on a cartridge driver equipped with a SCL, in AUTOMATIC mode, if you specify the END=UNLOAD option, the last used cartridge and the following unused ones (if there are any) declared in the media list of the corresponding ASSIGN statement, are unloaded during deassignment. In MANUAL mode, only the last used cartridge is unloaded.

2. END is not applicable if *input-enclosure-name is specified.

3. For multivolume temporary files for which MOUNT is specified, total deletion of the file can only occur if a sufficient number of devices are available to mount all of the volumes of the file simultaneously. However, if MOUNT is not specified (that is, all volumes are to be mounted, by default) when DEASSIGN is stated, total deletion of the file is guaranteed. If MOUNT is not specified when PASS is stated, total deletion can only occur if a sufficient number of devices is available to mount all of the volumes simultaneously. If PASS is stated, deletion will occur at the next DEASSIGN relating to an ASSIGN with END= DEASSIGN, or, at the latest, at end of job.

4. Foreign tapes (processed by the Foreign Tape Processor FTP) are always considered as monofile volumes by GCOS 7.

5. For a tape file for which LEAVE or UNLOAD has not been declared, or for which PASS has been specified, the implicit action is REWIND, with the following observations:

   - at close time, the tape is rewound when the ASSIGN has been performed with the POOL option, or when file concatenation has been declared.

   - at VOLUME SWITCHING time, refer to the description of VOLUME SWITCHING for END=LEAVE.

   - at the end of the step, the tape volume is rewound, except if the ASSIGN was performed with MD=*.

WARNING

When a cartridge library is used, dismounting can be forced. Refer to the *Cartridge Tape Library Unix Server User's Guide*, paragraph: Dismounting Library Volume.

ABEND

Indicates to the system the state in which the file should be left if the step is abnormally terminated. The values PASS, DEASSIGN, LEAVE and UNLOAD are described above.

ENDVOL

Applies to tape and cartridge device with multi-volume file; it concerns all volumes on which the file spread on except the last one which is concerned by END parameter. It indicates to the system the state which the volume should be left as follows:

UNLOAD	Intermediate volumes are unloaded at volume switching time.
REWIND	Intermediate volumes are rewound at volume switching time.
LEAVE	Intermediate volumes are left in their current position at volume switching time.

MOUNT

The MOUNT parameter applies to UFAS sequentially organized files.

MOUNT defines the number of devices allocated for the file to this step and the number of volumes in a particular device class that need to be mounted at step initiation (before step execution begins) when POOL is not specified. See POOL for a description of processing when POOL is specified. The volumes are those specified by the MEDIA parameter or those selected by the volset facility when the VOLSET parameter is used. If all the volumes are not mounted at step initiation, the remaining volumes are mounted during step execution according to program requirements. If MOUNT is not present, then in the case of disks, all volumes are mounted by default. The default value of MOUNT for all device classes except disk is 1.

NOTE: If MEDIA=WORK is specified, MOUNT specifies the number of drives to be allocated.

POOL

Specifies that volumes supporting the file are to be mounted (see the MOUNT parameter) on pool devices with the same attributes.

These devices may be allocated to the pool in one or more POOL statements. The related POOL statement(s), if present, must precede the ASSIGN statement in the step enclosure.

When the POOL parameter is used in an ASSIGN statement, the POOL Statement is not mandatory. If the POOL Statement is omitted, the pool of devices is dynamically reserved according to all the pool parameters of the step.

When POOL is specified, a pool device is reserved for use prior to the step execution and is assigned to the file when the file is opened. The device is only released and returned to the device pool when the file is closed. For example, in COBOL this is done with the CLOSE WITH LOCK option or with the CLOSE option. A detailed description of the use of device pools is given in the *JCL User's Guide*.

FIRST

May only be specified if the POOL option is chosen. It specifies that one or more volumes (according to the MOUNT parameter value) supporting the file, named in this ASSIGN statement in this step, are to be mounted on a pool device or devices before the file associated with the internal-file-name is opened in the step, and before step execution begins. As a result, the sum of all devices specified with "POOL, FIRST" must not exceed the minimum number of devices defined in a corresponding POOL statement.

NEXT

May only be specified with the POOL option. The volumes supporting the file, named in this ASSIGN statement in this step, are to be mounted on pool devices during step execution, according to program requirements.

LABEL

Specifies the type of volume label according to the values:

NATIVE	GCOS standard label organization
NONE	No label
NSTD	Unknown label format. Allows you to work with files in GCOS non-standard label and see them like GCOS standard files.

By default, the system expects tape and disk volumes to have standard label organizations (LABEL=NATIVE). If a volume is presented to the system which does not have standard labels then you must specify the label organization explicitly. If the value specified by LABEL, NATIVE by default, does not match the labels encountered by the AVR (Automatic Volume Recognition) feature of GCOS, then the execution of the step cannot start.

The value NONE declares that there are no recognizable labels present on the volume.

When LABEL=NONE is specified for an input tape volume, the GCOS access methods interpret the first tape mark encountered as the end-of-file.

FIRSTVOL

Applies only to multivolume tape files. This parameter defines the rank of the first volume to be accessed at open time, that is, the value gives the position in the media list of the first volume to be mounted. For cataloged files, the media list is stored in the catalog; for uncataloged files, the media list is specified by the MEDIA parameter of ASSIGN.

This parameter cannot be specified when the file is not cataloged and the processing mode is neither APPEND nor INPUT.

In addition, if FIRSTVOL is specified for a labelled file, File Management checks that the volume sequence number (the order of the volume relative to the start of the file) of the volume chosen is identical to the value of FIRSTVOL. If FIRSTVOL is not specified, the step proceeds without any check being made.

The EOF value for FIRSTVOL specifies that the first volume to be processed is that which contains the end-of-file marker. This is applicable to cataloged files only.

Example:

If the media list is (V1, V2, V3, V4) and FIRSTVOL=3 is specified, then the system will request V3 to be mounted, and, at open time, the volume sequence number of V3 will be examined (to check that it is the third volume of the file).

LASTVOL

For a multivolume file, this parameter defines the rank of the last volume to be accessed, that is, the value specifies the position in the media list of the volume which is considered to contain the end of the file. A data limit condition (DATALIM return code) will be signalled when the processing of this volume has been completed. The special value LASTVOL=EOF specifies that the last volume to be processed must contain the end-of-file marker. In this case, for an uncataloged file, an error condition will arise when the last record of the volume is accessed if the media list is incomplete.

LASTVOL=EOF is the default value for sequential files, i.e, the only type which do not require all volumes to be mounted at step initiation (tape and diskette).

This parameter cannot be specified for a disk file if an allocation request has been made.

Example:

```
LFILE,MY.BFIL,LASTVOL=EOF,DEVCLASS=MT/T9/D1600,
                                    MEDIA=(V1,V2);
```

In the above example, if V2 does not contain the end of file MY.BFIL, the step will be abnormally terminated. As LASTVOL=EOF is the default value for tape, it could have been omitted above.

However, if LASTVOL=2 were specified the step would terminate normally, even through V2 does not contain the EOF.

FSN

Applies only to multifile tape volumes; it must be specified for all files on a tape (or tapes) and defines the rank (relative position) of the file in the "family" of files. The value must be an integer between 1 and 253 inclusive. If the value ANY is given, File Management will search for the file on the tape at file open time. If the value NEXT is used for a file in output mode, the new file will follow the existing files on the tape even if a file of the same name already exists on the tape; (if ANY is specified, the first file with the same file name will be overwritten).

NOTES:

1. In output processing mode, a file identified by FSN may be overwritten by another file (provided the retention period has expired - see EXPDATE below).

2. The NEXT value is available only in output mode when using multivolume files or tapes. Use ANY or a number to process such files in input mode.

WARNING

When LABEL=NONE, ANY or NEXT is assumed as FSN=1.

NBEFN

Refers to tape and diskette files only, where file concatenation is desired. Its value gives the number of files to be concatenated starting from the file specified by FSN or external-file-name. For tape files the concatenation takes place using the files in the sequence on the volume (that is, multi-file tape volume).

For diskette files the concatenation sequence is that of the sequence of file labels on the VTOC. A further complication arises in the case of multi-extent diskette files. As a diskette file may have only one extent per volume, multi-extent implies multi-volume. In such cases the concatenation exhausts the extents of the multi-extent file and continues with the file whose label follows that of the last extent. Consequently files on the intervening volumes may be missed out of the concatenation process.

If the value ALL is specified, then all files on the volume are concatenated. The concatenation ends at the end of volume (tapes) or the end of VTOC (diskettes). If several volumes are listed then the concatenation continues through to the last volume on the list. If MEDIA=* is used then the processing stops when the operator cancels the mount request for the next volume.

DENSITY

Applies exclusively to tape or cartridge files opened in output mode. The use of DENSITY overrides the tape or cartridge density specified on the label and therefore allows the tape or cartridge to be written with a density different from that of the existing file on the tape or cartridge. For a multi-file tape or cartridge, the parameter applies only when the first file is written.

NOTE: In the absence of the DENSITY parameter, irrespective of the processing mode used, a warning message is issued when there is a conflict between the density given in the DEVCLASS parameter and that in the file label; however, the label density is chosen.

Examples of the ASSIGN Statement

Example 1:

```
ASSIGN HJFILE, MY.FILE1;
```

The above example assigns the internal-file-name HJFILE to the cataloged file MY.FILE1.

Example 2:

```
ASSIGN DMFILE,Z.ABC, EXPDATE=30, DEVCLASS=MS/D500, MEDIA=B012;
```

An uncataloged disk file with external-file-name Z.ABC is assigned. Provided the space for the file is allocated in the current step (with ALLOCATE), an expiration date of 30 days from the current date will take effect.

Example 3:

```
STEP S1, ...;

ASSIGN  SCRFIL, TMP1, TEMPRY, DEVCLASS=MT/T9/D1600,
            MEDIA=WORK, END=PASS;

..........

ENDSTEP;

STEP S2, ....;

ASSIGN  INDATA, TMP1, TEMPRY;

..........

ENDSTEP;
```

A temporary work file on 9-track tape is passed from one step to another. The file is deleted at the termination of step S2. Note that it is unnecessary to repeat the residency parameters (DEVCLASS and MEDIA) in the second step enclosure.

4.4 COMMENT

4.4.1 Function

Enables a comment to be inserted in a job description.

4.4.2 Enclosure

Job, step.

4.4.3 Format

```
COMMENT 'comment-string';
```

4.4.4 Description of Statement

The COMMENT statement allows a comment to be inserted in a job description. Comments may appear anywhere in a job enclosure other than within an input enclosure. The COMMENT statement is a dummy statement and is not executable.

4.4.5 Parameters

comment-string

The comment-string is a protected string; that is, any character can be used, except that a single quote within the string must be entered twice in succession. The single quotes enclosing the string appear as part of the comment in the Job Occurrence Report.

Example 1:

```
COMMENT 'NEXT JOB STEP PRINTS MONTHLY SALES FIGURES';
```

Example 2:

```
COMMENT 'OWNER''S NAME & ADDRESS REQUIRED';
```

4.5 CONSOLE

4.5.1 Function

Defines the destination for user console messages.

4.5.2 Enclosure

Job.

4.5.3 Format

```
CONSOLE    [user-name];
```

4.5.4 Description of Statement

The CONSOLE statement applies to all following SEND statements and to all operator messages issued by user load-modules (during step execution) and by system procedures (for example, execution of Extended JCL Statements).

This statement also determines the source and destination console of the COBOL verbs ACCEPT and DISPLAY. See the *COBOL User's Guide* for details.

The messages are sent to the terminal which is logged on to the system with the given user identification. The default terminal is the main console.

4.5.5 Optional Parameter

user-name

The user-name identifies the user of the required terminal. The job (for a SEND) or the step (otherwise) will abort if the user identification is not associated with a logged on terminal at the time the message is to be sent. If the user-name value is omitted, the console associated with the current job is the Main console.

4.6 $DATA

4.6.1 Function

Stores input data in a source library member.

4.6.2 Enclosure

Outside a Job enclosure.

4.6.3 Format

```
$DATA    {member-name   }       [{(output-library-description)}]
         {[alphanum 17]*}       [{SITE.IN                     }]

          [TYPE      =    {DATA   }]
          [               {DATASSF}]
          [               {COBOL  }]
          [               {COBOLX }]
          [               {RPG    }]
          [               {FORTRAN}]
          [               {GPL    }]
          [               {JCL    }]

          [FORMAT = (digits3, digits3, digits3, digits3)]

          [END       =    {   EOF    }]
          [               { ENDDATA  }]
          [               {'string 8'}]

          [ENDCHAR  = 'string1']

          [CONTCHAR = 'string1']

          [RECSIZE  = {  110  }]
          [           {digits3}]

          [REPLACE]     [PRINT]     [CKSTAT]
[USER=user-name]        [PROJECT=project-name]

          [BILLING=billing-name] ;
```

4.6.4 Description of Statement

This statement loads input data into a source library member. It is processed directly by the Stream Reader and it may be seen as a self contained job with one step, which loads the data into the file. As it is functionally equivalent to a job, it cannot be included in a job enclosure.

The $ sign is mandatory and must appear in position 1 of the record.

All records following the $DATA statement are treated as data records to be loaded into the output file until a terminator record is encountered. This terminator record may be any of the following:

- a $ENDDATA statement;
- a $EOS record;
- an EOF (end of file marker);
- a record containing the string specified in END='string 8'.

The output library must be a permanent file that can be accessed immediately at stream reading time (that is, it does not involve waiting for volume mounting or other resources). The USER and PROJECT have to be specified to match the access rules for the output library file and to conform to job submission rules. A Job Occurrence Report is produced if the PRINT parameter is specified or an error is detected during input stream processing.

If the member name already exists on the output-library the input data is not loaded unless the parameter REPLACE is also specified. The storing of input data in a file can also be done using a standard job enclosure that includes either a CREATE step (see the *Data Management Utilities Manual*) or a LIBMAINT step (see the *Library Maintenance Reference Manual*). However, by this method, the data is transferred during the execution of the JCL statements whereas with the $DATA statement, the data is stored at Stream Reader time. The Stream Reader statement $INPUT loads input data as a member of the library SYS.IN whereas $DATA may be used to load input data directly into a member of a permanent library file.

If a $JOB, $INPUT, or $SWINPUT statement is present in a data enclosure (that is, between the $DATA and $ENDDATA statements) then the result is as follows:

$INPUT	treated as a data record.
$SWINPUT	executed, if the CKSTAT parameter is present in the $DATA statement, otherwise it is treated as a data record.
$JOB	treated as a data record, if the stream is being read from a sequential file, a library member, or from a remote station (RBF); otherwise it leads to an abort.

4.6.5 Parameters

4.6.5.1 Mandatory Parameters

member-name

The first positional parameter, which is mandatory, gives the member name of the library in which the data is to be stored. This may be specified either as a name of up to 31 characters, which gives the member name in full, or as a name of up to 17 characters followed by an *. In this latter case, the member name is obtained by concatenating the name preceding the *, with a system-generated 14 character date-time string. This date-time string is in YYMMDD-HHMMSS format preceded by the letter D and represents the year-month-day-hour-minute-second at which the Stream Reader processes the $DATA statement. This date-time stamping may be used to give unique member names to members that would otherwise be identical.

output-library-description

The second positional parameter specifies the name of the library of which the first positional parameter is a member. This parameter is optional and has a default value of SITE.IN, a resident source library.

The library specified must be a permanent one and all the normal output-library-description parameters except SIZE may be specified.

4.6.5.2 Optional Parameters

USER

Specifies the user identification. It is needed if the output library is cataloged and protected. See the $JOB statement for a full description.

PROJECT

Specifies the project name. See the $JOB statement for a full description.

BILLING

Specifies the name used for accounting control within the PROJECT. The name may be up to 12 characters, the first of which must be alphabetic or numeric, the others may be alphanumeric, hyphen or underscore. The default value is the value specified for this PROJECT in the site catalog or, failing that, the project-name is used as the billing-name in the job report banner.

WARNING

The BILLING parameter is mandatory in the $JOB statement if there is no default value given in the site catalog.

TYPE

Specifies the format of the member to be loaded. DATA means that the records loaded will not contain SSF headers nor will there be control records. DATASSF means that the records loaded will contain SSF headers and that there will be control records. The other values refer to language types and are described under the MOVE command in the *Library Maintenance Reference Manual*.

FORMAT

Defines the position of the line number and the text on a record. The four values specify the starting position of the line number, the ending position of the line number, starting position of the text and ending position of the text, respectively. Zero values for the line number means that there are no line numbers. Line numbers, if given, must be right justified and blanks are treated as zeros.

This parameter applies only if TYPE=DATASSF. If this parameter is omitted the default is taken as the FORMAT corresponding to TYPE. This is specified in the *Library Maintenance Reference Manual*.

END

Specifies the format of the statement which marks the end of the input data to be stored. The value ENDDATA, which is the default, means that input data ends at the record before the first $ENDDATA statement that is encountered. If END='string8' has been specified, then the last input record stored is the one before the first occurrence of a record which contains only this string, starting in the first position. If END=EOF is specified, the last record stored is the last record of the input stream (that is, the one before the $EOF). The record which contains $ENDDATA, END='string8', or END=EOF is not loaded into the output library. Its character representation must be EBCDIC.

ENDCHAR

The character specified by this parameter is considered as an end of record delimiter if it is the last significant character encountered on an input record. Neither the character itself nor the trailing blanks will be loaded in the output record. See $INPUT for examples of the use of this parameter.

CONTCHAR

The character specified by this parameter is considered as a continuation mark if it is the last significant character encountered on an input record. The next record read is concatenated with the record that contained the continuation character. Neither the continuation character nor its trailing blanks are included in the concatenation. See $INPUT for examples of the use of this parameter.

RECSIZE

Specifies the record size. This can be specified only if the input comes from a KDS terminal and in this case its default value is 110. It is ignored for any other type of input.

REPLACE

If the data being loaded is to replace the contents of an existing member of the output library, the parameter REPLACE must be specified. This parameter should not be specified if the member name does not exist on the output library.

PRINT

Requests the printing of the records on the JOR. Up to 500 records may be printed.

CKSTAT

This parameter is necessary if any $SWINPUT statements are to appear and be interpreted in the data-enclosure.

Example:

```
$DATA   SALES*,    TOTAL,    USER=ACCOUNT,    PROJECT=TURNOVER
        TYPE=DATA,   PRINT;

X X X X X X            |
.  .  .  .  .  .  .    |
.  .  .  .  .  .  .    | input data
.  .  .  .  .  .  .    |
X X X X X X            |

$ENDDATA;
```

If the above is run on July 20th, 1991 at 2:55 and 10 seconds p.m. (14.55.10 on 20 JULY 1991) then the input data is loaded as member SALESD910720-145510 of the permanent library TOTAL. This member did not exist before the run. The input data records are printed.

Example:

```
$DATA    ABC, PROGLIB, USER=PROG, PROJECT=DEVELMT
   TYPE=JCL,  FORMAT=(0, 0, 1, 80) REPLACE, PRINT;

X X X X X X          |
.  .  .  .  .  .     |
.  .  .  .  .  .     |
.  .  .  .  .  .     | input data
.  .  .  .  .  .     |
.  .  .  .  .  .     |
X X X X X X          |

$ENDDATA;
```

Before this run the member ABC exists in the permanent library PROGLIB. This example replaces the contents of ABC with the input data supplied. The input data consists of JCL statements in record positions 1 to 80 and there are no line numbers. The input data records are printed.

Example:

```
$DATA ACCT, LIBRARY, USER=CHIEF, CKSTAT;

X X X X X X       |
.  .  .  .  .  .  |
.  .  .  .  .  .  |
.  .  .  .  .  .  | data A
.  .  .  .  .  .  |
.  .  .  .  .  .  |
X X X X X X       |

                          Contents of File          DATA FILE

$SWINPUT DATA-FILE;           X X X X X X               |
                              .  .  .  .  .  .          |
                              .  .  .  .  .  .          |
                              .  .  .  .  .  .          | data B
                              .  .  .  .  .  .          |
                              .  .  .  .  .  .          |
                              X X X X X X               |
X X X X X X       |
.  .  .  .  .  .  |
.  .  .  .  .  .  |
.  .  .  .  .  .  | data C
.  .  .  .  .  .  |           End of File : DATA -FILE
.  .  .  .  .  .  |
X X X X X X       |

$ENDDATA;
```

The data is to be loaded into the new member ACCT of the library file named LIBRARY. The presence of the parameter CKSTAT advises the Stream Reader to look for $SWINPUT statements in the data-enclosure and to give effect to each as encountered.

The data loaded is as follows: data A - data B - data C.

4.7 DEFINE

4.7.1 Function

Requests file journalization, specifies checkpoint positions on files and file processing performance parameters. Overrides and complements file parameters in a user program and/or in certain JCL statements; complements the file description in a file label.

4.7.2 Enclosure

Step.

4.7.3 Format

```
DEFINE     internal-file-name

           [JOURNAL  =  {AFTER }]
           [            {BEFORE}]
           [            {NONE  }]
           [            {BOTH  }]

         [CKPTLIM    =  {digits8 [EOV]}]
         [              {NO           }]
         [              {EOV          }]

         [NBBUF    =  digits5]

         [BPB      =  digits3]

         [{SYSOUT }]
         [{NSYSOUT}]

         [ERROPT    =  {SKIP   }]
         [             {ABORT  }]
         [             {IGNORE }]
         [             {RETCODE}]

         [WRCHECK]

         [OPTIMIZE]

         [{BPIOC}]
         [{FRIOC}]

         [{RAHEAD}]

         [{KEYLOC       =   digits5}]
         [{INKEYLOC     =   digits5}]
```

```
        [{DLREC }]                          [{TRUNCSSF}]
        [{NDLREC}]                          [{NTRNCSSF}]

        [{COMPACT }]
        [{NCOMPACT}]

        [FPARAM]

        [LTRKSIZE = digits5]

[FILEORG     = {SEQ     }]
[               {RELATIVE}]
[               {INDEXED }]
[               {LINKQD  }]
[               {NONE    }]

 [KEYSIZE    =  digits3]

 [CISIZE      =  digits5]

 [CIFSP       =  digits2]

 [BUFPOOL     =  alphanum4]

 [READLOCK    =  {NORMAL}]
 [               {EXCL  }]
 [               {STAT  }]

 [LOCKMARK]

 [FILEFORM    =  {UFAS  }]
 [               {ANSI  }]
 [               {NSTD  }]
 [               {LINKQD}]
 [               {NONE  }]

 [BLKSIZE     =    digits5]

 [RECSIZE     =    digits5]

 [RECFORM     =  {F  }]
 [               {V  }]
 [               {U  }]
 [               {FB }]
 [               {VB }]

 [FUNCMASK    =   hexa 8]

 [BLKOFF      =   digits3]

[{CONVERT }]
[{NCONVERT}]

[DATACODE = {BCD   }]
[           {ASCII }]
[           {EBCDIC}]

[COLLATE  = {BCD   }]
[           {ASCII }]
[           {EBCDIC}]
```

```
[{BSN }]
[{NBSN}]

[NSORTIDX]

[{PRINTER  =  (printer-options) }]
[{TN =  (terminal-options)}]
```

Where `printer-options` have the form:

```
[{SLEW}]
[{NSLEW}]

[MARGIN = digits3]

[PRDEN ={6}]
[       {8}]

[FORMHT = digits3]

[HOF ={digits3}]
[     {   1   }]

[FF1 ={digits3        }]
[     {value of FORMHT}]

[FF2 ={digits3}]
[     {   0   }]

[CHi = (digits3 [digits3] ...)]
```

Where `terminal-options` have the form:

```
[{PROMPT = char12}]
[{NPROMPT         }]

[EOF = char4]

[{MSG }]
[{NMSG}]
```

```
          [{SLEW }]
          [{NSLEW}]
          [MARGIN = digits3]
          [PRDEN  ={6}]
          [        {8}]
          [FORMHT = digits3]

          [HOF    ={digits3}]
          [        {   1   }]

          [FF1    ={digits3        }]
          [        {value of FORMHT}]

          [FF2     ={digits3}]
          [         {   0   }]

          [CH1    = (digits3 [digits3]  ...)]
```

4.7.4 Description of Statement

The DEFINE statement is used to request the activation of the JOURNAL facility, to specify checkpoint positions on files and to specify file processing performance parameters (for example, number of buffers).

The DEFINE statement is also used to override and complement specified file parameters in the corresponding description of the file in the user program and to override and complement certain JCL statements, particularly for input and output handling. There are no default values for the parameters within the DEFINE statement. If a parameter is not specified, its corresponding value, where one exists, in the file description or in the JCL is not overridden.

For each DEFINE statement except for those associated with standard SYSOUT files, there must be a corresponding ASSIGN statement preceding the DEFINE in the same step enclosure.

A DEFINE statement associated with the first file of a sequence of concatenated files is also applied to the others in the sequence. A DEFINE statement may be linked only to the first file of a concatenated set as the others do not have unique internal-file-names through which DEFINE could reference them.

The overriding is applied to the file with the specified internal-file-name at the time that the file is opened. The overriding is effective until the file is deassigned. However, DEFINE does not override the contents of the file label (for a disk file, or for tape file opened in input or append mode), unless the parameter FILEFORM=NSTD applies.

In the case of non-cataloged tape files being processed in output mode, file level information does not override DEFINE statement parameters.

A DEFINE statement associated with a standard SYSIN subfile is ignored.

In general, there is no checking of a specified value against the access method that is used in the program. As a result, some parameter specifications may be useless and will be ignored.

For example:

- the relationship of the values given in a DEFINE statement with those given in the corresponding COBOL program is not examined for consistency;

- NBBUF may specify more or fewer buffers than are specified in the program's file definition. If more buffers are specified, an abnormal termination of the step may result.

4.7.5 Parameters

4.7.5.1 Mandatory Parameters

internal-file-name

Specifies the file to which the overriding is to apply. Internal-file-name must not exceed eight alphanumeric, hyphen and underscore characters.

4.7.5.2 Optional Parameters

JOURNAL

Records images of data blocks on a journal (file or files) which are used in conjunction with the Checkpoint/Restart facility (see Section 1). The journal image is used to return the data blocks to a pre-abnormal- termination state.

If the file is being processed in input mode (that is, ACCESS=READ, SPREAD, or ALLREAD), journals are not used.

AFTER	The data block images are those which existed on the After journal (that is, the user records after they have been modified). This applies to UFAS and IDS/II files.
BEFORE	The data block images are those which existed on the Before Journal prior to any modification of the data since the last checkpoint was taken. UFAS, and IDS/II files are supported.
NONE	No journal file is produced.
BOTH	Both the AFTER and BEFORE journals are produced. This applies to UFAS and IDS/II files.

The type of journal to be used in processing a file can be stored in the catalog. In such cases, if a JOURNAL value is specified in the DEFINE statement, the following rules apply.

Rule 1:	This rule applies if Access Rights have not been implemented, or if they have been implemented and the user has at least the RECOVERY access right (that is, RECOVERY or OWNER) on the file concerned). The rule is the following. The DEFINE value is effective (that is, DEFINE overrides the catalog).

Rule 2: This rule applies if Access Rights have been implemented and the user does not have the RECOVERY access right (that is, his access right is LIST, EXECUTE, READ, or WRITE) on the file concerned. The DEFINE value is effective only if the DEFINE offers the same, or more, protection as the one stored in the catalog. that is:

The DEFINE value is	The Result is
the same as catalog value	OK
BOTH	OK
Any value or catalog value does not exist	OK
All other cases	Not OK in ASSIGN or DEFINE depending on the sequence DEFINE --> ASSIGN or ASSIGN --> DEFINE

CKPTLIM

Defines when an automatic checkpoint must be taken. It is independent of a COBOL program (which has CALLS to the checkpoint mechanism). The possible values are :

NO — No checkpoint is taken.

EOV — For a multivolume disk or tape file in the situation where all volumes where not mounted initially, this value specifies that a checkpoint is taken at the end of each volume.

digits8 — This value gives the frequency of checkpoints (that is, a checkpoint is taken once every digits8 records). Note that EOV may be specified simultaneously in this case.

If monitored files are used by the COBOL step, then an automatic commit is taken instead of a checkpoint. For further information about checkpoints, refer to the *GAC-Extended User's Guide*, Section 3.3.6.

NBBUF

Specifies the number of buffers to be assigned to the file.

BPB

Specifies the number of blocks per buffer (up to 255).

SYSOUT

The specification of this parameter forces the use of the SYSOUT mechanism for the writing of a file, provided the file characteristics (RECSIZE) are acceptable - see the SYSOUT statement. This parameter can be used, for example, where the clause ASSIGN ... TO ... -SYSOUT does not appear in a COBOL program.

NOTE: The use of SYSOUT in DEFINE is equivalent to the specification -SYSOUT in COBOL.

NSYSOUT

The specification of this parameter overrides any request for the SYSOUT mechanism (for example, in the program or in the SYSOUT statement). The SYSOUT mechanism will not be used to write the file. This is required if you wish to prevent the system from using the SYSOUT mechanism for the creation of an uncataloged permanent SYSOUT file on magnetic tape (the use of the SYSOUT mechanism here would force a record size of more than 600 bytes, unless TAPE=NSYSOUT has been specified in the CONFIG statement OWDFLT).

ERROPT

Specifies the action the system takes when an input/output error occurs.

SKIP	Applies only in sequential mode and means the record is ignored.
ABORT	The step is aborted (SEV=3).
IGNORE	Applies only to tapes. The I/O failure is ignored and the data returned is the part of the record which has been read, padded on the right with conventional characters. If SEQUENTIAL or DYNAMIC access is being used, the current record pointer points to the next valid record.
RETCODE	The system produces an abnormal return code signalling an I/O error (for example, IOFAIL). No further processing of the file is possible. The current record pointer is undefined in cases where SEQUENTIAL or DYNAMIC access is being used.

WRCHECK

Specifies that each block written to the file is verified at the time of writing (for queued files only).

OPTIMIZE

Enables fast read and write access mode, for UFAS sequential files. For further information, refer to the *Batch Booster User's Guide*.

BPIOC

If BPIOC is specified, file caching is bypassed. This parameter overrides all other cache options specified in the catalog or assignment parameters. For more information on file caching, refer to the *Large Memory Cache User's Guide*.

FRIOC

If FRIOC is specified, file caching is carried out. This parameter overrides all other cache options specified in the catalog or assignment parameters. For more information on file caching, refer to the *Large Memory Cache User's Guide*.

RAHEAD

By default, for cached files with sequential organization, the "read-ahead technique" is used when a file block is first referenced, so that the next 16 file blocks are also read. The RAHEAD parameter can be used to force the read-ahead technique to be used, whether the file organization is sequential or not. For more information on file caching, refer to the *Large Memory Cache User's Guide*.

KEYLOC

Applies to UFAS indexed files. KEYLOC defines the relative position (from the left) in the record of the first character of the data record key. The first byte of the record is byte 0.

NOTE: There is a difference of 1 byte between the KEYLOC described in the DEFINE basic JCL statement or in the OUTDEF parameter group, and the KEYLOC defined in the PREALLOC statement. For DEFINE and OUTDEF, the first byte of a record is byte 0; for PREALLOC, the first byte of a record is byte 1.

DLREC

Deleted records (those beginning with "FF"X are considered to be deleted) are allowed in queued files.

NDLREC

Deleted records are not allowed in queued files.

TRUNCSSF

The SSF header and SSF control records are suppressed.

NTRNCSSF

The SSF header and SSF control records are not suppressed.

COMPACT

Records must exist in compacted form on the media.

NCOMPACT

Records must exist in non-compacted form.

FPARAM

Forces the overriding of certain values by parameters specified in the DEFINE statement.

LTRKSIZE

The logical track size of the file.

FILEORG

Defines the file organization (access method).

Abbreviation	File Organization
SEQ	Sequential
RELATIVE	Relative
INDEXED	Indexed
LINKQD	Queued link
NONE	None

KEYSIZE

Gives the length in bytes of the record key. The maximum value allowed is 255.

CISIZE

For UFAS files, this gives the size in bytes of the control interval.

CIFSP

For UFAS indexed files; gives the control interval free space as a percentage.

BUFPOOL

For a UFAS disk file, BUFPOOL identifies a "pool" of buffers (a group of buffers with the same name) used for the file.

READLOCK

Applies only if SHARE=MONITOR has been specified for a UFAS, or IDS/II disk file. It defines the lock protocol that you wish to apply. The locks apply at the CI level (not the file level) and are freed at the end of the commitment unit in which they are applied. For more details see the *GAC-Extended User Guide*. The possible values are as follows:

NORMAL — While a CI is being shared under this lock, the following can access it simultaneously.
(i) (a) Several NORMAL readers
or
(b) One NORMAL writer
AND
(ii) Several STAT readers
AND
(iii) No EXCL user (whether reader or writer)

EXCL — While a CI is being shared under this lock, the following can access it simultaneously:
(i) (a) One EXCL reader
or
(b) One EXCL writer
AND
(ii) No NORMAL user (whether reader or writer)
AND
(iii) Several STAT readers.

STAT — While a CI is being shared under this lock, the following can access it simultaneously:
(i) (a) Several NORMAL readers
or
(b) One NORMAL writer
OR
(ii) (a) One EXCL reader
or
(b) One EXCL writer
AND
(iii) Several STAT readers

Using the normal Boolean convention, these rules may be summarized as follows:

NORMAL is (ia OR ib) AND (ii) AND (iii)

EXCL is (ia OR ib) AND (ii) AND (iii)

STAT is (ia OR ib) OR (iia OR iib) AND (iii)

NORMAL and EXCL are mutually exclusive for a given CI at any time. STAT readers can always read and are not concerned that updating is taking place while they do so.

LOCKMARK

Applies only if SHARE=MONITOR is specified for a UFAS disk file.The current record pointer may be retained between consecutive commitment units.

For more details, see the *GAC-Extended User's Guide*.

FILEFORM

Describes the file. Most formats are self-explanatory.

UFAS	applies to UFAS files.
LINKQD	applies to queued link files.
ANSI	applies to an ANSI magnetic tape file, written in ASCII data code.
NSTD	the tape file format is unknown. Data is delivered by blocks, including tape marks. Labels are not processed by the OPEN and CLOSE routines.
NONE	The file format is not defined.

BLKSIZE

Specifies the block size in bytes.

RECSIZE

Specifies the record size in bytes.

RECFORM

Specifies the format of the record.

Abbreviation	Record Format
F	Fixed record format
V	Variable record format
U	Undefined record format
FB	Blocked fixed record format
VB	Blocked variable record format

FUNCMASK

Used to specify the tape controller capabilities, pack/depack options for UFAS tape files. In particular, the FUNCMASK parameter is used for nonstandard tape files for which the message "MEDIA RECOGNITION IMPOSSIBLE" has been generated by GCOS. The specified mask, containing eight hexadecimal characters, is loaded in the Magnetic Tape Controller when the file is opened. See the *UFAS User's Guide* for details.

BLKOFF

Allows the program to bypass any unwanted information present on certain magnetic tape files at the beginning of each block. It applies in particular to ANSI (version 1) tape files. The specified value defines the size of the unwanted data (the block offset) and corresponds to the relative position in each block of the first byte of required data.

NOTE: The value of BLKSIZE must include the block offset, if any.

CONVERT

Data code is converted for tapes.

NCONVERT

Data code is not converted for tapes.

DATACODE

Defines the data code. Possible values are: BCD, ASCII and EBCDIC.

COLLATE

Specifies the collating sequence in the case of UFAS disk files. The possible values are: BCD, ASCII and EBCDIC.

BSN

Block serial numbers are to exist on a tape file.

NBSN

Block serial numbers are not to exist on a tape file.

NSORTIDX

Disables the updating of secondary indexes of UFAS indexed files in UPDATE processing mode. In this case SORT_INDEX utility must be executed after the close of the file to restore it safely, because file status is unstable.

PRINTER=printer-options
TN=terminal-options

These options are described in the following sub-sections.

Printer Options

The following options allow you to specify editing parameters for files to be printed:

SLEW

Any paper move instructions (slew control) are obeyed when the file is printed.

NSLEW

When the file is printed by the Output Writer, every skip function is transformed to "skip to next line": (that is, for this statement each paper move instruction is reduced to a move to the next line).

MARGIN

Specifies the number of blanks that are to be inserted at the beginning of each line of print. The default value is 0 (zero). The value that can be specified is limited by the maximum line width of the printer that is used.

PRDEN

This specifies the vertical print density as six or eight lines per inch. The default value is the value given by the form number in MEDIA, or, if MEDIA is not specified, the default value is that of the installation's default printer.

The FORMHT parameter is specified in lines per page. Thus, if a page is defined to contain n lines, it will cover n/6 inches if PRDEN=6 is given, or n/8 inches if PRDEN=8 appears. If PRDEN is to be used to increase the use of the paper, the FORMHT parameter must be changed accordingly. For example, if PRDEN=6 and FORMHT=54, the page occupies 9 inches. If PRDEN is to be changed to PRDEN=8 then FORMHT should be changed to FORMHT=72 to maintain the use of all 9 inches. Otherwise, the 54 lines would occupy about 7 inches.

FORMHT

Specifies the form size as a number of lines. The maximum value is 255. See PRDEN above.

HOF

Head of form: this gives the line number of the first line to be printed on the page. The maximum value is 255, but see the note below.

FF1

First level of full form; this gives the line number of the last line to be printed on the page. The maximum value is 225, but see the note below.

FF2

Second level of full form: when this line is reached or passed, a return code message is produced. The maximum value is 225, but see the note below.

NOTE: The following restrictions apply to the values of the form control options:

```
1 <= HOF <= FF2 <= FFF1 <= FORMHT
```

CHi

The Vertical Format (VF) tape of a printer is a physical loop containing channels that are punched in a certain position, to control vertical paper movement. Since DPS7000 printers do not have VF tapes, vertical paper movement is controlled by a simulated vertical format unit (VFU). This VFU functions in the same way as a standard 12-channel VF tape, with a limitation of 20 stop levels per form, shared among the 12 channels. For each CHi (where i is a number between 1 and 12) you can specify one or more stop levels, in increasing order. Each stop level gives a line number (up to 3 digits) whose value must be less than or equal to the value of FORMHT. The maximum number of stop levels for all the CHi entries is 20.

Example:

```
CH1=(10, 25, 50), CH2=(5, 15)
```

This specifies stop levels at lines 10, 25, 50 on channel 1 and lines 5 and 15 on channel 2.

Terminal Options

The file exists on a terminal. The following options allow you to specify parameters for terminal files. For more details see the *IOF Terminal User's Manual*.

PROMPT

This parameter defines a prompting message of maximum length 12 characters.

NPROMPT

No prompting will be used.

EOF

This parameter defines a string of four characters used to indicate the end of file in input mode.

The rest of the TERMINAL options have the same meaning as the PRINTER options of the same name.

NMSG

All messages to the console by any other agent than the interactive step are held until the file is closed.

MSG

Messages to the console are not held.

4.8 $ENDDATA

4.8.1 Function

Terminates the input data after a $DATA statement.

4.8.2 Enclosure

Outside a job.

4.8.3 Format

```
$ENDDATA;
```

4.8.4 Description of Statement

This statement closes the input data enclosure after a $DATA statement when the END parameter was not specified on the $DATA statement. The record containing $ENDDATA is not treated as an input data record to be loaded by $DATA. The $ sign is mandatory and must be located in the first position of the record. The entire $ENDDATA statement, including the semicolon, must be on one record.

4.9 $ENDINPUT

4.9.1 Function

Terminates an input enclosure.

4.9.2 Enclosure

Job.

4.9.3 Format

```
$ENDINPUT [input-enclosure-name];
```

4.9.4 Description of Statement

The $ENDINPUT statement marks the end of an input enclosure (See $INPUT statement). When the statement is recognized by the stream reader, the SYSIN subfile containing the input enclosure records is closed.

The $ sign must be present and must be located in the first character position of the record.

4.9.5 Optional Parameter

input-enclosure-name

This is meaningful only if the END=MATCH parameter was declared in the $INPUT statement corresponding to this $ENDINPUT statement. If both input-enclosure-names match, the $ENDINPUT is considered to be the closing statement of the input enclosure.

The $ENDINPUT statement is treated as data contained in the current input enclosure if the input-enclosure-name is not identical to one specified in a previous $INPUT statement.

4.10 $ENDJOB

4.10.1 Function

Terminates a job enclosure.

4.10.2 Enclosure

Job.

4.10.3 Format

```
$ENDJOB;
```

4.10.4 Description of Statement

The $ENDJOB statement must appear as the last statement of a job which starts with a $JOB statement. Jobs submitted with the RUN statement or the EJR operator command do not necessarily require a $JOB statement. If $JOB is omitted, then $ENDJOB is not necessary. When a job is being submitted to the system, the statement is recognized by the Stream Reader, which closes the file containing the source JCL statements (see $JOB statement).

If the $ENDJOB statement is required but omitted, then the JCL is translated but the job is aborted. A Job Occurrence Report is produced after JCL translation is terminated.

The $ sign must be present and must be located in the first character position of the record. The entire $ENDJOB statement must be on one record.

4.11 ENDSTEP

4.11.1 Function

Requests execution of the load module named in the corresponding STEP statement. This is the last statement in the step enclosure.

4.11.2 Enclosure

Step.

4.11.3 Format

```
ENDSTEP;
```

4.11.4 Description of Statement

The ENDSTEP statement must appear as the last statement of a step enclosure. Step enclosures are discussed in the STEP statement description. At job execution time, the ENDSTEP statement terminates step initiation procedures (resource allocation) and requests the execution of the load module (program) named in the corresponding STEP statement.

4.12 EXDIR

4.12.1 Function

Defines steps to execute main-operator commands in GCL format for a job running in batch mode.

4.12.2 Enclosure

Job.

4.12.3 Format

```
EXDIR 'command[;command]...';
```

4.12.4 Description of Statement

The main-operator commands described in the *System Operator's Guide* may be submitted this way if the batch job is running under a project that has the attribute MAIN. When this is not the case, only the IOF directives described in the *IOF Terminal Reference Manual* may be executed. The maximum length of the command-string is 255 bytes. Commands in the command string are separated by a semi-colon (;). The GCL directives MWINLIB BIN, BYE and EXDIR may not be executed with EXDIR. The image of the submitted commands and the error messages (if any) is provided in the Job Occurrence Report.

4.13 EXECUTE

4.13.1 Function

Translates and executes a stored sequence of JCL statements during step execution.

4.13.2 Enclosure

Job.

4.13.3 Format

```
EXECUTE   {[INFILE =]  {sequential-input-file-description)     }
          {                                                    }
          { member-name  [INLIB =]  (input-library-description)}

          [VALUES =   ([parameter-value 1] [parameter-value2]    ]
          [    [keyword1 = keyword-value 1]                      ]
          [    [keyword2 = keyword-value 2]  .. ]               )]

          [PRTFILE = (print-file-description)]
          [    [PRTDEF = (define-parameter)] ]

          [PRTOUT  = (sysout-parameters)]

          [SIZEOPT = (size-parameters)]

          [STEPOPT = (step-parameters)] ;
```

4.13.4 Description of Statement

The EXECUTE statement creates a job step that uses the JCL Translator at job execution time to translate a stored sequence of JCL statements. After translation, these statements are submitted for execution. Once the statements have been executed, processing of the original job continues with the statement following the EXECUTE. The effect of EXECUTE is to dynamically invoke, during job execution, a set of JCL statements.

A JCL sequence handled by the EXECUTE statement may contain other EXECUTE statements. The effect of the execution of each one is to do a JCL translation and then to execute the translated statements.

The JCL sequence can also refer to input enclosures contained in the original job description and can accept parameter values from the EXECUTE statement (the VALUES parameter).

For each EXECUTE statement, a JCL translation report is produced. By default, this report is printed as a standard SYSOUT file, but you may request that it be stored in a specified file.

NOTE: EXECUTE defines an entire step, but is considered as part of the basic operating system and treated in this manual as a Basic JCL statement.

Comments and Restrictions

It is important to distinguish between INVOKE, a statement interpreted at JCL translation time, and EXECUTE, a statement activated at execution time. INVOKE is static in the sense that a sequence of JCL statements is inserted into a job description at translation time and thus becomes part of the job description. EXECUTE is dynamic since the sequence to be executed is not identified and translated until execution time and, once the sequence has been executed. It has no further significance to the job description that contains the EXECUTE statement. The first three comments/restrictions below are a result of this basic difference between the two statements.

If a particular EXECUTE statement is executed several times in a job (for example, by means of a JUMP statement), it is possible for a different version of the sequence to be created each time (for example, if the file that contains the sequence is modified between each execution of the EXECUTE statement).

The library search path, if any, defined in the job that contains the EXECUTE statement is not available when the stored statements are executed. If a library search path is required (for example, for the execution of a LIBMAINT statement contained in the JCL sequence), it must be declared within the sequence itself (by means of the Library Management statement LIB).

No PREFIX statement present in the original job is valid within the stored sequence. Therefore, if the job's PROJECT identification is not to be used as the prefix within the sequence, a PREFIX statement must be specified there.

The sequential file that contains the sequence to be translated and executed must be in either DATA or DATASSF format.

The statements recognized by the Stream Reader ($JOB, $ENDJOB, $INPUT, $ENDINPUT, $DATA, $ENDDATA, $SWINPUT) must not appear inside a JCL sequence called by EXECUTE. Therefore, the sequence must not contain input enclosures.

The sequence must contain one or more complete step enclosures. Extended JCL Statements (including ATTACH, EXECUTE and RUN, but not including SORTWORK) can be considered as complete step enclosures.

If a warm restart occurs during the execution of the EXECUTE statement, the restart is always done from the beginning of the step (that is, the entire translation is repeated). If a warm restart occurs during the execution of the translated sequence, the standard repeat rules apply (see the *System Administrator's Manual*).

If an error occurs during translation of the JCL sequence, the system sets the step severity to 4. The abnormal termination of the job can be avoided by means of a JUMP statement with a suitable SEV value (see JUMP).

4.13.5 Parameters

4.13.5.1 Mandatory Parameter

sequential-input-file-description

Identifies the file that contains the JCL sequence to be EXECUTED. The format is given in Section 2. The file must be in DATA or DATASSF format.

4.13.5.2 Optional Parameters

VALUES

Specifies the parameter values that will be passed to the JCL sequence. See the INVOKE statement for a description of the parameter.

PRTFILE

The print-file-description identifies the permanent SYSOUT file which will contain the JCL Translator report. If this parameter is omitted, the report is sent to a standard SYSOUT subfile. The format is the same as for sequential-file-description above, with the addition, optionally, of the EXPDATE and DENSITY parameters (see ASSIGN); furthermore, the default value of the ACCESS parameter is WRITE.

PRTDEF

Allows you to specify editing parameters for the SYSOUT file identified by the PRTFILE parameter. The parameters of the parameter group define-parameters are listed and described in Section 3.

PRTOUT

Allows you to specify output handling parameters for the SYSOUT file identified by the PRTFILE parameter. Any of the parameters of the SYSOUT statement (except the internal-file-name) may be specified for the parameter group sysout-parameters. These are described in Section 3.

SIZEOPT

Allows you to specify size-parameters which are listed in Section 3 and described under the SIZE statement.

STEPOPT

Allows you to specify parameters to control the execution of the EXECUTE step-defining statement. Any of the STEP parameters, except DUMP=ALL, DEBUG and the OPTIONS parameter, may be specified for the parameter group step-parameters.

Example 1:

```
EXECUTE (EXEC.LIB, SUBFILE=SEQ1), VALUES =(X12, FILE=MY.LIB);
```

The effect of the above example is to translate, at step execution time, the JCL sequence stored in member SEQ1 of library EXEC.LIB, substituting the value X12 for every occurrence of &FILE, and to execute the translated statements.

Example 2:

```
EXECUTE JCLSEQ.EXFL, STEPOPT=(DUMP=DATA, LINES=50);
```

The above statement will translate the JCL sequence stored in the cataloged file JCLSEQ.EXFL and introduce the translated statements for execution. A limit of 50 lines of printer listing will apply to the EXECUTE step (that is, to the translation of the sequence), if the EXECUTE step terminates abnormally, the system will dump the data segments.

4.14 $INPUT

4.14.1 Function

Opens an input enclosure and names the SYSIN subfile into which the records of the enclosure are to be stored.

4.14.2 Enclosure

Job.

4.14.3 Format

```
$INPUT     input-enclosure-name

           TYPE =[{DATA   }]
                 [{COBOL  }]
                 [{RPG    }]
                 [{FORTRAN}]
                 [{JCL    }]
                 [{COBOLX }]
                 [{GPL    }]
                 [{DATASSF}]

           [FORMAT = (digits3, digits3, digits3, digits3)]

           [END =  {ENDINPUT }]
           [       {DOLLAR   }]
           [       {EOF      }]
           [       {MATCH    }]
           [       {'string8'}]

           [ENDCHAR = 'string1']

           [CONTCHAR = 'string1'}

           [PRINT]   [CKSTAT]

           [{CVALUES}]
           [{JVALUES}]

           [INFILE ={(diskette-file-name                      }]
           [        {        {  *                          } }]
           [        {MEDIA ={(volume-name [volume-name] ...)})}];
```

4.14.4 Description of Statement

The $INPUT statement, together with a corresponding $ENDINPUT statement, are the boundary statements of an input enclosure. An input enclosure allows data or processor-specific commands to be included inside a job description. This data is either input to a user program or system utility, or it may be a source program, which is the input to a compiler (see the TYPE parameter).

The $INPUT statement is recognized by the Stream Reader. The data that is read is stored in a temporary subfile of the system SYSIN file. Each SYSIN subfile is identified by the input-enclosure-name which is given in the $INPUT statement.

The $ sign must be present and must be located in the first character position of the record.

An input enclosure may occupy any position in the job enclosure other than within a step enclosure. Furthermore, one input enclosure may be made available to any number of steps in a job.

The ASSIGN statement for an input enclosure has the form:

```
ASSIGN internal-file-name, *input-enclosure-name;
```

Comments and Restrictions

Input-enclosure-names must be unique within a job enclosure.

The total number of input enclosures and ASSIGN statements (including the ASSIGN statements for input enclosures) must not exceed 31 in any step.

No more than 252 input-enclosure files in JCL, or 251 input-enclosure files in GCL, may be associated with a particular job. Note that a single definition of an input enclosure (that is, the data records between $INPUT and $ENDINPUT) can give rise to more than one input enclosure, if input enclosure parameter substitution is used (see below).

Use of Parameter Substitution in Input Enclosures

An input enclosure can contain parameter references (of the form "&string") that are defined in the current job description within one or more SET_VALUES statements or MODIFY_VALUES statements. Parameter substitution can take place if (and only if) either the JVALUES parameter is specified in the associated $INPUT statement.

Each time a reference to the input enclosure is encountered during JCL translation, a new version of the input enclosure is created: each parameter reference is replaced by the value associated with it. The updated input enclosure will be used at execution time.

If CVALUES is specified in $INPUT, the system substitutes the values specified or applies them, by default, at the current level of nesting (see INVOKE). If JVALUES is used, the values that apply at the level of the job description (for example, as specified by a RUN statement) are used, irrespective of the level of nesting of the associated ASSIGN statement.

A detailed explanation of the use of parameter substitution in input enclosures is given in the *JCL User's Guide*.

Restrictions

Whereas input enclosures containing parameter references can be associated (by ASSIGN) with invoked JCL sequences, parameter substitution is not permitted within an invoked input enclosure (see INVOKE).

Parameter substitution is permitted only within input enclosures of type DATA or DATASSF.

An input enclosure containing either of the parameters CVALUES or JVALUES cannot be used more than 31 times in ASSIGN statements of any number of steps.

Protection From Parameter Substitution

A contiguous sequence of records in an input enclosure can be "protected" from the parameter substitution mechanism: a record containing the five characters "//BOD" (starting in column 1) must precede the first record to be protected, and a record containing the five characters "//EOD" in the first 5 columns must follow the last record to be protected. Each time a version of the input enclosure is created, these two delimiting records are removed from the sequence, but no parameter substitution is made within the sequence.

This feature is particularly useful if an input enclosure contains MAINTAIN_LIBRARY editing statements, where & characters are not to be interpreted as parameter references (see the *Text Editor User's Guide* for details of the MAINTAIN_LIBRARY editing facilities).

Order of Application of Parameters

The parameters of the $INPUT statement are applied in the following order:

CKSTAT

If CKSTAT is specified then the $SWINPUT feature is applied first, except in the following two cases:

(1) END=DOLLAR has been specified, in which case a $SWINPUT encountered in the input enclosure is interpreted as the end of the input enclosure and not as a $SWINPUT statement.

(2) END='$SWINPUT' or END='$SWI' has been specified, in which case a $SWINPUT or a $SWI encountered in the input enclosure is interpreted as the end of the input enclosure and not as a $SWINPUT statement, unless the $SWINPUT (or $SWI) and its parameters appear on the same record.

Example:

```
$SWINPUT XXXX END='$SWINPUT';
$SWINPUT FILE-A;
```

The second $SWINPUT statement above is treated as a $SWINPUT statement, and not the end of the input enclosure. However,

```
$SWINPUT XXXX END='$SWINPUT';
$SWINPUT
FILE-A;
```

the second $SWINPUT statement above is treated as the end of the input enclosure and not as a $SWINPUT statement, because its parameter is not on the same record.

CONTCHAR; and/or ENDCHAR

This feature is applied next.

FORMAT

Applied to the result of the preceding two features.

Parameterization

Input-enclosure parameterization is applied last.

4.14.5 Parameters

4.14.5.1 Mandatory Parameter

input-enclosure-name

The name with which the programmer refers to the SYSIN subfile that contains the records of the input enclosure. The name must not exceed 16 characters and may contain letters, numbers (0-9) and the hyphen and underscore characters.

The input-enclosure-name must be unique within a job.

4.14.5.2 Optional Parameters

TYPE

Specifies the type (and implied format) of the input enclosure records. The types RPG, FORTRAN, JCL, COBOLX, GCL and GPL (and their implied formats) are described in the *Library Maintenance Reference Manual*. The other values are as follows:

DATA — Data for a user program or system utility is entered with no modification of the input format and no header added to the beginning of each input record. This value can also be used when COBOL or FORTRAN or RPG source statements are entered. If TYPE and FORMAT are omitted, DATA applies by default.

COBOL — This can be used for a COBOL source program. An SSF header is added to the beginning of each input record and processed automatically by the system.

DATASSF — An SSF header is added to the beginning of each input record and processed automatically by the system. Each data record is numbered (in the SSF header), starting at 10 with an increment of 10 for each record. An SSF control record is placed at the beginning of the SYSIN subfile. If FORMAT is specified but TYPE is not then TYPE defaults to DATASSF.
This value can be used:
- when input data to a COBOL object program is to be ACCEPTED from SYSIN;
- when RPG source statements are entered;
- when FORTRAN source statements are entered.

FORMAT

Defines the format of the input records. If this is specified, then TYPE must not be DATA for the output records. If FORMAT is not specified, then the default is the format corresponding to TYPE. These are listed in the *Library Maintenance Reference Manual*. The 4 values specify respectively:

- The location in the input record of the first digit of the line number. The first location in a record is 1 and leading blanks are treated as zeros. If 0 is specified it means that there are no line numbers.
- The location in the input record of the last digit of the line number. A value of 0 means no line numbers are given.
- The location in the input record of the first text character.
- The location in the input record of the last text character.

END

Indicates to the input reader how the input enclosure is terminated, as follows:

ENDINPUT	input enclosure terminated by the next $ENDINPUT statement. This is the default value, if the input enclosure is not stored on a separate diskette file.
DOLLAR	input enclosure terminated by the next input record which has a dollar sign ($) in column 1. The record is analyzed as a record outside an input enclosure. The statements $HOL, $BIN and $MAP are exceptions. These statements are processed as part of the input enclosure.
EOF	The input enclosure is terminated by the end of file condition on a diskette file. This parameter can be specified only if INFILE references a diskette file and it is also the default value in such a case.
MATCH	input enclosure terminated by the first $ENDINPUT statement which specifies the input-enclosure-name in this statement. This permits the insertion of a complete input enclosure within the input enclosure defined by this $INPUT statement and its matching $ENDINPUT.
string8	The input enclosure ends with the first record which starts with this string. This terminator record is not treated as one of the input enclosure data records. Its character representation must be EBCDIC.

ENDCHAR

Applies only to input enclosure records of DATA or DATASSF type. It can be used with input enclosures where each record contains a particular character as its last non-blank character. The character concerned is specified in the ENDCHAR parameter. This character and all trailing blanks will be removed from the input enclosure records. The value supplied by ENDCHAR may be protected (that is, enclosed in single quotes). Note that a blank may be specified as last-character. This character must be protected.

When ENDCHAR is specified, any record that does not contain the last-character in the required character position is concatenated to the next record. This concatenation continues (up to the maximum record length of 256 characters) until a record that contains the specified character in the last non-blank position is encountered. In other words, ENDCHAR allows you to supply a particular character to signify the end of a record.

WARNING

If ENDCHAR appears in $INPUT, the last record must contain the specified character.

Examples of ENDCHAR:

In the following examples, the lower-case letter b is used to represent a blank character.

Example 1:

```
ENDCHAR=Q

record input (80 characters):
1234Q5678Qbb9bQbbb...........b

record stored (14 characters):
1234Q5678Qbb9b
```

Example 2:

```
ENDCHAR='b'

record input (80 characters):
bVENIbVIDIbVICIbbb...........b

record stored (15 characters):
bVENIbVIDIbVICI
```

Example 3:

```
ENDCHAR= /
records input (80 characters):
AAAbbb..............b
BBB/bbb.............b
CCCbbb..............b
DDD/bbb.............b
EEE/bbb.............b

records stored:
AAAbbb..............bBBB (83 characters)
CCCbbb..............bDDD (83 characters)
EEE                    ( 3 characters)
```

CONTCHAR

Applies only to records of DATA or DATASSF type. For each input record that contains the specified character in the last non-blank character position, this character is deleted together with all trailing blanks, and the record is concatenated to the following one. The continuation-character may be protected by single quotes. A record formed by the use of CONTCHAR must not exceed the maximum length of 255 characters.

Examples of CONTCHAR

In the following example, the letter b is used to represent a blank character.

Example:

```
CONTCHAR=-

records input:
ABCDE-FFF-bbbAXZ-bbb........b
12345-678-bbb999bbb...........b
-XYZbbb..............b

records stored:
ABCDE-FFF-bbbAXZ12345-678-bbb999bbb.......b
-XYZbbb...................b
```

Examples of ENDCHAR and CONTCHAR Used Together

The parameters ENDCHAR and CONTCHAR may be combined in the same $INPUT statement, provided a different character is specified for each one.

In the following examples, the letter b is used to represent a blank character.

Example 1:

```
CONTCHAR= -, ENDCHAR=9

records input:
ABCD9b-bXZ-bbb........b
XZZZ9b-bAB-9bbb..........b

record stored:
ABCD9b-bXZXZZZ9b-bAB-
```

Example 2:

Consider the following input enclosure (80 characters per record):

```
$INPUT, CONTCHAR=+, ENDCHAR=/;
AAAbbb...................b
BBB/bbb...................b
CCCbbb...................b
DDD+bbb...................b
EEE/bbb...................b
FFFbbb...................b
GGG+bbb...................b
HHH/bbb...................b
$ENDINPUT;
```

The following records are stored:

```
AAAbbb.....................bBBB    (83 characters)
CCCbbb.....................bDDDEEE (86 characters)
FFFbbb.....................bGGGHHH (86 characters)
```

WARNING

If the result of the use of ENDCHAR and/or CONTCHAR is a record of length zero, the record is ignored.

PRINT

If this parameter is specified, the contents of the input enclosure are printed in the translation part of the Job Occurrence Report at job introduction time. Up to 200 input records per job can be printed.

CKSTAT

If this parameter is specified, the Stream Reader checks each input enclosure record for a $SWINPUT statement. If such a statement is found, it is interpreted and executed.

CVALUES

Applies only to input records of DATA or DATASSF type. It specifies:

- that the input enclosure contains parameter references;
- that, at execution time, the system will use a version updated by the substitution of parameter values that apply at the current invoked level within the job.

JVALUES

Applies only to input records of DATA or DATASSF type. It specifies:

- that the input enclosure contains parameter references;
- that, at execution time, the system will use a version updated by the substitution of parameter values that apply at job level.

INFILE

Applies only to cases where the job description is stored on a diskette and it specifies where the input data records are to be found.

diskette-file-name

This is the external-file-name of the diskette file which contains the data records.

MEDIA=*

No specific diskette volume name is given. The first volume encountered which contains a file name which matches the diskette-file-name, is taken as the volume. The file concerned may be multi-volume.

MEDIA=(volume-name)

The file is to be searched for on the volume or volumes listed.

If the MEDIA parameter is omitted, then the file is searched for on the diskette volume that contains the job description. By using the INFILE parameters, an input enclosure can be retrieved from diskette files which may be on the same volume as the job description or on subsequent volumes in the input stream. If an input enclosure is stored in a multivolume diskette file then only the volumes containing the first or the last extent may contain job descriptions or other input enclosure files.

The only data retrieved from the intermediate volumes (that is, volumes which do not contain the first or the last extent) of a multivolume diskette file is taken from the extents of that file. No other files on these volumes are accessible during this process. A similar restriction applies to concatenation of multi-extent diskette files (see File Concatenation in the ASSIGN statement). If the END parameter has not been specified, then the input enclosure is closed by $ENDINPUT or by the end of file condition.

Example 1:

```
$JOB MYJOB, USER=SMITH, PROJECT=Q141;

STEP MYPROG, (NEW.MYLIB, DEVCLASS=MS/M500, MEDIA=B015);

ASSIGN FILE1, *MYFILE;

SYSOUT PRTFILE;

ENDSTEP;

$INPUT MYFILE;
     .
     .
     .

  data
     .
     .
     .

$ENDINPUT;

$ENDJOB;
```

In the above example, the input enclosure MYFILE contains data and ends with a $ENDINPUT statement.

Example 2:

If the job in the previous example was being read from a diskette then the $INPUT statement could be as follows:

```
$INPUT MYFILE INFILE=(FILE1, MEDIA=(XYZ1,XYZ2,XYZ3));
```

The input enclosure data will be retrieved from the file FILE1 which will be searched for on the volumes XYZ1, XYZ2 and XYZ3. The enclosure will end when a $ENDINPUT statement is encountered or an end of file condition arises on FILE1.

4.15 INVOKE

4.15.1 Function

Inserts a sequence of JCL statements, stored in a source format as a member of a library or entered from an input enclosure, into the current job description.

4.15.2 Enclosure

Job.

4.15.3 Format

```
INVOKE {*input-enclosure-name1                          }
       { member-name   [{(input-library-description)}]}
       {               [( SYS                        )]}

       VALUES=( [parameter-value1  [parameter-value2]...]
        [keyword1=keyword-value1   [keyword2=keyword-value2]...])

      [UPDATE=*input-enclosure-name2]
       [LIST = ALL];
```

4.15.4 Description of Statement

The INVOKE statement requests that a JCL sequence in source format be entered from a JCL source library or from an input enclosure. If a label is defined inside an invoked sequence, this label cannot be referenced outside the invoked sequence. Furthermore, it is not possible to "jump" outside the invoked sequence. The same label can be defined both inside and outside the sequence with no subsequent ambiguity. In addition, a LIB statement in an invoked sequence applies only to that sequence.

An invoked sequence of JCL may contain other INVOKE statements. Up to nine levels of "nesting" are permitted and these invoked members can be stored in up to fifteen libraries.

An invoked sequence must contain one or more entire step enclosures and cannot contain an input enclosure, a $JOB statement, nor a $ENDJOB statement. The VALUES parameter allows parameter substitution in the invoked sequence, and the UPDATE parameter enables you to edit the invoked sequence before it is translated.

If the invoked sequence contains an assignment of an input enclosure whose $INPUT statement specifies the CVALUES parameter, a new input enclosure will be created by the substitution of the parameter values that apply currently to the invoked sequence (see $INPUT).

Temporary files and permanent files cataloged in private catalogs must not be used, unless the CATALOG is auto-attachable or unless the input stream is submitted using a RUN statement.

4.15.5 Parameters

4.15.5.1 Mandatory Parameters

input-enclosure-name1

The name of the SYSIN input enclosure which contains, in DATA or DATASSF format, the invoked JCL sequence. The corresponding $INPUT statement must not specify CVALUES or JVALUES since parameter substitution is not allowed within an invoked input enclosure.

member-name

Identifies the library subfile (member) containing the JCL statements to be entered. The name must not exceed 31 alphanumeric, hyphen and underscore characters and must start with a letter or a digit.

4.15.5.2 Optional Parameters

input-library-description

Identifies the user source library which contains the named member. See Section 3 for the format of this parameter.

WARNING

If the library file is not resident (that is, it is not on the resident system disk), there is a possibility that the system will be unable to access the file at JCL translation time (for example, if the volume containing the file is not mounted); if this happens, the job will be aborted. This situation can be avoided by the use of an EXECUTE statement (see the *JCL User's Guide* for more details).

SYS

SYS	indicates the resident system library, SYS.HSLLIB. If a library-description is not specified, SYS applies by default.

VALUES

The parameter-value or keyword-value defines the actual value to be used in the invoked JCL sequence in place of the &i (1 < i < 99) or &keyword. This value may be a protected string (that is, any characters enclosed in single quotes except that a single quote inside the string must be entered twice in succession) of up to 128 characters. A parameter-value in a string can be omitted by entering a comma in its place, resulting in two consecutive commas or two commas separated by one or more spaces.

Alternatively, if spaces are being used as separators, missing parameter values can be indicated by #. In this case, the corresponding value is an empty string; however, if a VALUES or MODVL statement exists within the invoked JCL sequence, it provides the default value for the omitted parameter.

If the parameter-value of a VALUES parameter is NIL, the &i or &keyword is not replaced and the corresponding parameter in the invoked JCL is ignored (even if a default value is stated in a VALUES or MODVL statement). Note that &0 and &00 are always replaced by empty strings.

Example:

If the INVOKE statement is:

```
INVOKE SEQ4, SYS, VALUES=(MYFILE1,,NIL,TAPES=3);
```

and the member SEQ4 contains:

```
ASSIGN TESTFILE, NEW.&1, &2, MOUNT=&TAPES;
.........................
VALUES OLD.FIL, 'DEVCLASS=MS/M500,MEDIA=VOLA',TEMPRY;
.........................
ASSIGN FIL2, &1, &2, &3;
```

```
the result will be:
ASSIGN  TESTFILE, NEW.MYFILE1, MOUNT=3;
.........................
VALUES OLD.FIL, 'DEVCLASS=MS/M500,MEDIA=VOLA',TEMPRY;
.........................
ASSIGN FIL2, MYFILE1, DEVCLASS=MS/M500,MEDIA=VOLA;
```

In the above example:

- the value of &2 in the first ASSIGN becomes an empty string.
- for the second ASSIGN, the values of &1 and &3 in the INVOKE (MYFILE1 and "NIL" respectively) apply; the stated values in the VALUES statement (OLD.FIL and TEMPRY respectively) would have applied only if no corresponding values had been specified in the INVOKE.
- the file with the external-file-name NEW.MYFILE1 in the first ASSIGN can be a cataloged, passed, or RESIDENT file whereas the corresponding file in the second ASSIGN is an uncataloged file.

UPDATE

Specifies that the "Mini-Editor" commands stored in the input enclosure identified by *input-enclosure-name2 must be applied to the invoked JCL sequence before translation. The use of the Mini-Editor is described in the *JCL User's Guide*.

The corresponding $INPUT statement must not contain the CVALUES parameter or the JVALUES parameter (that is, parameter substitution not allowed within the input enclosure). Note that the invoked JCL sequence may itself be an input enclosure.

Example 1:

```
INVOKE SEQ1, INV.LIB, VALUES=(X12, FILE=MY.LIB),
    UPDATE=*UP;
```

In the above example, the contents of member SEQ1 of the library INV.LIB are updated according to the editing commands contained in input enclosure UP; then the VALUES replacements are performed on SEQ1 and the final version is invoked into the current job description.

Example 2:

```
INVOKE *CRDSEQ, UPDATE=*EDITS;
```

In the above example, the contents of input enclosure CRDSEQ are edited from the commands in input enclosure EDITS and the result is invoked into the current job description.

LIST=ALL

The source JCL and all expansions of JCL sequences are listed, along with error messages.

4.16 $JOB

4.16.1 Function

Marks the beginning of a job enclosure and gives job identification.

4.16.2 Enclosure

Job.

4.16.3 Format

```
$JOB        job-name

              [USER = user-name]

              [PROJECT = project-name]

              [BILLING = billing-name]

              [NSTARTUP]

              [LIST   = {SOURCE}]
              [         {NO    }]
              [         {ALL   }]

              [JOR    = {NORMAL}]
              [         {NO    }]
              [         {ABORT }]

              [CLASS  = {identifier2}]

              [{HOLD             }]
              [{HOLD  =  digits2}]

              [HOLDOUT]

              [PRIORITY = digit1]

              [RECSIZE  = {110    }]
              [           {digits3}]

              [REPEAT]

              [HOST = {name4}]
              [JOBLANG  = {GCL}]
              [           {JCL}]

              [EXPVAL];
```

4.16.4 Description of Statement

The $JOB statement, if used, must be the first statement of a job description. Its function is primarily to provide identification in terms of user-name, project-name and a symbolic job-name. The $JOB and $ENDJOB are not needed if only one job is submitted using a RUN statement or an EJR operator command.

The $JOB statement is recognized by the Stream Reader, which stores the job description statements in the backing store. The $ sign must be present and must be located in the first character position of the record except when the job is submitted from a source using a RUN statement or the EJR operator command.

If a valid site catalog exists, any values specified in the $JOB statement for USER, PROJECT and BILLING must correspond exactly with the entries in the site catalog. The billing of EJR command is significant only if the submitted job does not contain the $JOB statement.

4.16.5 Parameters

4.16.5.1 Mandatory Parameters

job-name

The name by which the job is known to the user. The name is composed of up to eight alphanumeric, hyphen and underscore characters. The first character must be alphanumeric. The job-name must appear on the same record as the statement name ($JOB). The job is known to the system by its Run Occurrence Number (RON).

If there is no $JOB statement, then the job-name is the member-name (RUN member or EJR) or external-file-name (RUN efn). If the name exceeds 8 characters, only the first 8 are used.

USER

Specifies the name of the user. The name is composed of up to twelve alphanumeric, hyphen and underscore characters. The first character must be alphanumeric.

The USER parameter is not mandatory if it is submitted in any of the following ways:

- EJR operator command; in this case, the submitter identification is taken as the default.
- RUN statement; in this case the IOF user or spawning job user is taken as the default.

Apart from these three exceptions, if the USER parameter is omitted, the job execution is not requested. The job is given a run occurrence number (RON), and the Job Occurrence Report is produced.

If Access Rights have been implemented the USER value may be restricted as follows:

- If the submitter is the main or station operator, any USER value can be given;
- If the submitter is a batch or IOF user, the USER must be the submitter.

4.16.5.2 Optional Parameters

PROJECT

Specifies the name of the project associated with the given user. The name can consist of up to twelve alphanumeric, hyphen and underscore characters. The first character must be alphabetic or numeric.

> **WARNING**
>
> The PROJECT parameter is mandatory in the $JOB statement if there is no default value given in the site catalog, or if no site catalog exists.

BILLING

Specifies the name used for accounting and control of the current PROJECT. The name is composed of up to twelve alphanumeric, hyphen and underscore characters. The first must be alphabetic or numeric. If BILLING is omitted and no default value in the site catalog is associated with the current project, the PROJECT name itself is used for the BILLING in the job report banner.

> **WARNING**
>
> The BILLING parameter is mandatory in the $JOB statement if there is no default value given in the site catalog, or if no site catalog exists.

NSTARTUP

Start-up sequences are logically inserted after the $JOB statement if they exist in the start-up source library SITE.STARTUP. A start-up sequence may contain all the system level commands and directives which may be executed in batch mode. In Batch, two start-up sequences may be attached to a project in the catalog, a mandatory start-up sequence and an optional start-up sequence. The available start-up sequences are the following:

```
SITE_B

<project>_B

<project>_<user>_B
```

Rules for Implementing Start-up Sequences:

1. Any of the three start-up sequences can be specified as mandatory.

2. Only <project>_B and <project>_<user>_B can be specified as optional.

3. The mandatory start-up is executed before the optional start-up.

4. The optional start-up sequence can be inhibited by specifying NSTARTUP in $JOB, STARTUP =0 in EJR or RUN_JOB, or using the field STARTUP of the input structure if the programmatic interface is used.

5. If the project is SYSADMIN, and NSTARTUP is specified, then both the mandatory and optional startup sequences are inhibited.

LIST

Allows you to select the type of information to be printed when listing the JCL on the Job Occurrence Report.

SOURCE	The source JCL, inserted Stream Reader statements, records inserted using a $SWINPUT statement with the CONSOLE parameter, and error message are listed.
ALL	The source JCL and all expansions of JCL sequences which are entered using INVOKE, $SWINPUT, and project start-up are listed, along with error messages.
NO	Only Stream Reader statements, comments and error messages are listed.

If Access Rights have been implemented and you do not have the READ right over the file(s) containing the information to be printed, then LIST is ignored.

JOR

Determines the situations in which the Job Occurrence Report (JOR) is to be produced. The possible values are:

NORMAL	The JOR is printed. This is the default.
ABORT	The JOR is printed only if the job aborts.
NO	The JOR is not printed.

CLASS

The job is attached to the job class defined by one or two letters. There are 16 job classes which are defined by a single letter between A and P and 416 job classes of two letters from AA to PZ. The job class allows you to define the default scheduling and execution priorities and a maximum multi-programming-level for jobs of the class. The operator can suspend and reactivate a given class, permitting flexibility and efficient use of machine resources. The use of job class allows you to control job-selection processing and to manage the serial execution of jobs.

The class of a job may be restricted through the site catalog based on its PROJECT identification. Each project can be attached to a batch default class and an IOF default class. Up to 26 classes can be attached to a project. Any job submitted with a job class that does not exist, or that is not accessible to the project, is launched in class P.

HOLD

Specifies that, after JCL translation, the job has "hold" status. The job cannot be scheduled for execution until the operator issues a RELEASE_JOB command (RJ) or a job which is executing suppresses the HOLD with a JCL statement RELEASE.

If a value is specified with the HOLD parameter, this becomes the HOLD count for the job. This is decremented by 1 each time a JCL statement RELEASE is issued. The job is then made available for scheduling once the HOLD count becomes 0 (zero). The HOLD count may be forced directly to 0 if an RJ operator command is issued with the STRONG option.

HOLDOUT

If this parameter is present, output files produced by the job are not printed automatically but are held back until released by the RELEASE_OUTPUT command (RO).

PRIORITY

Specifies a one-digit scheduling priority between 0, the highest priority, and 7, the lowest priority. Jobs are scheduled according to this priority.

The scheduling priority of a job may be restricted through the SITE catalog based on its PROJECT identification. If your specified priority exceeds the maximum value allowed for the PROJECT, then this maximum value is used instead and a warning message is issued.

Within a specified priority, jobs are scheduled on a first-in-first-out (FIFO) basis.

NOTE: A job will not be scheduled if the maximum number of jobs in the same class is already scheduled.

The default value for PRIORITY depends on the job class.

RECSIZE

Applies only if input comes from a KDS terminal. It specifies the record length of the transmitted diskette files. The default value is 110.

REPEAT

Specifies that the job may be repeated from the beginning, at the discretion of the operator, after a warm restart. It is meaningful only in the case where the step in which the abort occurred does not itself have the REPEAT option. If there is a STEP REPEAT option, the step will be repeated (on the decision of the operator). If not, a repeat of the whole job will be considered.

If the REPEAT option is omitted, the entire job cannot be repeated.

HOST

Identifies the site on which the job is to be run, if different from the site where the JCL is entered or stored. It consists of up to 4 alphanumeric characters. The site specified in this parameter must be a DPS-7 or DPS-7000 system.

JOBLANG

Identifies the Command Language used for the JOB description:

GCL	GCOS Command Language
JCL	JOB Control Language

When omitted; the value given at submission time, if any, is used. If this has also been omitted, the JOBLANG value defaults to JCL.

EXPVAL

Specifies that the expansion of the values in the JCL of the job is shown in the Job Occurrence Report. This parameter is only applicable when JOBLANG = JCL.

Example:

```
$JOB DMJOB, USER=DJM, PROJECT=MECTP, HOST=BP3C;
```

In the above statement, job DMJOB will be transferrred to the system BP3C and then executed on that system.

4.17 JUMP

4.17.1 Function

Alters, conditionally or unconditionally, the sequential execution of the JCL from the current statement to another statement. Provides facilities to avoid job abort when a step terminates abnormally.

4.17.2 Enclosure

Job or step.

4.17.3 Format

```
JUMP          {label-name}
              {CONTINUE  }

              [{      }       {EQ}              ]
              [{STATUS}       {NE}              ]
              [{SEV   }       {GE}       digits5];
              [{SWi   }       {GT}              ]
              [{IOF   }       {LE}              ]
              [{      }       {LT}              ]
```

4.17.4 Description of Statement

The JUMP statement alters, conditionally or unconditionally, the JCL execution sequence from the current statement. Control normally passes to the labelled JCL statement corresponding to the label-name specified in the JUMP statement. If CONTINUE is specified, the next JCL statement is executed. This is useful particularly in the case of a step abort (see below).

The JUMP statement may test the status or severity code passed on by a step, or a system switch that you previously set with a LET statement or by a COBOL program (external switch). The jump will or will not be performed depending on the current value of the code or switch tested. If the label-name (or CONTINUE) is specified on its own, an unconditional jump is made. If JUMP is specified within a step enclosure, the corresponding labelled JCL statement must appear within the same step enclosure. Specifying JUMP within a step enclosure allows you to bypass certain JCL within a step as the result of the satisfaction of a condition being tested.

If JUMP is specified within a job enclosure, but outside a step enclosure, the labelled JCL statement must also appear within the job enclosure and outside a step enclosure.

Within a job enclosure a jump can be forward or backward. You can bypass or re-execute as many steps, in sequence, as desired. Within a step enclosure a jump must be forward.

Since the JUMP statement takes effect at job execution, a jump to a Translator statement (for example, INVOKE, VALUES) is meaningless. JUMP (effective at execution) cannot be used to bypass a LIB statement (effective at translation); see Example 4 below. The previous remark also applies to VALUES and MODVL statements; see Example 5 below.

STATUS, SEV, SWi or IOF

A conditional jump is only made when the tested condition is true. If a step is aborted, the remainder of the job is also aborted, unless a JUMP statement testing one of the system variables STATUS or SEV follows the JCL describing the step (that is, JUMP follows ENDSTEP for the aborted step and precedes the next STEP statement, if any). In this case, if a true condition exists, normal execution of the job continues according to the specified label-name. Any JCL statements between the aborted step and the JUMP statement are bypassed. If the reserved label "CONTINUE" is specified, execution continues with the next JCL statement.

4.17.5 Parameters

The parameters of the JUMP statement are positional (that is, they must be given in the order in which they appear in the format).

4.17.5.1 Mandatory Parameters

label-name

The label assigned to the JCL statement which is to be executed next, either unconditionally or if the specified condition is true. Label-name can be up to eight alphanumeric, hyphen and underscore characters in length and must begin with a letter or a digit.

CONTINUE

A reserved label-name indicating that, even in the event of a step abort, normal execution of the job is to continue with the next statement.

4.17.5.2 Optional Parameters

STATUS

Used to test the step completion code which is set by a program. The digits5 value following the specified relational operator (for example, EQ) is initialized to 0 at step initiation time. It can have the following values:

- User normal termination or user step abort (when the REPEAT parameter is not used in STEP) - user specified value between 0 (that is, normal) and 32767. Note that a value of 10000 or more represents abort status.
- Operator abort - 50000
- System abort - greater than or equal to 60000.

NOTE: 0 indicates a normal termination.

SEV

Used to test the step severity code which is set by a user program, a system abort or an operator TERMINATE JOB (TJ) command. The digits5 parameter can assume an integer value between 0 and 6, inclusive. The following table gives the correspondence between SEV and STATUS.

SEV	STATUS
0	0-99
1	100-999
2	100-9999
3	10000-19999
4	20000-32767
5	50000
6	> 60000

SWi

A system switch that you set with the LET command to indicate a program condition which you wish to associate with the value 0 or 1 for the switch. The character i must be an integer value between 0 and 31, indicating the switch which is to be tested. The digits5 parameter in this case, must take the value 0 or 1. These switches are initialized to 0 at job initiation time ($JOB statement) but can be preset for a "spawned" job by means of the RUN statement or for a released job (RELEASE).

IOF

This is a system switch. It is equal to 1 if the job is running in IOF mode and is 0 if the job is executing in batch mode. In this case the digits5 parameter must take the value 0 or 1.

The relational operators have the following meanings:

EQ	EQual to
NE	Not Equal to
GE	Greater than or Equal to
GT	Greater Than
LE	Less than or Equal to
LT	Less Than

Example 1:

```
$JOB A, .....;
STEP LM1, .....;
.
.
.
ENDSTEP;
JUMP FIN, SEV, GT,1;
STEP LM2, .....;
JUMP CAS, SEV, EQ, 1;
ASSIGN IFN1, FILE1, .....;
JUMP ENDLM2;
CAS; ASSIGN IFN1, FILE2,....; ENDLM2: ENDSTEP;
FIN:
$ENDJOB;
```

If step LM1 results in a severity code equal to 0, step LM2 is executed and FILE1 is assigned to IFN1.

If step LM1 results in a severity code equal to 1, step LM2 is executed and FILE2 is assigned to IFN1.

In all other cases, the JUMP statement following the ENDSTEP for step LM2, causes a jump to the JCL statement labelled FIN (that is, $ENDJOB).

Example 2:

In the event of an abort of a load module the job description continues to be scanned sequentially until one of the following JCL statements is encountered:

$ENDJOB	job execution is abnormally terminated.
STEP*	job execution is abnormally terminated.
JUMP	status codes (or switches) are tested and, if the condition is true, job execution continues normally according to the label-name specified.

* This is basic JCL, but the action applies also to an extended JCL statement which is expanded to a step description.

Consider the following test:

```
$JOB FIRSTJOB,....;
STEP LM1, ...;
.
.
.
ENDSTEP;
STEP LM2, ...;
.
.
.
ENDSTEP;
JUMP CONTINUE;
RUN OTHERJOB, ...;
STEP LM3, ...;
.
.
.
ENDSTEP;
$ENDJOB;
```

Step LM2 is executed only if step LM1 terminates normally. However, since the JUMP statement specifies CONTINUE, OTHERJOB is introduced and step LM3 is executed whatever the result of step LM2.

Example 3:

```
$JOB SWITCHER, ...;
 START: STEP LM1, ...;
 .
 .
 .
 ENDSTEP;
 JUMP T3, SW1, EQ, 1;
 LET SW1, 1;
 STEP LM2, ...;
 .
 .
 .
 ENDSTEP;
 JUMP START;
 T3: STEP LM3, ....;
 .
 .
 .
 ENDSTEP;
 $ENDJOB;
```

Since SW1 is initialized to 0 when the job is initialized, the order of execution is as follows:

- Step LM1 is executed followed by step LM2 if SW1 is still 0; otherwise a jump is made to step LM3 (which is labelled T3).
- An unconditional jump is made back to step LM1 (which is labelled START).
- Step LM3 is executed (which is labelled T3).

Example 4:

```
$JOB ... ;
.
.
LIB SL INLIB1=LIBX;
JUMP STEP2, SW1, EQ, 1;
.
LIB SL INLIB1=LIBY;
STEP2: LIBMAINT SL LIB=LIBZ;
.
.
.
.
$ENDJOB;
```

The idea is to use LIBX as the input library (for the LIBMAINT step) if the SW1 is set to 1 and, LIBY if switch SW1 is set to O. The statement LIB is processed at Job translation and the second statement LIB for LIBY overrides the statement LIB for LIBX. At the execution time, the Jump statement is not active and the input library is LIBY.

Example 5:

```
$JOB ........;
.
.
.
VALUES X=FIRSTVAL;
.
.
.                          X=FIRSTVAL, effective
.
.
.
JUMP NEXT, SW1, EQ, 1;
MODVL X=SECONDVAL;
.
.
.
.
.
.
NEXT : STEP ........;
.
.
.                          X=SECONDVAL, effective
.
.
ENDSTEP;
.
.
.
.
$ENDJOB;
```

VALUES and MODVL are effective at job translation. MODVL gives a new value SECONDVAL to &X. This new value is effective through to $ENDJOB (assuming that there are no other MODVL or VALUES except the two shown), irrespective of whether or not the jump to NEXT takes place at execution time. Thus, even if SW1 is 1 and the statements between JUMP and NEXT:STEP are skipped at execution time, &X will still have the value SECONDVAL from the position of the MODVL statement through to the $ENDJOB statement.

4.18 LET

4.18.1 Function

Sets the value of a system switch (or all system switches) or sets the step severity code to an integer value between 0 and 4 inclusive.

4.18.2 Enclosure

Job or Step.

4.18.3 Format

```
LET          {SWi     {0} }
             {        {1} }
             {            }
             {SW   hexa8  }
             {            }
             {SEV  digit1 };
```

4.18.4 Description of Statement

The LET statement can be used to set a system switch value to 0 or 1 to be tested by a subsequent JUMP statement which specifies the same switch.

Alternatively, all system switches can be simultaneously set from left to right to a specified hexadecimal value. You decides with what program condition each switch is to be associated. A third use of LET is to assign, outside a step enclosure, a specific value to the step severity.

4.18.5 Mandatory Parameters

SWi

The SWi parameter specifies one of 32 system switches which must be set to a value of 0 or 1. These switches are normally initialized to zero when the job is initiated. However, the initial switch values can also be set by RUN or RELEASE or by the operator using the operator commands MDJ or EJR.

SW

The SW parameter references all system switches.

Hexa8	A string of up to eight hexadecimal characters is specified as the set of 32 values to which all system switches are simultaneously set, from left to right. For example, FAB00000 sets SW0, SW1, SW2, SW3, SW4, SW6, SW8, SW10 and SW11 to the value 1 and all other switches to zero. (F (hex) =1111 (binary), A=1010 and B=1011).

SEV

The step severity code is set to the value supplied for digit1, an integer between 0 and 4 inclusive. After the execution of the LET statement, the job is in the same state as if a step had terminated with a severity code of the specified value. Therefore if the given value is equal to 3 or more, the job is put in the abort status; this situation can be intercepted by use of the JUMP statement. If several LET SEV statements occur together, only the last one is taken into account.

This form of the LET statement is useful with the INVOKE feature. By inserting LET SEV at the end of an invoked sequence, a user can set a severity value which can be tested in the calling job by means of a JUMP statement appearing immediately after the INVOKE. Thus, the result of the test will be independent of the severity set by the last executed step in the invoked sequence.

Restriction

This form of the LET statement must not occur inside a step enclosure.

4.19 LIB

4.19.1 Function

Defines a search path of input libraries to be used by a subsequent extended JCL statement.

4.19.2 Enclosure

Job.

4.19.3 Format

```
     [{SL }]
     [{CU }]              {TEMP                          }
LIB  [{LM }] [[INLIB1 =] {(input-library-description)}]
     [{SM }]
     [{BIN}]

                     {TEMP                          }
          [[INLIB2 =] {(input-library-description)}]

                     {TEMP                          }
          [[INLIB3 =] {(input-library-description)}]

                     {TEMP                          }
          [[INLIB4 =] {(input-library-description)}]

                     {TEMP                          }
          [[INLIB5 =] {(input-library-description)}];
```

4.19.4 Description of Statement

The LIB statement defines a search path of input libraries to be used in a subsequent extended JCL statement. The library member concerned is searched for, first in INLIB1, then INLIB2, INLIB3, etc. If the same member name is present in more than one library, the first one encountered is processed. Alternatively, you may restrict the search to a specific library by giving its INLIB number in the extended JCL statement.

The LIB statement may appear anywhere in a job enclosure. A LIB statement is effective until overridden by a subsequent LIB statement of the same type (the type being SL, CU, LM, SM, or BIN). The major use of LIB is to define input libraries for a subsequent LIBMAINT statement. However, LIB SL can be used to specify input for the compilers (for example, COBOL, FORTRAN), and LIB CU can be used to indicate input to LINKER.

The maximum length of the search path defined with LIB depends on the type of library.

The maximum values are:

3 for SL, SM, and BIN
1 for LM
5 for CU

There must be no gaps in the search path; that is, if INLIB 3 is given so must INLIB1 and INLIB2. For example, if there are two libraries in the search path these must be INLIB1 AND INLIB2 (and not INLIB1, INLIB3 or INLIB2, INLIB3).

The keywords INLIB1= may be omitted, in which case the first mentioned library becomes INLIB1, the second becomes INLIB2, etc.

In contrast to the way the VALUES JCL statement operates, a LIB statement of a main JCL sequence applies to an invoked sequence (that is, a sequence invoked with the INVOKE JCL statement) until a LIB statement (if any) is encountered within the invoked sequence. A LIB statement in an invoked sequence is effective only within this sequence. Thus after the INVOKE statement, the last LIB statement of the main JCL sequence (that is, the invoking sequence) is again applicable to the main sequence.

A LIB statement in a main JCL sequence is not applicable to an executed JCL sequence (that is, a sequence executed with the EXECUTE JCL statement). A LIB statement within an executed JCL sequence is applicable only within this sequence (thus it is not applicable to the main sequence after the EXECUTE statement).

A LIB statement in a main JCL sequence is applicable to a JCL sequence "inserted" in the main sequence with the $SWINPUT statement. In contrast to the cases of INVOKE and EXECUTE, a LIB statement within the switched sequence (that is, the sequence inserted with $SWINPUT) is applicable to the main sequence after the $SWINPUT statement.

The LIB statement is effective at JCL translation time, consequently the JUMP statement (which is effective at execution time) cannot be used to "jump around" a LIB statement.

For more details on the use of LIB with LIBMAINT, see the *Library Maintenance User's Guide*.

4.19.5 Parameters

4.19.5.1 Mandatory Parameters

input-library-description

Defines an input library to be used in the search path. TEMP means a temporary library. If the keyword INLIBi = is given then i is the rank of the library in the search path. If INLIBi = is omitted then the position of the library in the LIB statement determines its rank in the search path.

4.19.5.2 Optional Parameters

The optional parameter specifies the library type as follows:

SL	for a Source Language Library. This is the default.
CU	for a Compile Unit Library
LM	for a Load Module Library
BIN	for a Binary Library.

Example:

```
   $JOB ..............;
   .
   .
1  LIB SL INLIB1 = (.MYLIB1);
    .                              }
    .                              }
    .                              } scope of 1
    .                              }
    .                              }
2  LIB SL INLIB1 = (.MYLIB2);
    .                              }
    .                              }
    .                              } scope of 2
    .                              }
    .                              }

3  LIB CU INLIB1=(.MYLIBX)
          INLIB2=TEMP;
    .                              }
    .                              }
    .                              } scope of 2 and 3
    .                              }
    .                              }
   $ENDJOB;
```

The first LIB statement defines .MYLIB1 as the input SL Library. The second LIB statement overrides this, so that the input SL becomes .MYLIB2. The third LIB statement defines a CU library search path. As this LIB statement is for a different type of library (namely CU), it does not override the LIB SL statement and .MYLIB2 remains the effective SL library.

```
 LIB
   $JOB ............;
   .
   .
   .
1  LIB SL INLIB1=(.MYLIB1);
    .                          }
    .                          }
    .                          }
    .                          } scope of 1
    .                          }
    .                          }
    .                          }
   JUMP X1 SWO, EQ, 1;
2  LIB SL INLIB1=(.MYLIB2);
              .                }
              .                }
   JUMP X2 SW3, EQ, 1;         }
              .                } scope of 2
              .                }
              .                }
              .                }
3  LIB SL INLIB1=(.MYLIB3)
    .                          }
    .                          }
    .                          }
    .                          } scope of 3
    .                          }
    .                          }
                               }
   X1 : LIBMAINT SL .......;   }
                               }
    .                          }
    .                          }
    .                          }
    .                          }

4  LIB SL INLIB1=(.MYLIB4);
              .                }
    X2 : LIBMAINT SL .......;  }
              .                }
              .                } scope of 4
              .                }
              .                }
              .                }

$ENDJOB;
```

As the LIB statements take effect at translation time, they are not affected by the presence of JUMP statements (effective at execution) in the job. So irrespective of the actual execution path through the job (determined by the JUMP statements), each LIB statement overrides the preceding one and the scope of each is as indicated above.

4.20 MESSAGE

4.20.1 Function

Sends a message to the operator's console at JCL translation time.

4.20.2 Enclosure

Job.

4.20.3 Format

```
MESSAGE    'string105';
```

4.20.4 Description of Statement

The MESSAGE statement allows you to display a message on the operator's console at the time of JCL translation. The maximum permissible length of a message is 105 characters.

4.21 MODVL

4.21.1 Function

Modifies default values for parameters in JCL statements.

4.21.2 Enclosure

Job, INVOKE sequence or an EXECUTE sequence.

4.21.3 Format

```
MODVL  [parameter-value-1   [, parameter-value-2] ...  ]
  [keyword1=keyword-value-1  [keyword2=keyword-value-2] ...];
```

4.21.4 Description of Statement

This statement modifies the current set of parameter values. Only those parameters which explicitly appear in the MODVL statement are affected by it. Otherwise the effect of a MODVL statement is similar to that of the VALUES statement. A MODVL statement does not nullify the effects of a preceding MODVL or VALUES statement except for those parameters which are given values on the current MODVL statement.

The values apply to all following JCL statements until another MODVL statement, a VALUES statement, the end of the current sequence of JCL or the end of the job enclosure. The values set by MODVL can apply to values omitted within the VALUES parameter in INVOKE, EXECUTE and RUN or the EJR operator command.

The MODVL statement cannot be used to set values in the $JOB, $INPUT, $ENDINPUT, $DATA or $SWINPUT statements, nor in a following VALUES statement. Parameter-value-i replaces the value of the positional parameter &i and keyword-value-i replaces the value of the keyword parameter keyword-i. Positional and keyword parameters are discussed in Section 2 and the rules concerning parameter substitution in JCL statements are discussed in the *JCL User's Guide*.

MODVL statements are cumulative in effect that is, each one gives a value to a parameter which previously did not have one, or adds values of new parameters that have not yet appeared, or changes the value of a parameter, but leaves unaltered those parameters that were not explicitly specified on the MODVL statement. This should be contrasted with VALUES which nullifies those parameters that do not explicitly appear on the statement. To nullify a parameter value with MODVL, specify the value NIL.

If commas are being used as separators, missing positional parameter values are indicated by commas without a preceding value. If spaces are being used as separators, a # sign indicates a missing positional parameter value. Missing parameter values to the right of the last value specified need not be indicated by commas or # signs. See Example 1 below. No action is needed for missing keyword parameters as these are identified by the keyword and not by their position.

MODVL modifies values methodically from the leftmost to the rightmost parameter in the statement.

4.21.5 Optional Parameters

parameter-value-i

Defines the new default value for the string &i (where 1 < i < 99) in an invoked JCL sequence or in a job description.

The value now specified replaces the existing value, if any, which may have been given by a preceding VALUES or MODVL statement. The value supplied must follow the syntax rules for the parameter(s) for which it is defined; however it may be a protected string (that is, any character string enclosed in single quotes, except that a single quote inside a string must be entered twice in succession) of up to 128 characters. The position of a parameter value relative to other parameter values in a MODVL statement defines the value i for the identification of the associated i.

keyword-value-i

Defines the new default value to replace every occurrence of &keywordi in a invoked JCL sequence or in a job description. The value now specified replaces the existing value, if any, which may have been given by a preceding VALUES or MODVL statement. The value defined must follow the syntax rules of the parameter(s) for which it is defined; however it may be a protected string of up to 128 characters. The maximum length of a keyword is 8 characters.

Example 1:

There are 10 positional parameters (&1 through &10) and values p2, p4, p6 are to be given to the 2nd, 4th and 6th respectively (that is, &2,&4,&6). The following 4 MODVL statements are equivalent:

```
MODVL , p2,, p4,, p6,,,,;

MODVL , p2,, p4,, p6;

MODVL # p2 # p4 # p6 # # # #;

MODVL # p2 # p4 # p6;
```

The previously existing values, if any, of &2, &4 and &6 are replaced with p2, p4 and p6 respectively. It does not matter whether &2 had no old value or had an old value supplied by a preceding VALUES or MODVL statement; its value is now p2 and remains so until changed by another MODVL or VALUES statement.

The values of the parameters for which no new values are given are not affected by the MODVL statement. Thus &1, &3, &5, &7, &8, &9 and &10 retain whatever values they had, if any, before the MODVL statement took effect.

Example 2:

Consider the following sequence:

```
VALUES  DK1= 'MD=K116  DVC=MS/M500'
        DK2= 'MD=K200  DVC=MS/M500';

&DK1 refers to MD=K116  DVC=MS/M500.
&DK2 refers to MD=K200  DVC=MS/M500.

MODVL  DK1= 'MD=C200  DVC=MS/M500';

&DK1 refers to MD=C200  DVC=MS/M500.
&DK2 refers to MD=K200  DVC=MS/M500.
```

4.22 OUTVAL

4.22.1 Function

Sets default output handling and output editing values for parameters of SYSOUT and WRITER statements in the current job.

4.22.2 Enclosure

Job.

4.22.3 Format

```
OUTVAL    [CLASS = identifier1]
          [PRIORITY = digit1]

          [WHEN =(JOB    }]
          [      {STEP   }]
          [      {IMMED  }]
          [      {DEFER  }]
          [      {digits5}]

          [{HOLD }]
          [{NHOLD}]

          [NAME =identifier8]

          [{BANNER }]
          [{NBANNER}]

          [            {JOB   }]
          [BANLEVEL =  {OUTPUT}]
          [            {COPY  }]
          [            {MEMBER}]

          [BANINF = (alphanum12 [,alphanum12] ...)]

          [COPIES = digits2]

          [DEVCLASS = device-class   [ MEDIA = volume-name}]

          [DEST = [host-name].station-name]

          [{SLEW }]
          [{NSLEW}]

          [{DELETE}]
          [{NDELETE}];
```

4.22.4 Description of Statement

The OUTVAL statement overrides the system default values which apply to every SYSOUT file of the current job. It cannot appear inside a step enclosure but may be specified between steps; it is valid until the next OUTVAL statement (or until the end of the job if there are no more OUTVAL statements). The overriding works as follows:

1. At job initialization, the default output parameters correspond to the standard system parameters.

2. Each time an OUTVAL statement appears, the specified parameters replace the preceding default values as the new default values. If no parameters are specified (that is OUTVAL;), the standard system parameters replace the preceding default values.

3. Each time a WRITER or SYSOUT statement is encountered, the default values of the previous OUTVAL statement, if one exists, are used; however, any parameter values explicitly stated in the WRITER or SYSOUT statement temporarily override the defaults. (See below).

JOR and JOBOUT

The system defines by default the following output characteristics for the file (known as JOB-REP) that contains the Job Occurrence Report:

```
CLASS=C, PRIORITY=3, WHEN=JOB, COPIES=1
```

However, you can override these values in the first OUTVAL statement for printed output that appears before the first step; in this case, the WHEN parameter, if specified, is ignored and the value JOB still applies. If no OUTVAL appears before the first step, or if the first OUTVAL before the first step does not contain any parameters, the default system values apply. Any such OUTVAL statement affects all following SYSOUT and WRITER statements until the next OUTVAL statement, if any, appears.

The group of standard SYSOUT files that have the same characteristics as JOB-REP, including compiler, linker and step output, is regarded for output purposes as a single file, known as the "JOBOUT". Consequently, the OUTVAL statement, if any, that applies to JOB-REP also determines which standard SYSOUT files will be considered as part of the JOBOUT.

4.22.5 Parameters

For a description of all OUTVAL parameters, see the parameter descriptions for the SYSOUT statement. Note however that station or class dependent parameters are re-evaluated. If the output class, device class or media derive from a destination other than OUTVAL, then these new values override the OUTVAL values. Similarly, if a new class is specified, the priority of this class overrides the priority specified in the OUTVAL statement.

4.23 POOL

4.23.1 Function

Requests the allocation of one or more devices in the same device-class to a pool, for the duration of a step.

4.23.2 Enclosure

Step.

4.23.3 Format

```
            {[{ 1      }*]                                  }
POOL        {[{ digits2} ]   device-class   [MAX = digits2]};
            {                                               }
            {                                               }
            {  device-name                                  }
```

4.23.4 Description of Statement

The function of the POOL statement is to allocate one or more devices of a given device-class to a pool. The pool of devices is reserved for use by a step for the whole of the step's duration.

More than one POOL statement can be used in a step. The effect is to add the devices to the pool corresponding to the appropriate device class. Devices can be requested from a pool by an ASSIGN statement. In this case, one or more POOL statements naming devices with the same attributes may precede the ASSIGN statement in the step enclosure. See the ASSIGN statement, POOL parameter.

The POOL statement is used to minimize, in certain cases, the number of devices that are allocated to a step. This can be best shown by considering an example.

Assume that you wish to process two files that are stored on different volumes. Two ASSIGN statements are required in the step enclosure. From the system viewpoint, devices are allocated on an additive basis. Thus, the assignment of two files on different volumes is taken as a request for the allocation of two devices.

It is possible, however, that you wish both files to be processed sequentially in time, that is, the second volume may be mounted on the same device when processing of the first file is completed.

You know then, that one device is sufficient, but cannot, without POOL, indicate this to the system. For an installation with a limited number of available devices, this presents a problem which is made worse as more ASSIGN statements that request different volumes are specified.

To overcome this problem, you specify in both ASSIGN statements that you wish to use a pool device and indicates which file (and therefore which volume) is to be processed first.

At step execution, when processing of the first file is ended, you issue an order to open the second file. This results in a request to mount the second volume on the same (pool) device, which is still reserved for use by the current step.

Comments and Restrictions

The attributes of the device-class specified in the ASSIGN statements corresponding to the POOL statement must be identical to those of the device-class named in POOL.

If the requested number of devices in a device class are not immediately available, the step must wait until the POOL statement can be satisfied.

Once the step has started, the requested number of devices is guaranteed to the step; however, if another step has assigned a file contained on a volume that is mounted on one of the pooled devices of this step, the other step will be able to access that file (subject to any file sharing restrictions that apply to the file).

4.23.5 Parameters

Pool devices are described by specifying either the device class or device name.

4.23.5.1 Mandatory Parameters

device-class

Specifies the device class and its attributes for the device being pooled (see Section 3). This form of identification is recommended in preference to the "device-name" format.

device-name

The format is given in Section 3. When specified, no other parameter is coded and only the one named device is allocated. Thus, if more than one pool device is requested using device-names, it is necessary to specify one POOL statement for each such device. The effect is that the devices are added into the same pool.

4.23.5.2 Optional Parameters

The first optional parameter is used when a specified minimum number of devices of a given class must be allocated to the pool. The default value is 1 and the maximum value is 99. The asterisk (*) must immediately follow the specified value. The step will not be initiated until the minimum number of devices is available and these resources have been allocated to the step.

MAX

Specifies the maximum number of devices to be allocated, if available.

NOTE: The difference MAX-minimum number of devices represents a conditional request for devices.

Example:

```
POOL 2*MT/T9/D1600
```

Requests two 9 track tape devices for tape density 1600 bits per inch (b.p.i.).

```
POOL MS/M500
```

Requests one MSU0555 disk device.

```
POOL MT/T9/D1600, MAX=2;
```

The step can be initiated only if at least one 9 track tape drive is free: another 9 track tape drive is conditionally requested.

```
POOL 0*MS/M452, MAX=1
```

A MSU0452 disk drive is conditionally requested. In this case, the step may be initiated even if no MSU0452 disk drive is available.

```
POOL MS02
```

Requests one disk device whose identifier is MS02.

4.24 PREFIX

4.24.1 Function

Defines a prefix to be placed before every external file name that begins with a dot ".".

4.24.2 Enclosure

Job.

4.24.3 Format

```
PREFIX   {prefix-name }
         {project-name};
```

4.24.4 Description of Statement

The PREFIX statement is used to override the default prefix value for external file names. The default prefix-name is the name of the project with which the job is associated; it is either specified in the PROJECT parameter of the $JOB statement or taken from the site catalog.

Each PREFIX statement that appears in a job description overrides the previous one.

See the *Catalog Management Manual* for a description of the use of prefixes.

Comments and Restriction

PREFIX does not apply to external file names in protected strings (that is, enclosed in quotes).

The new external file name, composed of the prefix-name followed by the old external file (which starts with a period) must follow the rules for external file name (see Section 3).

The prefix-name may contain several component names (see Section 3) and period separators. This allows you to specify automatic prefixing for several levels of the catalog hierarchy (see Example 2 below).

4.24.5 Optional Parameter

prefix-name

The value given for prefix-name will be prefixed to every occurrence of an external file name that starts with a period and is not in a protected string.

The maximum permissible length of prefix-name is 42 characters. If prefix-name is not specified, the prefix is set to the project name.

Example 1:

Consider the following sequence of statements:

```
PREFIX SALES;
STEP......;
ASSIGN UPD, .ACCTS;
ENDSTEP;
```

The effect of the PREFIX is to replace .ACCTS by SALES.ACCTS.

Example 2:

```
PREFIX AREA1.CUSTOMER;
STEP ST1,....;
ASSIGN IFN1, .ACCTS;
ASSIGN IFN2, .ADDRESSES;
ENDSTEP;
```

In the above example, the two ASSIGN statements become :

```
ASSIGN IFN1, AREA1.CUSTOMER.ACCTS;
ASSIGN IFN2, AREA1.CUSTOMER.ADDRESSES;
```

4.25 QASSIGN

4.25.1 Function

Establishes the correspondence between the symbolic and external queue names and specifies a queue structure, message format and the method of scanning. Also allocates an MCS queue to the job step.

4.25.2 Enclosure

Step.

4.25.3 Format

```
QASSIGN       symbolic-queue-name [.symbolic-subqueue1-name
                                  [.symbolic-subqueue2-name
                                  [.symbolic-subqueue3-name]]]

              external-queue-name1

              [SITE = node-name]

     [ {IN                                        } ]
     [ {INOUT            ACCESS =    {LIN}    } ];
     [ {                             { RR}    } ]
     [ {OUT     [REPLY = external-queue2-name]} ]
```

4.25.4 Description of Statement

The QASSIGN statement is used to establish the correspondence between the symbolic (internal) queue name, as specified in the CD area (Communication Description) of the MCS COBOL program, (or in the H_CDOUT or H_CDIN of a GPL application) and the external queue name defined in the network description.

A QASSIGN statement is required for each input queue used in a step. A symbolic input queue can have one or more associated subqueues. Each subqueue must be identified by a QASSIGN statement which names the symbolic queue and the symbolic subqueue.

A QASSIGN statement is not required for a symbolic output queue (a symbolic output queue can be implicitly identified by the terminal name).

Up to 26 QASSIGN statements may be specified in a step enclosure. The first QASSIGN statement for a given queue defines the subqueue method of scanning for all subqueues related to the queue. In this case, the ACCESS parameter, if present on following QASSIGN statements related to the same queue, is ignored. A symbolic-subqueue-name must be unique within a step enclosure.

Where disk queues are used, no ASSIGN statement is required because the file is opened at system level and is therefore shareable by all process groups.

The connection mode between correspondents and input and output message format can be specified. Automatic insertion of end of line characters (that is carriage return, line feed) may also be requested.

Refer to the Communications Processing Facility, and *MCS User's Guide* for more details on the use of the QASSIGN statement.

4.25.5 Parameters

4.25.5.1 Mandatory Parameters

symbolic-queue-name

Identifies the queue name as follows:

- in MCS COBOL, in data-name-1 of the CD area (input and output)
- in GPL, by QUEUE_NAME of H_CDOUT or H_CDIN

The symbolic-queue-name consists of up to 12 alpahanumeric, hyphen and underscore characters.

external-queue1-name

Identifies the queue name as specified in the Network generation. The external-queue-name consists of up to 12 alphanumeric, hyphen and underscore characters.

4.25.5.2 Optional Parameters

symbolic-subqueue-name

Identifies a subqueue associated with the named queue; it consists of up to 12 alphanumeric, hyphen and underscore characters. This parameter does not apply if OUT or INOUT is specified. It is applicable only for input queues for partitioning the logical queue as follows:

- in MCS COBOL in data-name-2, data-name-3 and data-name-4, of the input CD area;
- in GPL, by SUBQUEUE_NAME, SUBQUEUE2_NAME and SUBQUEUE3_NAME of H_CDIN.

SITE

Identifies a DSA system which can either be 'lsys' or 'rsys' for DPS 7000.

IN

Specifies that the named queue is an input queue. Applicable only to program queues. This is the default.

INOUT

Processing is both input and output mode. A subqueue cannot be specified in INOUT mode. Applicable only to program queues.

OUT

Specifies that the named queue is an output queue. A subqueue cannot be specified when OUT is used.

ACCESS

Specifies the method of scanning subqueues on a receive action. ACCESS does not apply if OUT is specified.

LIN

Each receive action initiates a scanning of the subqueues, beginning with the first subqueue. This is the default value for ACCESS.

RR

A "round robin" search is performed. The search resumes at the subqueue immediately following the subqueue that provided data on the last receive.

REPLY

Specifies the external-queue-name of the queue where messages issued within the current step are sent (applies to INOUT and OUT parameters only). For COBOL, this queue name must appear in the SYMBOLIC SOURCE field of the receiver's INPUT CD (see the *COBOL 85 Language Reference Manual*, Communication Section). For GPL, the queue name must be defined in the H_CDIN. Applicable only to application-to-application communication.

Notes on Line Length and the Number of Blocks

These are network generation options. The number of blocks specifies the number of logical lines to be inserted into a message sent to the terminal queue by the application. The line length specifies the length of the logical line.

4.26 RELEASE

4.26.1 Function

Removes a job from HOLD status and makes it available for scheduling.

4.26.2 Enclosure

Job.

4.26.3 Format

```
RELEASE    job-name  [{SWITCHES = {hexa8}               }]
                     [{           {PASS }               }]
                     [{                                 }]
                     [{SWITCHi = {0} [SWITCHj = {0}].. .}];
                     [{          {1}            {1}     }]
```

4.26.4 Description of Statement

The RELEASE statement is interpreted at job execution time as releasing a job which was previously put into HOLD status. If two or more jobs have the specified job-name and the relevant job attributes (see the Restriction below), all those jobs are released when the RELEASE is executed.

Where the job in HOLD state has a HOLD count associated with it, the effect of RELEASE is to reduce this count by 1. The job in HOLD status is not actually released until this count becomes 0.

Severity codes returned by RELEASE are:

SEV	**Meaning**
0	RELEASE executed successfully.
1	The job (or jobs) to be released was not in HOLD status and RELEASE is ignored.
2	No job in the system had the specified job-name and the relevant job attributes (see the Restriction below).

Restriction

The job to be released must have the same attributes (user-name, project-name) as the job which contains the RELEASE statement.

4.26.5 Parameters

4.26.5.1 Mandatory Parameter

Job-name

Identifies the job to be released. See $JOB for a full description of this parameter.

4.26.5.2 Optional Parameters

SWITCHES

Specifies the initial value of all system switches in the released job.

The hexa8 represents a value of up to eight hexadecimal characters to which the switches are set. If less than 8 characters are given, zeros are added on the right to make a total of 8 characters.

PASS

Indicates that the values of the switches of the released job are to be set identical to those of the releasing job at the time that the RELEASE is executed.

SWITCHi

The specific switches listed are set to their associated values. Other switches are left as they were on the HOLD job before RELEASE was executed.

4.27 REPORT

4.27.1 Function

Prints a message in the JOR (Job Occurrence Report) at execution time.

4.27.2 Enclosure

Job or Step.

4.27.3 Format

```
REPORT   'string110';
```

4.27.4 Description of Statement

The REPORT statement allows you to place a message in the execution part of the JOR at execution time. The message can be up to 110 characters long.

REPORT should be used instead of SEND wherever possible, in order to avoid contacting the operator unnecessarily.

REPORT can be used, in conjunction with JUMP, to provide in the JOR a trace of the execution of a particular job description or to summarize what has happened during the execution of a job.

4.28 RUN

4.28.1 Function

Requests the execution of a job that is independent from the job issuing the RUN statement.

4.28.2 Enclosure

Job.

4.28.3 Format

```
RUN

    {[INFILE =] (sequential-input-file-description)       }
    {                    [INDEF = (define-parameters}]    }
    {                                                     }
    {  {member-name}                                      }
    {  {           } [INLIB =) (input-library-description)}
    {  {star-name  }                                      }

     [INFILE2 = (sequential-input-file-description)
                          [INDEF2  = (define-parameters)]]

     [INFILE3 = (sequential-input-file-description)
                          [INDEF3  = (define-parameters)]]

     [INFILE4 = (sequential-input-file-description)
                          [INDEF4  = (define-parameters)]]

     [CLASS = identifier2]

     [HOLD]

     [HOLDOUT]

     [PRIORITY = digit1]

     [JOBS = ([job-name1]   [job-name2])]

     [VALUES = ([parameter-value1] [parameter-value2]...]
               [keyword1=keyword-value1
               [keyword2=keyword-value2]...])]
```

```
     [{SWITCHES = hexa8                    }]
     [{           PASS                     }]
     [{                                    }]
     [{SWITCHi=  {0}  [SWITCHj=  {0}  ]...}]
     [{          {1}              {1}      }]

[DELETE]

[SIZEOPT = (size-parameters)]

[STEPOPT = (step-parameters)];
```

4.28.4 Description of Statement

The RUN statement requests the introduction into the system of a separate job stream from that in which the statement occurs (that is, a "spawned" job stream). The RUN statement and the EJR operator command are identical in function. The structure of the disk library member or sequential input file named in the RUN statement must be that of an input job stream. The effect is as if the operator had initiated the Stream Reader to read the specified job stream.

The data format may be DATA or DATASSF. Non-native files may be used if their file characteristics are given by the INDEF parameter group. In all cases, file characteristics must be compatible with Stream Reader rules (for example, record size must not exceed 255 characters). Secondary input may come from files defined by INFILE2, INFILE3 and INFILE4 if a $SWINPUT statement references them.

The JOBS parameter allows you to select a particular job or sequence of jobs from the job stream in the named files. If JOBS is omitted, all the jobs in the stream are selected for the input stream. Note that if Access Rights have been implemented, only an operator project can run a job whose user-name is different from that of the calling job.

If CLASS, PRIORITY, HOLD OR HOLDOUT are specified in the RUN, the stated parameter values override those in the $JOB statements of the selected jobs.

If only one job is submitted, the $JOB and $ENDJOB statements are not needed. The default value for the job-name is the member-name (RUN member-name) or external-file-name (RUN efn). If the length exceeds 8 characters, only the first 8 characters are used.

If the VALUES parameter is used, the spawned jobs are updated with the specified values corresponding to the positional parameter-values or to the associated keyword-values.

If a warm restart is performed as a result of an incident which occurred during the execution of RUN step (RUN produces its own job step), processing of the input stream will resume from the job immediately following the last job that was successfully introduced into the system.

NOTE: The RUN statement defines an entire step, but is considered as part of the basic operating system and is treated in this manual like a Basic JCL Statement.

4.28.5 Parameters

4.28.5.1 Mandatory Parameters

The only mandatory parameter group is the one which identifies the file that contains the job stream. This may be specified as a sequential input file (optionally preceded by INFILE=) or as member (or members) of an input library (optionally preceded by INLIB=). The parameters of a sequential-input-file-description and of an input-library-description are listed in Section 3.

In the case of an input library the member (that is, subfile) may be identified by its member-name. The member-name must not exceed 31 alphanumeric, hyphen and underscore characters and must begin with a letter or a digit.

Alternatively the member (or members) of a library file may be identified by the star-name (see the *Library Maintenance Reference Manual* for an explanation of this). Essentially this convention permits the selection of one or more library file members according to whether their names match a specified pattern; as a special case, all members may be selected. The members selected are included in their name order (using EBCDIC collating sequence). The job description(s) in each member must be complete. The value SYS for INLIB identifies the system library SYS.HSLLIB.

4.28.5.2 Optional Parameters

INDEF

May be used to associate define-parameters (see Section 3) with the sequential-input-file in which the job stream is stored.

INFILEi

Specify secondary input files (2<i<4) which are referenced by $SWINPUT statements. The contents of these files are included in the job stream at the position where their respective $SWINPUT statements occur. Specifying these on the RUN statement ensures resource availability at execution time.

INDEFi

Specify the define-parameters to be associated with the file referenced by the corresponding INFILEi (where 2<i<4).

CLASS

Specifies the job class of the spawned jobs (see $JOB for more information). The specified CLASS overrides the CLASS contained on the $JOB statements in the spawned job stream.

HOLD

Places the selected jobs in "hold" status (see HOLD parameter in the $JOB statement).

HOLDOUT

Holds back the output files produced by the selected jobs until the operator releases them (RO command).

PRIORITY

Specifies a scheduling priority between 0, the highest priority, and 7, the lowest priority. The selected jobs are scheduled according to this priority which overrides the priorities, if any, specified on their $JOB statements.

Within a specified priority, jobs are scheduled on a first-in-first-out (FIFO) basis.

NOTE: Even a high priority job will not be scheduled if the maximum number of jobs in the same class is already scheduled.

JOBS

Allows you to select a sequence of jobs from the stored job stream. Job-name1 gives the name of the first job in the sequence and job-name2 gives the name of the last job. Thus all jobs from job-name1 to job-name2 inclusive are selected from the input file. JOBS should not be used if several members of a library file are being selected using the star-name convention.

If job-name1 is omitted, the sequence starts with the first job contained in the input file. If job-name2 is omitted, the sequence finishes with the last job contained in the input file. If the JOBS parameter is omitted, all jobs contained in the input file are selected for the input stream.

NOTE: JOBS=(JBA, JBA) may be written as JOBS=JBA. This specifies that only one job, named JBA, will be selected.

The above explanation can be illustrated as follows:

no JOBS parameter	every job is selected.
JOBS=X or JOBS=(X,X) or JOBS=(X X)	the job X is selected.
JOBS=(,X)	every job up to job X (inclusive) is selected.
JOBS=(X,)	every job starting from job X (inclusive) is selected.
JOBS=(X,Y) or JOBS=(X Y)	every job starting from job X up to job Y (both inclusive) is selected

VALUES

Specifies the parameter values that will be passed to the JCL sequence. See the INVOKE statement for a description of VALUES, and the *JCL User's Guide* for an explanation of parameter substitution.

SWITCHES

Specifies the initial value of all system switches in the selected jobs. The hexa8 represents a value of eight hexadecimal characters to which the 32 switches are initialized, corresponding, from left to right, to switches SW0 to SW31. For example, SWITCHES=40000001 will set switches SW1 and SW31 to the value 1. If less than 8 characters are given then zeros are added on the right to arrive at a total of 8 characters.

PASS

If this value is specified for SWITCHES, the values of the switches in every selected job are set according to the values in the calling job (that is, the job which contains the RUN statement) at the time the RUN step is executed.

SWITCHi

The switches listed are set to the values specified. Those not listed have an initial value of 0 (contrast with the rule for SWITCHi in the RELEASE statement).

DELETE

Applies only if the input comes from a source library. It requests deletion of the member after it has been processed by the Stream Reader. If the star-name convention has been used to select members, then they are deleted after the last one has been processed by the Stream Reader. If a member cannot be properly processed (for example, due to a permanent I/O error) then deletion does not take place.

SIZEOPT

Allows you to specify size-parameters. See Section 3 and the SIZE statement.

STEPOPT

Allows you to specify parameters to control the execution of the RUN step-defining statement. Any of the STEP parameters except DUMP=ALL, DEBUG and the OPTIONS parameter can be specified for the parameter group step-parameters.

Examples 1:

```
RUN DMJOB16, SYS;
```

This introduces the job stream contained in member DMJOB16 of the system library SYS.HSLLIB.

Example 2:

If the RUN statement is

```
RUN WJCIII, JSTREAM.TEST1, VALUES=(NEW.TFILE, FLSTAT=NIL),
CLASS=D;
```

and one of the requested jobs contains:

```
$JOB JOB1, USER=BIG, PROJECT=YIN;
   .       .     .      .
   .       .     .      .
   ASSIGN TFILE, &1, &FLSTAT;
```

the result will be:

```
$JOB JOB1, USER=BIG, PROJECT=YIN, CLASS=D;
   .       .     .      .
   .       .     .      .
   ASSIGN TFILE, NEW.TFILE;
```

Example 3:

```
RUN WELL1E, MST.STRS, JOBS=(,JBX3);
```

This will select from the job stream in subfile WELL1E of library MST.STRS, a sequence of jobs starting from the first one up to the job named JBX3.

Example 4:

```
RUN QDLY, ACCT.PROC, JOBS=(NAT1,), VALUES=(TAPE=X393);
```

All jobs from NAT1 to the last job in the specified job stream will be introduced to the system, once every occurrence of &TAPE in these jobs has been replaced by the string X393.

4.29 SEND

4.29.1 Function

Sends a message text from the user to the operator.

4.29.2 Enclosure

Job or Step.

4.29.3 Format

```
SEND          'string105'

            [{user-name}]
            [{MAIN     }]
```

4.29.4 Description of Statement

The SEND statement is normally used to inform the operator of certain processing requirements, such as the need for more sort work tapes, or in conjunction with the results of conditional JCL statement (JUMP, LET) execution. When you wish to note the results of JCL statement execution when the operator need not be involved at the time, the REPORT statement should be used instead.

The operator concerned is the operator of the console associated with the job issuing the SEND unless either user-name is specified or a CONSOLE statement has appeared previously in the job description. Note that the SEND statement operates at job execution time whereas the MESSAGE statement operates at JCL translation time.

4.29.5 Parameters

4.29.5.1 Mandatory Parameter

'string105'

The message is a string of up to 105 characters which are printable on the operator console. The Command Interpreter adds the run occurrence number before the message when it is sent to the operator.

4.29.5.2 Optional Parameters

user-name

Identifies a user (and therefore a terminal) that is logged to the system. If this parameter value is omitted, the message will be displayed on the operator console associated with the issuing job by default, or, if a CONSOLE statement has been declared previously in the job, the message will be displayed on the console identified by the CONSOLE statement. The user-name parameter allows you to override either of those values.

MAIN

The use of this parameter allows you to override a previous CONSOLE statement to specify that the current message will be sent to the main operator console.

4.30 SIZE

4.30.1 Function

Declares the size of a working set required for the execution of a given job step.

4.30.2 Enclosure

Step.

4.30.3 Format

```
SIZE     [declared-working-set]
         [CHPPAGE = digits]
         [NBBUF = digits4]
         [POOLSIZE = digits];
```

4.30.4 Description of Statement

The SIZE statement is used to declare the size of the working set, in units of 1024 (1K) bytes. The working set is the amount of memory required for code and data segments and process control structures. Included in these segments are data management buffers used for Input/Output operations, which are declared when the file is opened. Therefore the SIZE statement includes segments resident in memory permanently during execution (that is, process control structures), and segments which are loaded from the backing store as and when required during execution (that is, code and data segments).

When the system can provide the declared-working-set value stated in SIZE, as well as other resources (e.g., devices, volumes, files) step execution can be initiated. If the system cannot fulfil the memory requirements at step initiation, the step is "WAITING FOR MEMORY" in the same way as it may be waiting for a device or volumes (see the *JCL User's Guide*). Once step execution is proceeding, the amount of memory available to a step can vary from the stated declared-working-set value, depending on the instantaneous load and the requirements of the executing step, except when the MAXMEM or MINMEM options of STEP are used (see STEP).

You can make an initial estimate of the memory requirements for a particular step by referring to the value tables in the *System Administrator's Manual*.

During optimization of the declared-working-set value, the linker listing can be used to obtain values for process control structures and the number and size of code and data segment. The Job Occurrence Report gives information relating to the memory requirements of the buffers. In the cases where a reasonable estimate of the declared-working-set value cannot be made, the MAXMEM option of the STEP statement can be used in conjunction with the SIZE statement to generate a graph of execution efficiency (as an inverse function of the number of missing segments stated in the JOR), against the declare-working-set value. The MAXMEM option informs the system that the executing step is under going performance measurements and, whatever its load, it is never allowed more memory than the one stated in the SIZE statement. If no value or a value less than 200K is declared, size is forced to 200 KBytes.

The channel program page size default value can be overwritten in cases where a channel program overflows the 4 pages available to it.

The default size of the total UFAS buffer pool can be overwritten using the POOLSIZE parameter of SIZE: the default value may be inadequate when more than one file is being accessed by a step (see the *UFAS-EXTENDED User's Guide*).

4.30.5 Optional Parameters

declared-working-set

Specifies the amount of memory required in units of 1 KBytes. It includes the value specified, or applying by default, for POOLSIZE. The default value is 200 KBytes.

CHPPAGE

Gives the size of a channel program page in bytes. The default value is 4096 bytes (4 KBytes).

NBBUF

Specifies the number of buffers which are required for the non-swappable memory (program control structures). The number of buffers must not exceed 4,000. The default value is 50.

POOLSIZE

Specifies the maximum size of the UFAS buffer pool in units of 1 KBytes. It is included in the value specified (or applying by default) for the declared-working-set. It must not exceed the declared-working-set value. The default value is 15 KBytes less than the value of SIZE.

4.31 STEP

4.31.1 Function

Opens a step enclosure and names a load module to be executed.

4.31.2 Enclosure

Job.

4.31.3 Format

```
STEP

     load-module-name  (input-library-description)

       [XPRTY = digit1]

       [CPTIME ={9999999}]
       [        {digits7}]

       [ELAPTIME ={9999   }]
       [          {digits4}]

       [LINES ={99999999}]
       [       {digits8 }]

       [DUMP ={NQ             }]
       [      {DATA  [PRIVATE]}]
       [      {ALL   [PRIVATE]}]

       [DEBUG [= (sequential-input-file-description)]]

       [OPTIONS = 'string4096']

       [REPEAT]

       [{MAXMEM}]
       [{MINMEM}];
```

4.31.4 Description of Statement

The STEP statement must be the first statement of a step enclosure. Its function is to request the loading of the load module whose execution constitutes the job step. The load module is a member of a load module library which is either a temporary library created by the system or a permanent cataloged or uncataloged user library.

Statements that follow the STEP statement (and constitute the step enclosure) request the resources that are required for step execution. The STEP statement must be matched with a corresponding ENDSTEP statement, which must be the last statement of the step enclosure.

4.31.5 Parameters

4.31.5.1 Mandatory Parameters

load-module-name

Specifies the required load module for step execution. Its length must not exceed 31 alphanumeric, hyphen and underscore characters and must begin with a letter or digit. This is the first positional parameter of STEP.

input-library-description

Identifies the required load module library. The expansion of input-library-description is given in Section 3. This is the second positional parameter of STEP.

For the standard temporary library, TEMP. LMLIB, the parameter TEMP is specified.

SYS specifies the system library, SYS.HLMLIB.

4.31.5.2 Optional Parameter

XPRTY

Specifies a step execution (dispatching) priority between 0, the highest priority, and 9, the lowest priority. The default value for XPRTY is determined by the CLASS of the job containing the step. The execution priority of a step can be modified by the operator.

NOTE: The system uses execution priorities between 0 and 3. Therefore, for user jobs, the recommended value for execution-priority is between 4 and 9, inclusive.

CPTIME

Specifies a maximum permitted CPU usage time for the step, in units of one-thousandths of a minute. If the specified value is exceeded, the job is abnormally terminated (SEV value 3) and the return code TIMEOUT is produced.

The default value (9999999) means that there is effectively no limit on CPU usage time.

ELAPTIME

Specifies a maximum permitted clock-time during which the step can execute. The unit of time is one minute. The default value (9999) means that there is no limit on the elapsed time for step execution. If the specified value is exceeded, the job is abnormally terminated (SEV value 3) and the return code TIMEOUT is produced.

LINES

Specifies the maximum number of printer records that may be written to each SYSOUT file by the step. If the limit is exceeded, the job is abnormally terminated and the return code ERLMOV is produced. The default value (99999999) means that there is effectively no limit.

NOTE: If several copies of a SYSOUT file are produced (see the COPIES parameter of SYSOUT, WRITER, OUTVAL), only the first copy is taken into account for the LINES parameter.

DUMP

Specifies if a dump listing is to be produced in the event of an abnormal step termination, and if so, what its contents will be.

NO	By default, no dump listing is produced.
DATA	Only segments containing data are dumped.
ALL	All segments of the step are dumped.
PRIVATE	Only those segments private to the process (type 3) are dumped.

DEBUG

Specifies that step execution is in debug mode. The default is that step execution is in normal mode. The Program Checkout Facility (PCF) commands are retrieved from the sequential input file specified, in a batch environment. In an IOF environment these commands come from your console.

OPTIONS

Specifies a string of up to 4096 characters to be read by a user program during load module execution. Refer to the appropriate User's Guide (for example, FORTRAN) for details about the use of OPTIONS.

REPEAT

Specifies that the step is to use the Checkpoint/Restart facilities. This provides the step with a recovery point (either the beginning of the step itself or a checkpoint) at which it can possibly be restarted after a system crash or abnormal termination of a step.

If a file is specified by EFN*, a step with the REPEAT option cannot be repeated from a Checkpoint after a system crash or abnormal termination of a step.

If REPEAT is not specified, by default the step has no recovery points and cannot be restarted. In this case, calls to checkpoints are treated as dummy statements. However, the entire job can be repeated (see the REPEAT option on $JOB).

MAXMEM

If this self-identifying-parameter appears, throughout the execution of the step the system will limit to the value of the declared-working-set (applying by default or specified in SIZE) the memory that is available to the step. The declared-working-set is reserved for the step but, in addition, the total amount of memory available to the step at any instant cannot exceed the declared-working-set value. This parameter should only be used when the performance of the step is being evaluated in order to find an optimum declared-working-set value (see *JCL User's Guide*).

MINMEM

MINMEM applies only to telecommunications jobs. If this self-identifying parameter appears, throughout the execution of the step the system will guarantee for the step the value of the declared-working-set.

Examples:

```
STEP MJPROG, TEMP;

STEP DMPROG, (NEW.DMLMLIB, DEVCLASS=MS/M500, MEDIA=B016);

STEP COBSTEP, NLIB.TEST, DUMP=DATA, REPEAT;
```

The first example names the load module MJPROG for execution. The module is to be found in the standard temporary library TEMP.LMLIB.

The second example refers to the load module DMPROG. This module is to be found in the library NEW.DMLMLIB. This library is not cataloged and resides on the MSU0555 disk volume B016.

The third example refers to the load module COBSTEP which is a member of the library file NLIB.TEST. If the step terminates abnormally the data segments are dumped. If a system crash occurs, the step will be repeated from the beginning of the step (or from the last checkpoint if any has been taken).

4.32 $SWINPUT

4.32.1 Function

Switches input from the current input stream (or file) to another file.

4.32.2 Enclosure

Job, step, input-enclosure, data enclosure.

4.32.3 Format

```
$SWINPUT {[INFILE]=  (sequential-input-file-description)       }
         {member-name [INLIB =] (input-library-description)    }
         {                                                     }
         {             {'string105'  [ANSWERS = ('string105' } }
         {             {             ['string105']...)]      } }
         {CONSOLE =    {                                     } }
         {             {('string105'  ['string105'] )        } }
         {             {              END = 'string8'        } }
         {                                                     };
```

4.32.4 Description of Statement

The $ sign is mandatory and it must be in the first position of the record.

This statement is recognized and processed by the Stream Reader. The effect of a $SWINPUT statement is to cause a switch in the input stream from the current stream (or file) to the file named in the $SWINPUT statement. Logically the effect may also be considered as equivalent to the removal of the $SWINPUT statements from the stream by the Stream Reader and the replacement of each by the contents of the file which it references.

Temporary files and permanent files cataloged in private catalogs must not be used unless the input stream is submitted using a RUN statement. A $SWINPUT statement can appear anywhere within a job-enclosure, an input-enclosure or a data-enclosure; it may be embedded in a JCL statement but it must not be embedded in a Stream Reader statement ($JOB, $ENDJOB, $INPUT, $ENDINPUT, $DATA, $ENDDATA, $SWINPUT).

The file referred to by $SWINPUT must not contain a $ENDJOB or a $ENDDATA statement.

If a $SWINPUT statement appears within an input-enclosure or data-enclosure the parameter CKSTAT must be specified on the $INPUT or $DATA statement. The end of file condition on the file to which the input has been switched causes a return to the stream containing the $SWINPUT statement.

Instead of referencing a file, a $SWINPUT statement may refer to CONSOLE input. If the job has been generated by another job using a RUN statement, the console concerned is that of the spawning job submitter. Otherwise the console is that of the submitter, that is, main operator, RBF operator or IOF user, the console is taken as the main operator.

Any number of $SWINPUT statements may appear in a stream, but the level of nesting cannot exceed 10. The files referenced by a $SWINPUT statement may be assigned using the INFILE 2, INFILE 3 and INFILE 4 parameters of the RUN statement.

The $SWINPUT statement is useful in an IOF environment.

4.32.5 Mandatory Parameters

The only mandatory parameters are those which specify the file to which the input stream is to be switched. This may be done in one of the following ways, a sequential input file, a member of an input library, a CONSOLE or a DISKETTE. The meaning of each is as follows.

sequential-input-file-description

This may be prefixed by INFILE=. It references a sequential file whose contents will be included in the input stream in place of the $SWINPUT statement. This parameter group is listed in Section 3.

input-library-description

This may optionally be prefixed by INLIB=. It references an input-library from which a member (specified by member-name) will be selected. The contents of this member will replace the $SWINPUT statement in the input stream. This parameter group is listed in Section 3.

CONSOLE

Specifies that the switched input is to come from a console. The way in which the console concerned is selected is described in the STATEMENT DESCRIPTION above.

The console input may be supplied in either of the following two ways:

(1) ANSWERS option

This functions as follows: the 'string 105' preceding ANSWERS specifies the prompt message that is sent to the console to solicit the desired input. The string following ANSWERS defines the possible valid inputs. The console operator is given three attempts to give one of these valid replies before the job is aborted. A valid reply is inserted in the input stream in place of the $SWINPUT statement. The first valid reply constitutes the only switched input in this case. If ANSWERS and the valid replies are not specified, any reply up to one record length is accepted.

(2) END option

In this case several lines of input are expected from the console. The leftmost string is the initial prompt message sent to the console to solicit the first line of input. The next string is the repeat prompt message which is sent to the console to solicit the second and each subsequent line of input. If the second string is missing, it defaults to the first string. Both of these are positional parameters. The string specified with END= is the terminator line which indicates the end of the input. The data supplied by the operator, as replies to the prompts, is not restricted to being one of a set of valid responses, as it is in the case of ANSWERS. The replies given before the terminator line form the switched input data which is inserted in place of the $SWINPUT statement in the original stream.

4.33 SYSOUT

4.33.1 Function

Requests the use of the SYSOUT mechanism for a file which is to be written in the current step; notifies the Output Writer mechanism for the printing the file, known as a SYSOUT file. In addition, SYSOUT can be used to override certain standard options provided by the SYSOUT mechanism for the handling of the SYSOUT file.

NOTE: A permanent sequential or library file which is associated with the specified internal-file-name is called a permanent SYSOUT file. If no ASSIGN statement is associated in the current step with the internal-file-name, the system uses a standard SYSOUT subfile.

4.33.2 Enclosure

Step.

4.33.3 Format

```
SYSOUT      internal-file-name

          [CLASS = identifier1]

          [PRIORITY = digit1]

          [WHEN ={JOB    }]
          [      {STEP   }]
          [      {IMMED  }]
          [      {DEFER  }]
          [      {digits5}]

          [HOLD           ]
          [NHOLD          ]

          [NAME = identifier8]

          [{BANNER }]
          [{NBANNER}]

          [            {JOB   }]
          [            {OUTPUT}]
          [BANLEVEL =  {COPY  }]
          [            {MEMBER}]

          [BANINF = (alphanum12 [,alphanum12] ...)]

          [COPIES = digits3]

          [DEVCLASS = device-class  [MEDIA = volume-name]]
```

```
[DEST = [host-name].station.name]

[{SLEW }]
[{NSLEW}]

[{DELETE }]
[{NDELETE}];
```

4.33.4 Description of Statement

The SYSOUT statement activates the SYSOUT mechanism for writing records produced by your program; its secondary function is to inform the Output Writer how to handle the SYSOUT file, by overriding certain default output parameters defined within the system for SYSOUT files, for example:

- identification of the output;
- certain output delivery characteristics;
- the time of the notification of the Output Writer.

Standard SYSOUT Files

If no ASSIGN statement is associated in the current step with the specified internal-file-name, a temporary subfile of the standard SYSOUT file is automatically assigned by the system. This subfile is erased after its contents have been printed. The WHEN parameter can be used to specify when the Output Writer will be requested to output the file.

If a COBOL program contains the suffix-SYSOUT in the SELECT ... ASSIGN clause (see *COBOL 85 Language Reference Manual*, section "File Control"), the SYSOUT statement can be omitted unless parameter are to be overridden. The SYSOUT mechanism will be called automatically, the file name given in the program will be assigned to the standard SYSOUT file, and the records written to the file will be printed by the Output Writer, which will be notified by default at the end of job execution.

Permanent SYSOUT Files

The SYSOUT statement requests the SYSOUT mechanism to write the records to the permanent file associated with the specified internal-file-name by means of an ASSIGN statement in the current step. The Output Writer will be notified at a time governed by the value of the WHEN parameter to print the contents of the file - but see below for the case where WHEN=DEFER is specified.

If no SYSOUT statement is given for the file but a COBOL program contains the suffix-SYSOUT in the SELECT ... ASSIGN clause, the SYSOUT mechanism will be used to write records to the file, but the file contents will not be printed; this also applies if a SYSOUT statement with the parameter WHEN=DEFER appears. In both cases, to print the file, a WRITER statement must be included at a later stage, normally in another job.

For further information on the SYSOUT and Output Writer mechanisms, refer to the *JCL User's Guide*.

Comments and Restrictions

The SYSOUT statement cannot be used to refer to the subfile that contains the Job Occurrence Report. The OUTVAL statement, described earlier, can be used instead.

The DEFINE statement can be used to override output edition parameters, for example, margin setting or form height definition. For permanent SYSOUT files written using the SYSOUT mechanism (see the comment below), all editing parameters specified by DEFINE or applying by default are incorporated into the structure of the file; this means that the editing parameter values will automatically apply when the file is output at a later stage by WRITER, unless at that time the FPARAM parameter is specified (see WRITER).

When the system allocates space for a standard SYSOUT file, a record size of greater than 600 bytes is established. This action is an efficiency requirement of the SYSOUT mechanism. Correspondingly, when you reserve space for a permanent SYSOUT file, the PREALLOC utility should specify a record size of greater than 600 bytes with RECFORM=VB in order to use the SYSOUT mechanism; for SYSOUT files only, this specification is independent of the record size declared in your program. If the space for a permanent SYSOUT file is allocated using the ALLOCATE statement, the DEFINE statement can be used to declare the desired record size. On the other hand, if you wish to write a file with a record size of less than 600 bytes, the SYSOUT mechanism cannot be used to edit the file; in this case, the effect of the SYSOUT statement is to output the file according to the output and editing parameters that apply (by default, and/or in SYSOUT, and/or in DEFINE), but if you use WRITER for the file in the future, you must respecify those parameters. For a file with a record size of less than 600 bytes, the use of SYSOUT with WHEN=DEFER is pointless.

4.33.5 Parameters

4.33.5.1 Mandatory Parameter

internal-file-name

Identifies the SYSOUT file. If no corresponding ASSIGN statement is present in the current step, a subfile of the standard SYSOUT file is assigned automatically.

4.33.5.2 Optional Parameters

CLASS

Allows you to categorize each output file that is produced according to the requirements of a particular output group. The associated value is a single letter of the alphabet. Each class has a corresponding default output priority (See PRIORITY below).

If the CLASS parameter is omitted, the default value is C. For local outputs, the default CLASS value can be modified either at system generation time, or by using the CATMAINT command MDS (MoDify Station).

The operator can control the production of output on a class basis. For example, if printed output which uses a particular type of paper is grouped into the same output class, the operator can arrange to print all of that class at the same time. In this way, the need to change the paper is minimized.

PRIORITY

Takes an integer value between 0 and 7 and indicates the priority of selection by the Output Writer of the SYSOUT file, where 0 has the highest urgency and 7 the lowest. Therefore, PRIORITY specifies the ordering for the output of a SYSOUT file relative to other SYSOUT files that are waiting for the services of the Output Writer.

Within the output queue, SYSOUT files are ordered from the highest to the lowest priority and, for a given priority, ordering of output for display is on a FIFO (first-in-first-out) basis. The default value depends on the specified output class (or the default output class), as follows:

A	B	C	D	E	F to Z
1	2	3	4	5	6

These default values may be modified at system configuration time or by the MDC operator command. For RBF stations, they may be modified by the CATMAINT command MDS (MoDify Station). Note that the valid priority values 0 and 7 are not used in these default values, unless you specify them with the GCL command MDC OC, or by using the CONFIG or CATMAINT utilities.

NOTE: The output-priority specified here does not correspond with either the scheduling-priority of the $JOB statement, or with the step-execution-priority of the STEP statement.

WHEN

Specifies the point at which the Output Writer is notified that the output is to be displayed. One of the following values may be given:

JOB — Notification when job execution is terminated. This is the default value.

STEP — Notification when step execution is terminated. This is useful for multi-step jobs, since Output Writer action can take place concurrently with the execution of other steps of the same job.

NOTE: If the step is repeatable; WHEN=IMMED or digit5 is replaced by WHEN=STEP to avoid duplication of output. If the job is repeatable, enqueueing is delayed until the end of the job.

IMMED — Notification when the SYSOUT file is closed. This allows Output Writer action to commence before the termination of the step, if circumstances permit.

DEFER — For a permanent SYSOUT file only; no notification is sent to the Output Writer. The printing of the file must be requested by a WRITER statement, normally in a different job at a later time. When DEFER is specified, all other optional parameters except DEVCLASS and MEDIA are ignored.

digits5 — For public SYSOUT files only, a separate SYSOUT file is created each time this number of pages has been output. After each such file is completed, it may be edited by the Output Writer depending on the output class and priority. For example, if WHEN=100 each 100 pages of output constitutes a separate SYSOUT file. The Output Writer can process each file (that is, 100 pages) as soon as it is complete.

NOTE: Note that the time interval between output Writer notification and the commencement of display on the output device is clearly dependent on current Output Writer activity and on the priority associated with the SYSOUT file. Even if IMMED is specified, the printing may still be delayed beyond the execution of the step or job.

HOLD

If this parameter appears, the Output Writer will be prevented from selecting the SYSOUT file for display on the output device until the operator issues a RO command. Its effect is to suspend temporarily a SYSOUT file which is on the output request queue (the list, in order of priority, of files waiting to be output by the Output Writer). Note that the HOLD parameter takes effect after the Output Writer has been notified; therefore it is independent of the WHEN parameter. The NAME parameter should be specified when HOLD is given, in order that the operator can identify the output from this particular SYSOUT.

NHOLD

The SYSOUT file will be output without the intervention of the operator. This parameter can be used for the current file to override a HOLD parameter in a previous OUTVAL statement.

NAME

Specifies a symbolic-name to be associated with the output of a SYSOUT file produced by the Output Writer. This name can be useful in case of abnormal termination during printing of the output, and can be referred to in various operator commands. The name is composed of up to eight alphanumeric, hyphen and underscore characters and must begin with a letter. The use of NAME is strongly recommended if HOLD is specified.

NOTE: The system automatically provides a unique index for each unit of output whether or not NAME is specified. Both the symbolic-name and the index can be displayed on the operator's console by means of the DO command. In contrast, the system gives a special name to all DUMP outputs (for example DUMPs produced by the COBOL DUMP facility).

BANNER

Resets the effect of a previous BANNER specification at OUTVAL level.

NBANNER

Specifies that the banner pages are not to be printed.

BANINF

Allows you to replace on the banner pages the following four standard entries used by the system: Run Occurrence Number (RON), User-name, job-name, Billing. Each replacement value may be up to 12 characters in length. If n values are specified, where n is less than or equal to 4, then the last n entries are replaced. Empty values cannot be specified.

Example:

```
RON: X123, user-name: MAXI, job-name: PRIM, billing: CAP
```

If BANINF=(MINI, SEC, PES) is specified, the following entries will be printed:

```
    X123   MINI   SEC   PES
```

BANLEVEL

Allows you to request the insertion of a banner between each member or copy of an output, or to restrict the occurrence of a banner to cases of output, baninf, or job switching.

Note that the banners are more detailed when they concern only one member, one copy or one output; outname, file-identification, member-name or copy-number are indicated where applicable.

Any output which is enqueued immediately, at the end of the step or by the early mechanism, and which is therefore eligible to be printed separately from jor jobout, is enclosed by copy level banners.

For outputs enqueued at the end of job, such as jor jobout, job level banners are inserted only in the case of job or baninf switching, even if several copies are requested.

Note that BANLEVEL=MEMBER does not apply to outputs spooled on another site, since if this is the case, the members are gathered together by the server spooling mechanism.

BANLEVEL=JOB requests the insertion of a banner when job switching is detected by route processing. However, this has no gathering action on the different outputs of the job, and therefore should not be used for outputs that are enqueued before the end of job.

COPIES

Specifies the number of copies of the SYSOUT file data to be produced by the Output Writer for this statement. The maximum value is 99 and the default value is 1.

DEVCLASS

Gives the device class and attributes of the output device - a printer, for example. The default device class is installation dependent.

MEDIA

For a printer, the volume-name gives the print belt character set and paper identification as a six-character value, as follows:

- characters 1-2: character set;
- characters 3-6: form number.

The default value is installation dependent.

DEST

Applies to a RBF (Remote Batch Facility) environment. It gives the RBF station or the remote host (or possibly a station of this remote host), where the output is to be delivered. If DEST is not specified, the output is delivered to the station which issued the job. The DEFAULT CLASS, device, DEVICE CLASS and MEDIA are those of the station to which the output is attached.

SLEW

Any paper move instructions (slew control) are obeyed when the file is printed. This value applies by default in the absence of a previous OUTVAL statement with the value NSLEW.

NSLEW

When the file is printed by the Output Writer, every skip function is transformed to "skip to next line"; (that is, for the SYSOUT statement, each paper move instruction is reduced to a move to the next line).

DELETE

Applies only to a member of a permanent SYSOUT library file. If DELETE is specified, the subfile associated with the internal-file-name is deleted once the data to be output has been printed. Care should be taken with the use of this parameter, since the Output Writer does not check if other output requests (by SYSOUT or WRITER) are pending for this file. Normally, that is if DELETE does not appear in the current SYSOUT nor in a previous OUTVAL, no deletion of a library member occurs.

If the output concerned is cancelled by a CO [STRONG] command, any subfiles not entirely printed by the Output Writer will not be deleted.

NDELETE

Applies only to a member of a permanent SYSOUT library file; it applies by default if no previous OUTVAL has specified DELETE. If NDELETE appears, there is no deletion of the subfile after the data has been output. It can be used to override for the current SYSOUT file a DELETE parameter that appears in a previous OUTVAL statement.

Example 1:

```
SYSOUT LIST1, WHEN=STEP, NAME=HAJ;
```

This informs the Output Writer at the end of the current step to print the SYSOUT file with internal-file-name LIST1; the name HAJ is associated with the output.

Example 2:

```
SYSOUT HJPRINT, BANINF=(NJOB, PROJX);
```

The contents of the file with internal name HJPRINT will be printed. The entries NJOB and PROJX will appear on the banner pages in place of the current job-name and billing respectively. The RON and the user-name to be printed will be unchanged.

Example 3:

Consider the following sequence of JCL statements.

```
ASSIGN DMPRINT, MY.LIB, SUBFILE=TEMPA;

SYSOUT DMPRINT, COPIES=2, DELETE;
```

This produces two copies of library subfile TEMPA and deletes the subfile after it has been printed.

4.34 VALUES

4.34.1 Function

Sets default values for specified parameters in the JCL statements which follow the VALUES statement.

4.34.2 Enclosure

Job, or an INVOKE sequence or an EXECUTE sequence.

4.34.3 Format

```
VALUES [parameter-value1   [parameter-value2] ...]
       [keyword1 = keyword-value1 [keyword2 = keyword-value2]...];
```

4.34.4 Description of Statement

The VALUES statement sets values to be used for substitution in symbolic parameter references in a job enclosure. These values can also be default parameter values to permit omission in a parameter list within the VALUES parameter in INVOKE, EXECUTE and RUN or for the values which can be specified in the EJR operator command. This VALUES statement must be contained in the JCL sequence called by INVOKE, EXECUTE, RUN or the EJR operator command. The default values apply to all following JCL statements until another VALUES or MODVL statement, the end of the job enclosure or the end of the current sequence of JCL.

A parameter-value replaces a &1, where "i" indicates the position of the parameter-value in the VALUES statement (1<i<99). The keyword-value replaces a &keyword.

Parameter-values are positional parameters whereas keyword-values are contained in keyword parameters (see Section 2).

The VALUES statement cannot be used to set values in the $SWINPUT statements nor in a following VALUES statement.

A VALUES statement nullifies the preceding VALUES statement if any exists. It also nullifies the effects of preceding MODVL statements.

See the *JCL User's Guide* for the rules concerning parameter substitution in JCL statements.

4.34.5 Optional Parameters

parameter-value

The parameter-value-i defines the default value for the &i (1<=i<=99) string in an invoked JCL sequence or in a job description. This value must follow the syntax rules of the parameter(s) for which it is defined; however, it may be a protected string (that is, any character string enclosed in single quotes, except that a single quote inside a string must be entered twice in succession) of up to 128 characters. The position of a parameter value relative to other parameter-values in a VALUES statement defines the value of i for the identification of the associated &1.

If commas are being used as separators, missing positional parameter values are indicated by commas without a preceding value. If spaces are being used as separators, a # sign indicates a missing positional parameter value. Missing parameters to the right of the last value specified need not be indicated by commas or # signs. See Example 2 below.

keyword-value

The keyword-valuei defines the default value to replace every occurrence of & followed by the specified keywordi in an invoked JCL sequence or in a job description. This value must follow the syntax rules of the parameter(s) for which it is defined; however it may be a protected string of up to 128 characters. The maximum length allowed for a keyword is 8 characters.

Example 1:

```
VALUES MYFILE ,,MS/M500,STAT=TEMPRY;
```

The above statement defines the default values MYFILE and MS/M500 for every occurrence of &1 and &3 respectively and the default value TEMPRY for every occurrence of &STAT; by default, every occurrence of &2 will be replaced by an empty string (that is, the parameter will be ignored). Consider the following statement:

```
ASSIGN IFN&2A, &1, DEVCLASS=&3, MEDIA=ABC, &STAT;
```

Provided the default values specified in the above VALUES statement are not overridden (that is, by a RUN statement), the ASSIGN statement becomes:

```
ASSIGN IFNA, MYFILE, DEVCLASS=MS/M500, MEDIA=ABC, TEMPRY;
```

Example 2:

There are 10 positional parameters (&1 through &10), and values p2, p4, p6 are to be specified for the 2nd, 4th and 6th (that is, &2, &4, &6) respectively. The following 4 VALUES statements are equivalent:

```
VALUES , p2,, p4,, p6,,,,;

VALUES , p2,, p4,, p6;

VALUES # p2 # p4 # p6 # # # #;

VALUES # p2 # p4 # p6;
```

4.35 WRITER

4.35.1 Function

Requests the Output Writer mechanism for use with an existing permanent SYSOUT file.

4.35.2 Enclosure

Job.

4.35.3 Format

```
WRITER { (sequential-input-file-description)                          }
       {                                                              }
       {     [{PART = (ai [:bi] [ai+1 [:bi+1]] ...)    }]  }
       {     [{                                        }]  }
       {     [{SELECT = ( [ron [ :   index      ]] ...)}]  }
       {     [{                      output-name       }]  }
       {                                                              }
       {(input-library-description)                                   }
       {                                                              }
       {       SUBFILES = ({member-name [member-name] ...} }
       {                   {star-name [star-name]...     })}

       [CLASS = identifier1]

       [PRIORITY = digit1]

       [WHEN ={JOB  }]
       [      {IMMED}]

       [{HOLD }]
       [{NHOLD}]

       [NAME = identifier8]

       [{BANNER }]
       [{NBANNER}]

[            {JOB   }]
[            {OUTPUT}]
[BANLEVEL  = {COPY  }]
[            {MEMBER}]

 [BANINF = (alphanum12 [,alphanum12] ...)]

       [COPIES = digits2]
```

```
[DEVCLASS = device-class [MEDIA = volume-name]]

[DEST = [host-name].station-name]]

[DATAFORM ={SSF    [REPORT = {string 2}]}]
[          {SARF             {ALL     } }]
[          {DOF                 }]
[          {ASA                 }]

[NUMBER]

[{FPARAM }]
[{NFPARAM}]

[{SLEW }]
[{NSLEW}]

[{DELETE }]
[{NDELETE}]

[PRINTER = {printer-options)]
                              ;
```

where the format of 'printer-options' is given in the description of the DEFINE statement.

4.35.4 Description of Statement

The WRITER statement requests the use of the Output Writer mechanism for an already existing permanent SYSOUT file, on disk or on tape. The file may be UFAS sequential or a library file, cataloged or uncataloged, and may or may not have been created with the SYSOUT mechanism. The WRITER statement is often specified in a job other than the one in which the file was created. Reference is made to the SYSOUT mechanism in the SYSOUT statement description above; a more detailed explanation is given in the *JCL User's Guide*.

The WRITER statement allows you to specify non-standard values for output-editing parameters (for example, margin setting, printing density) and for output handling (for example, output destination, number of copies).

Specific pages of the SYSOUT file or specific subfiles of a SYSOUT library file can be selected for output.

Comments and Restrictions

Any value of the output handling parameters (for example, DEST, COPIES) specified, for example with a SYSOUT statement, when the file was written, applied only for the job step during which the file was written. If you wish to override any of the standard values of these parameters, the new values must be specified in the current WRITER statement.

If the file to be output was written using the SYSOUT mechanism, any editing parameters (that is, PRINTER) given in this WRITER statement would be ignored. In other words, it is assumed that the original specifications given when the file was written still apply. However, if you wish to override the original specifications (applied by default, or corresponding to the MEDIA value given in the SYSOUT statement, or as specified by a DEFINE statement), the FPARAM parameter must be given in the WRITER statement.

The WRITER statement may only occur in a job enclosure outside a step enclosure whereas the SYSOUT statement only occurs inside a step enclosure.

The action of the Output Writer to print the specified file is not necessarily synchronous with the execution of the job which contains the WRITER statement. Therefore a permanent SYSOUT file should not be deleted (for example, using DEALLOC) in the same job as the request to output its contents (or even in later jobs if sufficient time has not elapsed), as there is a danger that the file will be deallocated before it has been completely printed. For this reason, a temporary file or an input enclosure is not permitted. Therefore, see the description of the DELETE parameter for the SYSOUT statement.

4.35.5 Parameters

4.35.5.1 Mandatory Parameters

Either sequential-input-file-description or input-library-file-description, but not both, must appear.

The sequential-input-file-description identifies a sequential file (or a single library member) to be output by the Output Writer. The expansion of this parameter group is given in Section 3. Note that this file description can identify a single library member with the SUBFILE parameter.

In this case, it is possible to:

- print just certain pages (with the PART parameter) or;
- print several copies (with the COPIES parameter) or;
- select the reports(s) to be printed (with the SELECT parameter), if the file concerned has been produced by the Output Writer.

Example:

```
WRITER (.MYLIB,SUBFILE=S1), COPIES=6;
```

The member S1 of the cataloged library file .MYLIB is printed 6 times.

The input-library-file-description identifies the library which contains the subfiles (specified by the SUBFILES parameter described below) that are to be output by the Output Writer. The expansion of this parameter is given in Section 3.

SUBFILE

Applies only to library files. It selects for printing the required subfiles of the specified library file. The contents of all named library members are output. By the use of the star convention (see the *Library Maintenance Reference Manual*), any number of families of subfiles may be selected.

In this case, the PART, SELECT, and COPIES parameters are not applicable.

Example:

```
WRITER (.MYLIB), SUBFILES=(S1, S2, S3);
```

The members S1, S2, and S3 of the cataloged library file .MYLIB are printed.

If a single subfile is to be printed, then it is more efficient to do this with a sequential-input-file-description rather than with an input-library-file-description.

4.35.5.2 Optional Parameters

See the description of the parameters of the SYSOUT statement for an explanation of all the parameters not included below (that is, the output handling parameters CLASS, PRIORITY, HOLD, NHOLD, NAME, BANNER, NBANNER, BANINF, DEVCLASS, MEDIA, DEST, SLEW, NSLEW, DELETE, NDELETE).

PART

Allows you to define which part or parts of the specified file (or simple library subfile) is/are to be output. Each value pair (ai: bi) specifies the required part in terms of the first and last pages of the file to be output. Each value supplied must be a digit or a string of digits. The default value of any bi is ai. A list of value pairs, separated by commas, must appear in ascending order of page number (that is, the value of ai must be less than or equal to the value of bi, and the value of bi must be less than the value of aj where j = i+1). In the last pair (ai : bi), bi may take the value $ to mean that the upper limit is the end of file. In this case, the ai is necessary and it cannot take the value $.

Example:

```
PART=(4:7, 9, 11:11, 13:$)
```

This example specifies page 4, 5, 6, 7, 9, 11, and 13 to end of file. Note that the second page specification could have been written 9:9 and the third could have been written 11. Note that the last pair is valid even if 13 happens to be the end of file. However, the last pair cannot be written $:$ or $.

SELECT

Specific reports of a job may be selected by run occurrence number (ron), or by index within ron, or by output-name within ron. The SELECT parameter applies only if the specified file was produced with the Output Writer. If the specified value is a run occurrence number (ron) all outputs from the job with that ron are printed. If the value is of the form ron:output-name, the named output from the job with the specified ron is printed. If the value is of the form ron: index, the output with the specified index from the job with the specified ron is printed. (See the description of the SYSOUT statement, NAME parameter, for a brief explanation of the output index).

COPIES

Specifies the number of copies to be produced. The maximum value is 99, and the default is 1. COPIES has no effect if SUBFILES or PART is specified. COPIES is also ineffective if the file to be processed by WRITER has been created by the Output Writer.

WHEN

Specifies the point at which the Output Writer is notified that the output is to be printed. One of the following values is specified:

JOB	Notification when job execution is terminated. This is the default value.
IMMED	Notification as soon as the WRITER statement is executed.

Note that the time interval between Output Writer notification and the commencement of display on the output device is clearly dependent on current Output Writer activity and the priority of the job class associated with the SYSOUT file. Even if IMMED is specified, the printing may still be delayed beyond the execution of the job.

DATAFORM

Applies only if the SYSOUT mechanism was not used when the file was written. It is ignored for SYSOUT files in SYSOUT format. DATAFORM overrides the output data format computed by WRITER. This is only useful for ASA outputs as SSF and DOF records are automatically recognized by WRITER. SARF is assumed when the file is in neither SSF nor DOF format.

SSF	The records of the SYSOUT file contain data and an SSF header. This is the default.
SARF	The records of the SYSOUT file contain pure data only.
DOF	This must be specified when printing a 100 Program Mode permanent SYSOUT file. It is a device-dependent data format.
ASA	The records of the SYSOUT file contain data and a one-byte ASA header (that is, slew-control byte). This first character indicates the skip function to be performed during output processing of the record. See the *Unit Record Devices User's Guide* for details.

REPORT

Applies to files created by the COBOL Report Writer. When a value (two characters long) is supplied it becomes the record-identification-code; only records that are preceded by this code are taken into account by WRITER. The record-identification-code is not displayed when a record is output. If any characters other than alphanumeric ones are used for the code value, they must be enclosed in quotes. If the REPORT parameter is omitted, any record-identification-codes that exist are displayed (that is, the records are output in full).

ALL	If this value is specified for REPORT, every record of the file is output, irrespective of the presence of a record-identification-code. The record-identification-code is not displayed. This is the default.

WARNING

For the use of the Record Selection feature of the COBOL Report Writer, the file to be output must be in SSF format. Therefore the SYSOUT mechanism must not be used (when the file is written) to edit the file to SYSOUT format (see the SYSOUT statement above and the *JCL User's Guide* for details of the SYSOUT mechanism).

NUMBER

Applies only if SSF source library members are being printed. It specifies that the line number (in the SSF header) must also be edited. This parameter is only available for non-temporary libraries.

FPARAM

Applies only to files in SYSOUT format (that is, edited with the SYSOUT mechanism). This parameter is required if any editing parameters (PRINTER) which have been specified in this WRITER statement are to be obeyed. If FPARAM is omitted, the original specifications at the time of file writing will apply even if PRINTER is given in the WRITER statement. In other words, FPARAM is used to force the current editing parameters.

NFPARAM

The opposite of FPARAM. Editing parameters currently being supplied are not valid for files already in SYSOUT format. This is the default if FPARAM is not specified.

PRINTER

Applies to files in SYSOUT format provided FPARAM is specified, and to files not written in SYSOUT format. The parameter values supplied here override those that apply by default (and, in the case of files in SYSOUT format when FPARAM is specified, those applied by DEFINE at file creation).

printer-options	Allow you to specify editing parameters for files to be output on a line printer. See the DEFINE statement for the descriptions of the parameters (but note that SLEW and NSLEW are not included in the PRINTER parameter set for WRITER - see the WRITER format above).

Examples:

```
WRITER HJ.REPS, COPIES=3;
```

This prints 3 copies of the permanent SYSOUT file HJ.REPS.

```
WRITER NOV.FIGS, PRINTER=(PRDEN=8, FORMHT=80);
```

Assuming that the file NOV.FIGS was not written with the SYSOUT mechanism, this WRITER statement prints its contents, setting the print density to 8 lines per inch and the value of FORMHT to 80.

5. Extended JCL Statements

The extended Job Control Language is similar in format to the basic Job Control Language. However, each extended JCL statement is usually expanded into a step. Therefore, except for SORTWORK and GSORTWK, you must never write the statement inside a step enclosure. The rules for writing basic JCL also apply to writing extended JCL statements.

The extended JCL statements are shown in alphabetical order in Table 5-1 on the following pages. Note that the extended JCL statements EXDIR, EXECUTE, and RUN, are described in Section 4 of this manual (they are treated as Basic JCL Statements).

In Table 5-1, the **Statement** column lists the name of the extended JCL statement. The **Document** column gives the reference code of the manual in which you can find detailed information about the statement. The **Function** column gives a brief description of the extended JCL statement. The document reference codes are explained in Table 5-2.

Table 5-1. Extended JCL Statements (1/3)

Statement	Document	Function
BINDER	BIUG	Enables execution of Binder commands.
C	CLUG	Compiles a C source program
CATALOG	CATM	Places an object (file or directory) in a catalog.
CATBUILD	CATM	Creates a private catalog.
CATCHECK	CATM	Checks the validity of catalog structures.
CATDELET	CATM	Deletes a private catalog.
CATEXTD	CATM	Increases the size of a catalog.
CATLIST	CATM	Lists catalog structures.
CATMAINT	SADM	Creates, modifies and deletes information in the site catalog.
CATMODIF	CATM	Modifies the description of a file in a catalog.
CATMOVE	CATM	Moves (copies) structures from one catalog to another.
CBL	COUG	Compiles a COBOL source program.
CMDMGT	GCPM	Compiles GCL source procedures.
COMPARE	DMUT	Compares logically the contents of two separate files and prints the differences.
CREATE	DMUT	(1) Loads or reorganizes a UFAS file. (2) Copies an IDS/II file.
DEALLOC	DMUT	Re-allocates the space previously allocated to a disk, diskette or cataloged tape file.
DMLPROC	IDUG	Pre-processes a COBOL program that contains DML statements.
DUMPJRNL	TDSU	Extracts the OWN data of a TDS user from the After Journal and appends them to a TDS User Journal File.
FILALLOC	DMUT	Preallocates space for any kind of disk file.
FILCHECK	DMUT	Ensures that a sequential or library disk file may be processed by the standard access methods.
FILDUPLI	DMUT	Copies the contents of a source file into another file of identical type.
FILLIST	DMUT	Lists selected subsets of the label, catalog and usage information of a specified file.
FILMAINT	DMUT	(1) Changes data or control structures inside the physical records of a disk file. (2) Dumps the physical records of a native disk or tape file.
FILMODIF	DMUT	(1) Changes the name and/or expiration date of a disk or diskette file. (2) Modifies the catalog status of a disk or tape file. (3) Clears data from a disk or diskette file.
FILREST	DMUT	Restores the contents of a disk file from a tape file or a sequential UFAS disk file.

Table 5-1. Extended JCL Statements (2/3)

Statement	Document	Function
FILSAVE	DMUT	Saves the entire contents of a disk file onto a tape file.
FILTFR	UFTG	Transfers a file to a remote site, and may create, rename, or delete the file.
FORMGEN	FOUG	Invokes the Format Generator Utility program which generates forms to be used on a screen.
FOR77	F7UG	Compiles a FORTRAN77 source program.
GPL	GPUG	Compiles a native GPL source program.
GSORT	SMUG	Concatenates up to 9 input files and sorts the resulting file according to keys specified in a separate file.
GSORTWRK	SMUG	Specifies the device type (tape or disk) and amount of work space to be allocated when the SORT utility is dynamically invoked (e.g. through the SORT verb) within a user program.
JAGEN	TDSU	Calls the After Journal Maintenance Processor.
JASMAINT	FRFU	Enables execution of JAS maintenance functions.
LIBALLOC	DMUT	Requests space allocation for user libraries.
LIBDELET	DMUT	Deletes a library and deallocates its space or erases the contents of its directory and members while retaining the library space.
LIBMAINT	LMRM	Calls the library maintenance processor
LINKER	LKUG	Links one or compile units into an executable load module.
MERGE	SMUG	Merges up to 8 input files according to keys specified in a separate file.
MIRFIL	DMUT	Duplicates the contents of a MIRROR ALONE disk to an empty standard FBO-formatted disk, creating a pair of mirrored disks.
MIRSTART	DMUT	Creates a pair of mirrored disks from two standard non-MIRROR FBO-formatted disks without any files.
PASCAL	PAUG	Compiles a native PASCAL source program.
PREALLOC	DMUT	Allocates space for a disk or cataloged tape file.
PRINT	DMUT	Prints records from a file.
ROLLFWRD	TDSU	Brings restored files to restart point by using the After Journal.
SETLIST	DMUT	Produces a report which displays the names of all the member files of file-sets and shows the pairing of input and output files.

Table 5-1. Extended JCL Statements (3/3)

Statement	Document	Function
SHIFT	CATM	Updates the generation group of a cataloged file.
SORT	SMUG	Concatenates up to 9 input files and sorts the resultant file according to keys specified in a separate file.
SORTIDX	DMUT	Sorts and loads the secondary indexes of a UFAS indexed sequential file.
SORTWORK	SMUG	Specifies the device type (tape or disk) and amount of work space to be allocated when the SORT utility is dynamically invoked (e.g. through the SORT verb) within a user program.
SYSMAINT	TDAG	Enables execution of system maintenance commands.
TAPEPREP	DMUT	Labels and formats a tape volume for use by GCOS, or removes the existing standard labels.
UNCAT	CATM	Removes an object from a catalog.
URINIT	GSOG	Updates the system file that contains unit record device control parameters.
VOLCHECK	DMUT	Checks the integrity of a disk volume.
VOLCOMP	DMUT	Compares two tape or disk volumes of the same type and reports the differences.
VOLDUPLI	DMUT	Copies the contents of an entire disk or tape volume onto another volume of the same type.
VOLLIST	DMUT	Lists the contents of a native disk, or tape volume.
VOLMAINT	DMUT	Displays physical records from a named disk, diskette or tape volume. It can also modify the physical records of a disk volume.
VOLMODIF	DMUT	(1) Declares defective tracks on disk volumes without deleting the files on the volume. (2) Formats free-extent disk areas; (3) Resets the sequential allocation flag on diskette volumes.
VOLPREP	DMUT	Labels and formats a disk or tape volume for use by GCOS, or removes the existing standard labels.
VOLREST	DMUT	Restores the contents of a disk volume from a tape file.
VOLSAVE	DMUT	Saves the contents of a disk volume into a native labelled tape file.
VSETLIST	ASMG	Produces a list of all the volumes in the configuration, showing the space usage and availability.

Table 5-2. Explanation of Document Codes used in Table above

Code	Manual Name
ASMG	*Administering the Storage Manager Guide 47 A2 36UF*
BIUG	*Binder User's Guide 47 A2 11UP*
CATM	*Catalog Management Manual 47 A2 35UF*
CLUG	*C Language User's Guide 47 A2 60UL*
COUG	*COBOL 85 User's Guide 47 A2 06UL*
DMUT	*Data Management Utilities Manual 47 A2 34UF*
F7UG	*FORTRAN 77 User's Guide 47 A2 16UL*
FSEG	*Full Screen Editor User's Guide 47 A2 06UP*
FOUG	*FORMS User's Guide 47 A2 15UJ*
FRFU	*File Recovery Facilities User's Guide 47 A2 37UF*
GCPM	*GCL Programmer's Manual 47 A2 36 UJ*
GPUG	*GPL User's Guide 47 A2 36UL*
GSOG	*GCOS 7-V8 System Operator's Guide 47 A2 53US*
IDUG	*IDS/II User's Guide 47 A2 12UD*
IDUG	*Full IDS/II User's Guide 47 A2 07UD*
LKUG	*Linker User's Guide 47 A2 10UP*
LMRM	*Library Maintenance Reference Manual 47 A2 01UP*
PAUG	*PASCAL User's Guide 47 A2 52UL*
SADM	*System Administrator's Manual 47 A2 54US*
SMUG	*Sort/Merge User's Guide 47 A2 08UF*
TDAG	*TDS Administrator's Guide 47 A2 32UT*
TDSU	*TDS COBOL Programmer's Guide 47 A2 33UT*
UFTG	*UFT User's Guide 47 A2 13UC*

A. JCL Statement Names and Abbreviations

This appendix lists, in alphabetical order, the names of all Basic and Extended JCL statements for which standard abbreviations exist. The acceptable abbreviation(s) is given next to each statement.

Examples of job descriptions, first with complete statements, and then with abbreviated statements, are given at the end of this appendix.

NOTES:

1. The names of statements are given in complete form throughout this *JCL Reference Manual*, and in GCOS 7 documentation in general.

2. Not all JCL statements have acceptable abbreviations.

Table A-1. JCL Statement Names and Abbreviations (1/2)

Statement Name	Abbreviation(s)
ALLOCATE	ALC
ASSIGN	ASG
ATTACH	ATT
CATALOG	CAT
CATBUILD	CBD
CATCHECK	CCK
CATDELET	CDL
CATEXTD	CEXT
CATLIST	CLS
CATMAINT	CMN
CATMOVE	CMV
COBOL	CBL
COMMENT	COMM
COMPARE	CMP
CONSOLE	CNSL
CREATE	CR
DEALLOC	DALC
DEFINE	DEF
DMLPROC	DMLP
ENDDATA	EDATA
ENDINPUT	EIN
ENDJOB	EJOB
ENDSTEP	EST ESTP
EXECUTE	EXC
FILCHECK	FCK
FILDUPLI	FDUP
FILMODIF	FMOD
FILREST	FRST
FILSAVE	FSV
FORTRAN77	F7C
INPUT	IN
INVOKE	IVK IV
JASMAINT	MNJAS
LABEL	LBL
LIBALLOC	LALC
LIBDELET	LDL
LIBMAINT	LMN
LINKER	LINK LK
MERGE	MRG
MESSAGE	MSG
MODIFY	MOD
MODVL	MVL
OUTVAL	OVL

Table A-2. JCL Statement Names and Abbreviations (2/2)

Statement Name	Abbreviation(s)	
PREALLOC	PALC	
PREFIX	PFX	
PRINT	PR	
QASSIGN	QASG	
RELEASE	RLS	
REPLACE	REPL	
REPORT	RPT	
SFLIST	SFLS	
SHIFT	SHF	
SIZE	SZ	
SORTFMT	SFMT	
SORDIDX	SIDX	
SORTWORK	SWK	
STEP	STP	
SWINPUT	SWI	
SYSOUT	SYO	
UNCAT	UNC	
VALUES	VALUE	VL
VOLCHECK	VCK	
VOLCOMP	VCMP	
VOLDUPPLI	VDUP	
VOLPREP	VPRP	
VOLREST	VRST	
VOLSAVE	VSV	
WRITER	WR	

Examples of Job Descriptions

Example 1, with statement names given in full:

```
$JOB SCOTTY, USER = TOM, PROJECT = ONE;
 OUTVAL;
 OUTVAL COPIES = 2;
  STEP TSTA, MST.CDM;
   ASSIGN IN99, .DAT99;
   ASSIGN SCR99, .ERR99;
  ENDSTEP;
 JUMP CONTINUE;
 JUMP END, SEV, LT, 3;
  WRITER .ERR99;
 END:
$ENDJOB;
```

Example 2, with abbreviated statement names:

```
$JOB SCOTTY, USER = TOM, PROJECT = ONE;
 OVL;
  OVL COPIES = 2;
  STP TSTA, MST.CDM;
   ASG IN99, .DAT99;
   ASG SCR99, .ERR99;
  EST;
 JUMP CONTINUE;
 JUMP END, SEV, LT, 3;
  WR .ERR99;
 END:
$ENDJOB;
```

B. Character Conversion

This appendix contains conversion tables for the Extended Binary-Coded Decimal Interchange Code (EBCDIC) codes, and the American Standard Code for Information Exchange (ASCII) hexadecimal codes.

These tables also give the COBOL collating sequences, and the corresponding graphic representations or meanings, as interpreted internally by the DPS 7000 system.

B.1 ASCII

Denotes that no control code or graphic symbol is present

COBOL C/S denotes "collating sequence", obtained from the decimal conversion of the EBCDIC hexadecimal value, then add 1.

ASCII	EBCDIC	COBOL C/S	Code : Symbol	ASCII	EBCDIC	COBOL C/S	Code : Symbol	ASCII	EBCDIC	COBOL C/S	Code : Symbol
00	00	1	NUL	28	4D	78	(	50	D7	216	P
01	01	2	SOH	29	5D	94	)	51	D8	217	Q
02	02	3	STX	2A	5C	93	°	52	D9	218	R
03	03	4	ETX	2B	4E	79	+	53	E2	227	S
04	37	56	EOT	2C	68	108	'	54	E3	228	T
05	2D	46	ENQ	2D	60	97	-	55	E4	229	U
06	2E	47	ACK	2E	48	76	.	56	E5	230	V
07	2F	48	BEL	2F	61	98	/	57	E6	231	W
08	18	23	BSV	30	F0	241	0	58	E7	232	X
09	05	6	HTV	31	F1	242	1	59	E8	233	Y
0A	25	38	LFV	32	F2	243	2	5A	E9	234	Z
0B	0B	12	VTV	33	F3	244	3	5B	4A	75	[c
0C	0C	13	FFV	34	F4	245	4	5C	E0	225	\
0D	0D	14	CRV	35	F5	246	5	5D	5A	91	] !
0E	0E	15	SOV	36	F6	247	6	5E	5F	96	^ Ø
0F	0F	16	SIV	37	F7	248	7	5F	6D	110	-
10	10	17	DLE	38	F8	249	8	60	79	122	.
11	11	18	DC1	39	F9	250	9	61	81	130	a
12	12	19	DC2	3A	7A	123	:	62	82	131	b
13	13	20	DC3	3B	5E	95	;	63	83	132	c
14	3C	61	DC4	3C	4C	77	<	64	84	133	d
15	3D	62	NAK	3D	7E	127	=	65	85	134	e
16	32	51	SYN	3E	6E	111	>	66	86	135	f
17	26	39	ETB	3F	6F	112	?	67	87	136	g
18	18	25	CAN	40	7C	125	0	68	88	137	h
19	19	26	EMV	41	C1	194	A	69	89	138	i
1A	3F	64	SUB	42	C2	195	B	6A	91	146	j
1B	27	40	ESC	43	C3	196	C	6B	92	147	k
1C	1C	29	FSV	44	C4	197	D	6C	93	148	l
1D	10	30	GSV	45	C5	198	E	6D	94	149	m
1E	1E	31	RSV	46	C6	199	F	6E	95	150	n
1F	1F	32	USV	47	C7	200	G	6F	96	151	o
20	40	65	V	48	C8	201	H	70	97	152	p
21	4F	80	| |	49	C9	202	I	71	98	153	q
22	7F	128	"	4A	D1	210	J	72	99	154	r
23	7B	124	#	4B	D2	211	K	73	A2	163	s
24	5B	92	$	4C	D3	212	L	74	A3	164	t
25	6C	109	%	4D	D4	213	M	75	A4	165	u
26	5Q	81	&	4E	D5	214	N	76	A5	166	v
27	7D	126	'	4F	D6	215	O	77	A6	167	w

Denotes that no control code or graphic symbol is present

COBOL C/S denotes "collating sequence", obtained from the decimal conversion of the EBCDIC hexadecimal value, then add 1.

ASCII	EBCDIC	COBOL C/S	Code : Symbol	ASCII	EBCDIC	COBOL C/S	Code : Symbol	ASCII	EBCDIC	COBOL C/S	Code : Symbol	ASCII	EBCDIC	COBOL C/S	Code : Symbol
78	A7	168	x	A0	41	66		C6	8C	141	¶	F0	DC	221	ü
79	A8	169	y	A1	42	67	a	C7	8D	142	Y	F1	DD	222	ù
7A	A9	170	z	A2	43	68	a	C8	8E	143	ρ	F2	DE	223	ú
7B	C0	193	{	A3	44	69	a	C9	8F	144	±	F3	DF	224	ÿ
								CA	90	145	°	F4	EA	235	2
7C	6A	107	|	A4	45	70	a	CB	9A	155	a	F5	EB	236	Ő
7D	D0	209	}	A5	46	71	a					F6	EC	237	Õ
7E	A1	162	-	A6	47	72	a	CC	9B	156	°	F7	ED	238	ò
7F	07	8	DEL	A7	48	73	ç	CD	9C	157	æ				
								CE	9D	158		F8	EE	239	ó
80	20	33		A8	49	74	ñ	CF	9E	159	Æ	F9	EF	240	õ
81	21	34		A9	51	82	é	D0	9F	160	o	FA	FA	251	3
82	22	35		AA	52	83	è	D1	A0	161	µ	FB	FB	252	ü
83	23	36		AB	53	84	ë					FC	FC	253	ù
								D2	AA	171	i	FD	FD	254	ú
84	24	37		AC	54	85	è	D3	AB	172	¿	FE	FE	255	u
85	15	22		AD	55	86	f	D4	AC	173	D	FF	FF	256	
86	06	7		AE	56	87	î	D5	AD	174	γ				
87	17	24		AF	57	88	ï	D6	AE	175	ρ				
								D7	AF	176	•				
88	28	41		80	58	89	i								
89	29	42		81	59	90	B	D8	80	177					
8A	2A	43		82	62	99	A	D9	81	178	£				
8B	2B	44		83	63	100	Ä	DA	82	179	γ				
								DB	83	180	.				
8C	2C	45		84	64	101	A	DC	84	181	°				
8D	09	10		85	65	102	A	DD	85	182	§				
8E	0A	11		86	66	103	A								
8F	1B	28		87	67	104	A	DE	86	183	¶				
								DF	87	184	¼				
90	30	49		88	68	105	Ç	E0	88	185	½				
91	31	50		89	69	106	N	E1	89	186	¾				
92	1A	27		BA	70	113	Ø	E2	8A	187	¿				
93	33	52		BB	71	114	É	E3	8B	188	³				
94	34	53		BC	72	115	Ê	E4	BC	189	-				
95	35	54		BD	73	116	Ë	E5	BD	190	-				
96	36	55		BE	74	117	È	E6	BE	191	.				
97	08	9		BF	75	118	Í	E7	BF	192	x				
							Î	E8	CA	203					
98	38	57		C0	76	119	Ï	E9	CB	204	ô				
99	39	58		C1	77	120	Ï								
9A	3A	59		C2	78	121	Ï	EA	CC	205	ö				
9B	3B	60		C3	80	129	Ø	EB	CD	206	ò				
				C4	8A	139	«	EC	CE	207	ó				
9C	04	5		C5	8B	140	»	ED	CF	208	õ				
9D	14	21						EE	DA	219					
9E	3E	63						EF	DB	220	û				
9F	E1	223	÷												

B.2 EBCDIC

Denotes that no control code or graphic symbol is present

COBOL C/S denotes "collating sequence", obtained from the decimal conversion of the EBCDIC hexadecimal value, then add 1.

EBCDIC	ASCII	COBOL C/S	Code : Symbol	EBCDIC	ASCII	COBOL C/S	Code : Symbol	EBCDIC	ASCII	COBOL C/S	Code : Symbol
00	00	1	NUL	28	88	41		50	26	81	&
01	01	2	SOH	29	89	42		51	A9	82	é
02	02	3	STX	2A	8A	43		52	AA	83	ê
03	03	4	ETX	2B	8B	44		53	AB	84	ë
04	9C	5		2C	8C	45		54	AC	85	è
05	09	6	HTV	2D	05	46	ENQ	55	AD	86	ì
06	86	7		2E	06	47	ACK	56	AE	87	î
07	7F	8	DEL	2F	07	48	BEL	57	AF	88	ï
08	97	9		30	90	49		58	80	89	ì
09	8D	10		31	91	50		59	81	90	ß
0A	8E	11		32	16	51	SYN	5A	5D	91	] !
0B	0B	12	VTV	33	93	52		5B	24	92	$
0C	0C	13	FFV	34	94	53		5C	2A	93	°
0D	0D	14	CRV	35	95	54		5D	29	94	)
0E	0E	15	SOV	36	96	55		5E	3B	95	;
0F	0F	16	SIV	37	97	56	EQT	5F	5E	96	^]
10	10	17	DLE	38	98	57		60	2D	97	-
11	11	18	DC1	39	99	58		61	2F	98	/
12	12	19	DC2	3A	9A	59		62	B2	99	Å
13	13	20	DC3	3B	9B	60		63	B3	100	Á
14	9D	21		3C	14	61	DC4	64	B4	101	Å
15	85	22		3D	15	62	NAK	65	B5	102	Ä
16	08	23	BSV	3E	9E	63		66	B6	103	Ã
17	87	24		3F	1A	64	SUB	67	B7	104	Â
18	18	25	CAN	40	20	65	v	68	B8	105	ç
19	19	26	EMV	41	A0	66		69	B9	106	Ñ
1A	92	27		42	A1	67	ä	6A	7C	107	|
1B	8F	28		43	A2	68	a	6B	2C	108	.
1C	1C	29	FSV	44	A3	69	à	6C	25	109	%
1D	10	30	GSV	45	A4	70	á	6D	5F	110	-
1E	1E1E	31	RSV	46	A5	71	ã	6E	3E	111	>
1F	1F1F	32	USV	47	A6	72	å	6F	3F	112	?
20	80	33		48	A7	73	ç	70	BA	113	Ø
21	81	34		49	A8	74	ñ	71	BB	114	É
22	82	35		4A	5B	75	[ç	72	BC	115	È
23	83	36		4B	2E	76	.	73	BD	116	Ë
24	84	37		4C	3C	77	<	74	BE	117	È
25	0A	38	LFV	4D	28	78	(	75	BF	118	Ì
26	17	39	ETB	4E	2B	79	+	76	C0	119	Í
27	1B	40	ESC	4F	21	80	!]	77	C1	120	Ï

Denotes that no control code or graphic symbol is present

COBOL C/S denotes "collating sequence", obtained from the decimal conversion of the EBCDIC hexadecimal value, then add 1.

EBCDIC	ASCII	COBOL C/S	Code : Symbol	EBCDIC	ASCII	COBOL C/S	Code : Symbol	EBCDIC	ASCII	COBOL C/S	Code : Symbol	ASCII	EBCDIC	COBOL C/S	Code : Symbol
78	C2	121	Ì	A0	D1	161	m	C6	46	199	F	F0	30	241	0
79	60	122	\	A1	7E	162	-	C7	47	200	G	F1	31	242	1
7A	3A	123	:	A2	73	163	s	C8	48	201	H	F2	32	243	2
7B	23	124	#	A3	74	164	t	C9	49	202	I	F3	33	244	3
								CA	E8	203	■	F4	34	245	4
7C	40	125	0	A4	75	165	u	CB	E9	204	ô	F5	35	246	5
7D	27	126	'	A5	76	166	v					F6	36	247	6
7E	3D	127	=	A6	77	167	w	CC	EA	205	ö	F7	37	248	7
7F	22	128	»	A7	78	168	x	CD	EB	206	Ò				
								CE	EC	207	Ó	F8	38	249	8
80	C3	129	0	A8	79	169	ỹ	CF	ED	208	ö	F9	39	250	9
81	61	130	a	A9	7A	170	z	D0	7D	209	}	FA	FA	251	3
82	62	131	b	AA	D2	171	i	D1	4A	210	J	FB	FB	252	U
83	63	132	c	AB	D3	172	¿					FC	FC	253	Ü
								D2	4B	211	K	FD	FD	254	Ú
84	64	133	d	AC	D4	173	D	D3	4C	212	L	FE	FE	255	Ù
85	65	134	e	AD	D5	174	Y	D4	4D	213	M	FF	FF	256	■
86	66	135	f	AE	D6	175	ρ	D5	4E	214	N				
87	67	136	g	AF	D7	176	•	D6	4F	215	O				
								D7	50	216	P				
88	68	137	h	80	D8	177	c								
89	69	138	i	81	D9	178	£	D8	51	217	Q				
8A	C4	139	«	82	DA	179	γ	D9	52	218	R				
8B	C5	140	»	83	DB	180	.	DA	EE	219	I				
								DB	EF	220	û				
8C	C6	141	∂	84	DC	181	o	DC	F0	221	ü				
8D	C7	142	y	85	DD	182	§	DD	F1	222	ù				
8E	C8	143	ρ	86	DE	183	¶								
8F	C9	144	±	87	DF	184	¼	DE	F2	223	ü				
								DF	F3	224	ÿ				
90	CA	145	°	B8	E0	185	½	E0	5C	225	\				
91	6A	146	j	B9	E1	186	¾	E1	9F	226	+				
92	6B	147	k	BA	E2	187	¿	E2	53	227	S				
93	6C	148	l	BB	E3	188	I	E3	54	228	T				
94	6D	149	m	BC	E4	189	-	E4	55	229	U				
95	6E	150	n	BD	E5	190	"	E5	56	230	V				
96	6F	151	o	BE	E6	191	.	E6	57	231	W				
97	70	152	p	BF	E3	192	x	E7	58	232	X				
								E8	59	233	Y				
98	71	153	q	C0	7B	193	{	E9	5A	234	Z				
99	72	154	r	C1	41	194	A								
9A	CB	155	°	C2	42	195	B	EA	F4	235	2				
9B	CC	156	°	C3	43	196	C	EB	F5	236	Ô				
				C4	44	197	D	EC	F6	237	O				
9C	CD	157	æ	C5	45	198	E	ED	F7	238	Ó				
9D	CE	158						EE	F8	239	Ò				
9E	CF	159	Æ					EF	F9	240	Ö				
9F	D0	160	o												

Glossary

basic JCL statement	A statement that requests a specific system operation.
batch job	A job that runs without user intervention.
boolean parameter	A parameter that takes only the values 0 (NO) or 1 (YES).
cartridge library	A cartridge storage module equipped with a robot to perform the mounting/dismounting operations. Allows unattended cartridge operations.
catalog	A file that records the links between a master directory and its files.
compaction	A hardware recording mode on cartridge, following the IDRC standard (Improved Data Recording Capability), and increasing the effective cartridge data capacity.
control interval (CI)	The unit of transfer between main memory and a file on disk.
conventional batch job	A job that runs sequences of Job Control Language (JCL) statements.
CU library	Compiled Unit library - a library that holds compiled programs.
data enclosure	The part of a job beginning with a $DATA statement and ending with a $ENDDATA statement, that contains data to be loaded into a specific library.
data field	A unit of information belonging to a record.
device class	Specifies the type of disk, cartridge, tape drive.
directory	A group of files in a hierarchical tree structure.
EJR	See ENTER_JOB_REQUEST operator command.
ENTER_JOB_REQUEST operator command	Requests the execution of one or more jobs or submits a jobset request.
E-tape	Enhanced-capacity half-inch cartridge tape (0.5 mil thick, 1100 feet long), it is also called long tape or thin tape. Is two tone colored and can only be used on 36-track devices.

EXECUTING state	The job state where an IN SCHEDULING job is executed.
extended JCL statement	A statement that expands into a set of basic JCL statements and is equivalent in itself to a complete step description.
file status	Describes whether a file is catalogued, uncatalogued or temporary.
file	The basic unit of information maintained by the operating system.
first in, first out (FIFO)	Priority order whereby the first job to enter is treated first by the system.
full path name	The complete name of a file, which includes the name(s) of the directory(ies) where it is located.
GCL	See GCOS 7 Command Language.
GCOS 7 Command Language	The set of commands used by IOF terminal users to manage the GCOS 7 functions on the DPS 7000.
GCOS 7	General Comprehensive Operating System - operating system for the DPS 7000.
HJ	See HOLD_JOB operator command.
HOLD_JOB operator command	An operator command that puts a job in the HOLD state.
HOLD state	The job state where a job is temporarily suspended by a HOLD command.
IDLE state	Job state for the service jobs JCL Translator and WRITER when they are not active.
IN SCHEDULING state	The job state where a job is ready for execution.
input enclosure	The part of a job beginning with a $INPUT statement and ending with a $ENDINPUT statement, that contains data records to be associated with one or more steps in the job.
input library	A library used to read information from.
input reader	The input reader is in charge of reading a stream of job descriptions, analyzing each job and translating the JCL to a format recognizable by GCOS, and storing the corresponding data and JCL statements separately for scheduling and executing the job later.
input stream	The input stream is the work that is normally submitted to the operating system, and which consists of one or more job descriptions.

INTRODUCED state	The first job state, when a new job is translated into JCL.
IOF	Interactive Operation Facility - The GCOS 7 time-sharing application.
JCL Translator	The service job that translates a newly-submitted job into internal form.
JCL	See Job Control Language.
job	Work submitted to the system.
job class	A group of jobs that perform related tasks.
Job Control Language	The GCOS 7 language that is used to control the execution of jobs.
job description	A set of JCL statements which supplies information about jobs and directs how they are run.
job enclosure	A complete set of JCL statements that make up a job description, starting with a $JOB statement and finishing with a $ENDJOB statement.
Job Occurrence Report	A printed report that contains all system information concerning the execution of any job step that produces output.
job run	The entire life of a job.
Job Scheduler	Controls the system load at job level and enables you to organize the work.
job state	One phase of the life of a job.
job step	A segment of a job.
JOB_OUT	A printed report that contains all system information concerning the execution of any job step that produces output.
JOR	See Job Occurrence Report.
library maintenance utility	The utility that contains the commands used to modify and maintain libraries.
library	A file organized into a series of subfiles (members).
load module	A program known to the system at execution time.
main operator	The operator who manages the daily functioning of a GCOS 7 system.
mandatory parameter	A parameter of a command that requires a value in order for the command to be executed.

master directory	The first directory level in a hierarchical tree structure.
media	a disk, tape or cartridge.
member	A library subfile.
Multiprogramming Class Limit (MCL)	The maximum number of jobs that can be simultaneously executed or suspended in a job class.
multiprogramming	The environment whereby more than one job is executing at any one time.
non-resident volume	A volume that is loaded on a device only when it is needed.
non-standard tape	A tape without a standard label.
output library	A library used for read and write operations.
output priority	The order in which jobs are printed by WRITER.
OUTPUT state	The job state during which the outputs of a job are printed.
Output Writer	The program that controls the selection and printing of queued outputs for the WRITER service job.
permanent file	A file created for long-term use and stored on a volume.
private catalog	A catalog that holds information on the files belonging to one project only.
processor	A group of commands that performs a specific set of functions in GCOS 7.
project	A logical group of users.
protected string	A string of characters enclosed in single quotes (').
RELEASE_JOB operator command	An operator command that puts a job from a HOLD state into the IN SCHEDULING state.
resident volume	A volume that is permanently on-line (available) to the system.
RJ	See RELEASE_JOB operator command.
RON	See Run Occurrence Number.
root	The basis of a hierarchical tree structure.
Run Occurrence Number	The number assigned to a job after it is submitted to the system.
Scheduling Priority (SPR)	The order in which jobs are queued while waiting to be run.

simple name	Each component of the full path name of a file.
site catalog	The catalog that holds information on the SITE files and on all other directories and their contents.
SL library	Source Language library - a library that holds any kind of textual information.
spooling	From **S**imultaneous **P**eripheral **O**perations **O**n **L**ine; spooling refers to printing a document or file while allowing users to work on something else.
step	Specifies the loading and execution of a program, known at execution time as a load module.
standard labelled tape	A tape that can be read on the device on which it has been mounted, with labels conforming to the ISO/ECMA standard (for cartridges, in EBCDIC format only).
standard length half-inch tape	A half-inch cartridge tape used on 18-track device (541 to 807 feet long). It is also called short tape or thick tape. It can be used on 36-track devices as well.
step enclosure	Part of a job enclosure. A typical job enclosure is made up of one or more step enclosures.
stream reader	A load module in a specific job that reads input, assigns the input device and waits for the first record.
SUSPENDED state	The job state where a job is suspended while EXECUTING.
symbolic names	A special set of names (LIB, INLIB1, INLIB2, and INLIB3) that represents the working libraries.
SYS.OUT	The standard system file for printing job outputs.
SYSIN mechanism	Allows programs that read data from a unit record input device to execute concurrently without having to deal with device assignment problems.
System Administrator	The person responsible for the technical management of a GCOS 7 system.
system catalog	The catalog that holds information on System (SYS.*) files.
temporary file	A file created for the duration of a job step or log-on session only.
Text Editor	The GCOS 7 line editor used to create and modify library members.
UFAS indexed sequential file	A disk file that contains indexes used to locate records.

volume	A disk, tape or cartridge used to save information.
volset	A set of disks, tapes or cartridges used to save information
working directory	The directory that contains the user's working files.
working libraries	The input and output libraries that are active in the current session.
WRITER	The service job that manages the printing of job output.
18-track cartridge	Cartridge on which the 18-track recording format is used. It is always a standard length half-inch cartridge.
18-track device	A device that uses the18-track recording format to write data. It can use standard length half-inch tapes only.
36-track cartridge	Cartridge on which the 36-track recording format is used. It can be a standard length half-inch tape or an E-tape.
36-track device	A device that uses the 36-track recording format to write data. It can use standard length half-inch tapes and E-tapes.

Index

.

=

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Technical publication remarks form

Title :	DPS7000/XTA NOVASCALE 7000 JCL Reference Manual Job Control and IOF

Reference N° :	47 A2 11UJ 04	**Date :**	**December 1997**

ERRORS IN PUBLICATION

SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

Your comments will be promptly investigated by qualified technical personnel and action will be taken as required.
If you require a written reply, please include your complete mailing address below.

NAME : _____ Date : _____

COMPANY : _____

ADDRESS : _____

_____

Please give this technical publication remarks form to your BULL representative or mail to:

Bull - Documentation Dept.
1 Rue de Provence
BP 208
38432 ECHIROLLES CEDEX
FRANCE
info@frec.bull.fr

Technical publications ordering form

To order additional publications, please fill in a copy of this form and send it via mail to:

BULL CEDOC
357 AVENUE PATTON
B.P.20845
49008 ANGERS CEDEX 01
FRANCE

Phone: +33 (0) 2 41 73 72 66
FAX: +33 (0) 2 41 73 70 66
E-Mail: srv.Duplicopy@bull.net

CEDOC Reference #	Designation	Qty
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
[_ _] : The latest revision will be provided if no revision number is given.		

NAME: ______________________________ Date: ____________

COMPANY: ______________________________

ADDRESS: ______________________________

____

PHONE: ______________________ FAX: ______________________

E-MAIL: ______________________________

For Bull Subsidiaries:

Identification: ______________________________

For Bull Affiliated Customers:

Customer Code: ______________________________

For Bull Internal Customers:

Budgetary Section: ______________________________

For Others: Please ask your Bull representative.

BULL CEDOC
357 AVENUE PATTON
B.P.20845
49008 ANGERS CEDEX 01
FRANCE

REFERENCE
47 A2 11UJ 04