

**Chapter
1**

**RUNNING
A SIMPLE JOB**

*Get on the
Fast Track!*



TM

**SYS-ED/
Computer
Education
Techniques, Inc.**

Objectives

You will learn:

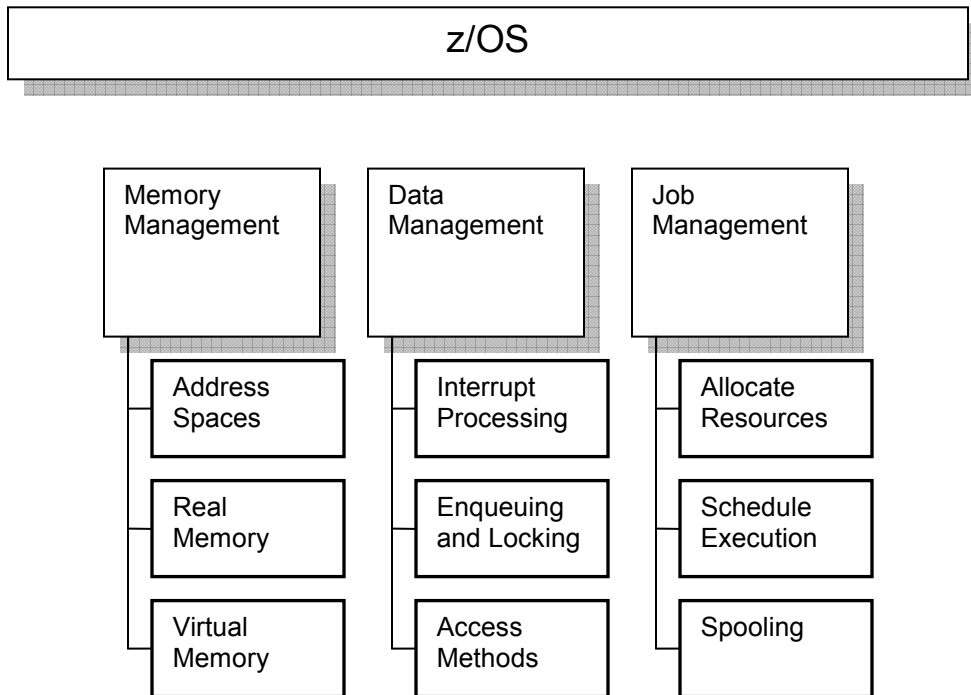
- z/OS operating system and resource management.
- The role and functions of JCL.
- How to code basic JCL statements.
- JCL syntax and naming rules.
- How to use JCL for running a utility program.
- How to interpret JCL listings.
- IEBGENER utility.
- Allocation and termination messages.
- Utility messages.

1 JCL: Job Control Language

JCL: Job Control Language controls the execution of jobs on an IBM mainframe operation. It provides instructions to the z/OS operating system for executing a job.

1.1 z/OS Operating System

The z/OS operating system is a set of programs which controls the activity of the computer. Its primary function is to manage the resources of the computer, such as memory, processor time, and I/O devices, such as printers and disk drives.



2 z/OS Resource Management

The z/OS operating system has a number of facilities and subsystems which it utilizes to manage the resources of a mainframe system. They are important when coding JCL.

Component	Explanation
Task	A task is the basic unit of work for a mainframe computer. A single program is referred to as a task while it is executing. In JCL, each step of a job corresponds to a task.
Address Space	An address space is an area of virtual storage which contains a program while it executes. Every batch job and TSO user has its own address space. The z/OS operating system ensures that the activities in each address space are kept separate from others in the system.
JES	<p>The Job Entry Subsystem, JES, is a part or subsystem of z/OS which:</p> <ul style="list-style-type: none"> • sets up the program in an address space. • controls slow I/O devices, such as printers. • processes all the JCL. <p>There are two varieties of JES: JES2 and JES3. For the most part, they function similarly. Unless otherwise noted, everything in this reference guide applies equally to both JES2 and JES3.</p>
Spool	<p>The spool is a special disk storage area used by JES. It is used to store records to and from slow I/O devices, such as printers.</p> <p>The spool enables a program to be processed more efficiently. As JES reads JCL from a reader, it stores the data on the spool. Later when the program executes, it accesses data from the spool as though it was receiving data the reader. Since reading from a disk is faster than reading from a card reader, the program finishes more quickly, freeing system resources for another program.</p> <p>The spool also provides flexibility for computer operations. Jobs are read and stored on the spool until an address space is available. A program will produce reports which are kept on the spool. Operators are then able to direct the reports to the appropriate printers.</p>
Access Method	<p>An access method is the software subsystem in the z/OS operating system which controls I/O: input/output for faster devices, such as disks and tapes. There are several access methods, one for each type of file organization.</p> <p>All access methods perform the same basic function. They take the READ or WRITE statements in a program and issue the appropriate machine commands to read or write to the disk or tape.</p>
Dataset	<p>The dataset is the z/OS unit for storing data. Datasets are associated with their physical medium. It is proper to speak of a disk dataset, a tape dataset, or even a print dataset.</p> <p>Programs view datasets as files. A program is not concerned with the physical storage of a dataset. It examines the organization of records, such as a sequential file or an indexed file.</p>

3 z/OS Job Processing

The z/OS operating system provides for two types of job processing:

Interactive	Batch
-------------	-------

Interactive processing is available on demand and provides real-time results. The user interacts with program execution by using TSO commands from a terminal.

Batch processing allows for delayed program execution. This provides for flexibility in scheduling program execution. It also enables a number of jobs to be processed concurrently.

JCL statements control batch processing. As a job flows through the system, there are four times that JCL effects the job.

1- Entering the System

Batch jobs enter the system from a variety of input devices. The data is usually read into a spool dataset or submitted from a TSO terminal.

Regardless as to where the job originates, JES immediately examines the JCL for syntax errors.

- If JES finds an error, it flushes the job with a JCL error listing.
- If there are no errors, JES stores the job on the spool until it is ready to execute.

2- Begin Processing

The job begins processing when JES finds an available address space. During initiation, JES moves the job from the spool into the address space and informs the z/OS operating system about the job.

Once the job has been initiated, z/OS performs allocation processing. During allocation, z/OS locates all the datasets and devices which the job needs in order to execute. After allocation, z/OS brings the program into memory and begins execution.

Under JES2, dataset allocation takes place at the beginning of each step.

Under JES3, all allocation takes place at the beginning of the job.

3- End Processing

When the program finishes, the z/OS operating system resumes control.

During deallocation, the z/OS operating system disposes of all the datasets used by the program. Based upon the instructions provided in the JCL, it either saves or deletes the datasets.

After deallocation comes termination. The job is taken from the address space in order that JES can use it for the next job.

4- Process Output

After termination is complete, JES will process the job's output. JES prints the JCL listing for the job and any reports which were generated by programs in the job. After all output has been printed, JES purges the job from the system.

All jobs follow the same process from beginning to end. JCL is used to control what goes on during each part of the process.

4 Job Control Language

JCL has its own terminology, syntax, and rules which govern its use.

4.1 JCL Statements

JCL consists of eight basic statements which define specific functions.

The most commonly used JCL statements are:

Statement	Explanation
JOB	The job statement defines a job and provides information about the job as a whole.
EXEC	The execute statement defines a job and provides information about the program to be executed.
DD	The data definition statement describes a dataset needed by the program.
//*	The comment statement permits descriptive comments to be included with the JCL.
/*	The delimiter statement ends or delimits an in-stream dataset.
//	The null statement ends a job.
PROC	The procedure statement begins a group of predefined JCL.
PEND	The procedure end statement ends a procedure.

The three control statements which comprise the majority of the JCL code are: JOB, EXEC, and DD.

4.2 JCL Syntax

JCL statements are made up of one or more of these fields:

//NAME OPERATOR PARAMETERS COMMENTS

Field	Explanation
//	The IDENTIFICATION field or slash-slash identifies a statement belonging to the job control language. All JCL statements, except the delimiter, must contain two slashes (//) in columns 1 and 2 of the statement. The first two columns of the delimiter must contain a slash asterisk (/*)
NAME	The NAME field identifies the control statement in order that other statements and system control blocks can refer to it. If a name is used, the NAME field must begin in column 3.
OPERATOR	The OPERATOR field specifies the type of control statement, such as JOB and DD. The OPERATOR field follows the name field and must be preceded and followed by at least one or more blank spaces.
PARAMETERS	The operand field contains PARAMETERS separated by commas. These PARAMETERS specify the action to be taken by the statement and comprise most of the JCL. The operand field must follow the OPERATOR field and be preceded and followed by at least one blank space.
COMMENTS	The COMMENTS field contains the additional information which will be helpful to information technology personnel who will be examining the JCL. It must follow the operand field and be preceded by at least one blank.

Control statement fields, except the NAME field which must begin in column 3, may be used in free form. Free form indicates that the fields do not need to begin in a particular column as long as they are separated with a blank.

4.3 Naming Rules

All names used in JCL must follow these rules:

- Contain from 1 through 8 characters.
- Begin with an:

Alphabetic character	A – Z
National character	@ # \$

- Other characters can contain only an:

Alphabetic character	A – Z
National character	@ # \$
Numeric character	0 – 9

Example:

Valid names which can be used include:

JOB398	FRED	\$1984	Z
--------	------	--------	---

Most z/OS installations will have a naming convention which will determine the structure of a name.

5 JOB Statement

The first JCL statement of a job is the JOB statement. A job consists of one or more programs that the z/OS operating system schedules for execution and executes in sequence. The JOB statement designates the start of a job. The null or // statement should only be used as the last JCL statement of job. This will indicate that no other JCL is to be processed. If the null statement is not present, the next JOB statement marks the end of the previous job and the beginning of the next job.

The JOB statement provides the z/OS operating system with a way of identifying a job. The name has to be defined to z/OS. Additional information is specified for the purpose of describing how a job should be classified and handled by z/OS: accounting information, scheduling directions, and printing.

A job consists of one or more steps. Each job step is defined by an EXEC or execute statement which identifies the program to be run.

5.1 JOB Statement Format

One Job -- One Step

```
// JOB           (Identifies the start of the job.)
// EXEC         (Defines the step and identifies the program.)
//             (Defines the end of job.)
```

One Job -- Multiple Steps

```
// JOB           (Identifies the start of the job.)
// EXEC         (Defines the first step.)
// EXEC         (Defines the second step.)
.
.
.
//             (Defines the end of job.)
```

Only use one JOB statement per job. Each JOB statement must have a jobname which adheres to the naming rules. Giving each a job a unique name will make it easier to keep track of jobs.

At most z/OS installations, the jobname will contain the TSO userid followed by one or more characters.

The operand field can have either positional or keyword parameters. Positional parameters must follow in a specific sequence or position on the JOB statement. Keyword parameters are designated by a keyword and an equal sign. They can be provided without regard to sequence.

Unless required by a z/OS installation, all JOB statement parameters are optional. If no parameters are coded, the COMMENTS field can not be included.

5.2 Accounting Information

The first positional parameter of the JOB statement allows accounting information to be supplied; the z/OS operating system does not need this parameter. A z/OS installation may require accounting information for the purpose of charging specific users for their jobs and keeping track of the system resources utilization.

Each installation determines the content of this parameter.

If the job accounting information includes an account number and the number does not contain special characters, then only the account number needs to be coded.

Example:

```
//SYSEDJOB JOB 12A75
```

The account number and each additional item of accounting information must be separated by commas. The job accounting information can be enclosed either in parentheses or apostrophes if more than one value is used.

Example:

```
//SYSEDJOB JOB (12A75,DEPTD58,706)
```

```
//SYSEDJOB JOB '12A75,DEPTD58,706'
```

5.3 Programmer Name

The second positional parameter on the JOB statement indicates the name or group responsible for a job. The parameter is positional and must follow the accounting information. The z/OS operating system does not use the programmer name parameter. The operators or production control staff use it to deliver listings to the appropriate person.

The programmer name parameter cannot exceed 20 characters. If special characters are included other than hyphens, or leading or imbedded periods, enclose the name in apostrophes.

Example:

```
//SYSEDJOB JOB 12A75,'M. HARRIS'
```

If accounting information is not coded, a leading comma is necessary. This informs z/OS that the first positional parameter is missing.

Example:

```
//SYSEDJOB JOB , 'M. HARRIS'
```

5.4 CLASS Parameter

The CLASS is an optional keyword parameter. It assigns the job to an input job class. JES uses this job class to determine the order in which jobs are scheduled for execution.

The CLASS parameter is coded in this format:

```
CLASS=job-class
```

Any character A-Z or 1-9 defined by an installation can be utilized.

Example:

```
//SYSEDJOB JOB , 'M. HARRIS', CLASS=T
```

If a class is not specified, the system uses a default based on where the job was submitted from: workstation, time-sharing user, etc.

Each installation defines input classes based on the scheduling requirements.

Example:

Input Class	Role
A	Regular; this is the default.
N	Overnight turnaround.
P	Production.
Q	Special setup requirements.
T	Tests.

The installation standards manual should be used to select the job class appropriate to the characteristics of a job.

5.5 MSGCLASS Parameter

The MSGCLASS parameter specifies the output class to which the job listing is written. The MSGCLASS parameter is coded in the following format:

```
MSGCLASS=output-class
```

Output classes may be A-Z and 0-9.

Example:

```
//SYSEDJOB JOB ,GEORGE,MSGCLASS=A
```

As with the input classes, each installation defines output classes which provide for a variety of output requirements:

Output Class	Role
A	Standard forms; this is the default.
B	Punch.
L	Laser printer.
P	Programmer test.
T	TSO.
Z	Discard.

As with the CLASS parameter, the MSGCLASS has a default specified by an installation.

5.6 Continuing JCL Statements

Keyword parameters can be coded in any order. They can be coded on a single line as long as they do not extend past column 72.

Example:

```
//NIGHT318 JOB ACCT12,MIKE,CLASS=N,MSGCLASS=P
```

Or they can be coded on separate lines.

Example:

```
//NIGHT318 JOB ACCT12,MIKE,  
//      CLASS=N,  
//      MSGCLASS=P
```

Coding parameters on separate lines makes JCL easier to read and interpret. When continuing a statement, ensure that each line, except the last, ends in a comma. Parameters on the continuation lines must begin between columns 4 and 16.

6 EXEC Statement - Execute

Each step in a job is described by an EXEC statement. It instructs the system which program to execute. It may also contain other information associated with that job step.

An EXEC statement must be coded for each job step. A single job can have a maximum of 255 steps.

6.1 EXEC Statement Format

The format of the EXEC statement is:

```
//stepname EXEC positional,keyword comments
```

The stepname is a name given to an EXEC statement which identifies the step within a job. Although a stepname is optional, it is the recommended practice to include it when a job has more than one step. Having a stepname will make it easier to interpret output listings for a job.

When coding a stepname, follow the standard naming rules. Stepnames should be unique within a job.

6.2 PGM Parameter - Program

The PGM parameter is a positional parameter that specifies a program to be executed. A PGM parameter needs to be used in order to instruct the system which program to execute for this step.

Even though it contains a keyword, the PGM parameter is positional and should be the first parameter on an EXEC statement.

The format of the PGM parameter is:

```
PGM=parameter name
```

Example:

The program name is the name of the load module or program, which is to be executed.

```
//STEP4 EXEC PGM=UPDATE
```

6.3 ACCT Parameter - Account

Some installations keep track of accounting information for job steps instead of the entire job. The ACCT parameter provides the capability to supply accounting information for the job step.

The ACCT parameter on the EXEC statement specifies the job step accounting information:

ACCT=accounting information

Accounting information is defined by the installation.

Example:

```
//STEP2 EXEC PGM=TESTPAY,  
//      ACCT=(3862,'T/24',#AV)
```

Accounting information for the step can be provided instead of or in addition to the job. If the step accounting information is used in addition to job accounting information, the value specified for ACCT overrides the job information for that step only. The value specified for the job applies to all other steps in the job.

A supervisor will make the decision whether step accounting information will need to be supplied.

7 DD Statement

All programs use files. Files are read, created, or updated as a program executes. The DD or Data Definition statements are used for describing each file that a program uses.

The DD statement is used by the operating system to locate the file. "Where is the input located?" and "Where should it put the output?" are questions which DD statements answer.

7.1 DD Statement Format

DD statements are associated with a specific program or job step. Since an EXEC statement begins a JOB step, the DD statements for one step must be included before the next EXEC statement or the end of the job.

The format of a DD statement is:

```
//ddname DD positional,keyword comments
```

7.2 ddname Field

The ddname identifies a DD statement. It follows the standard JCL naming rules. The ddname should be unique within a step, but can be repeated in other steps.

The ddname is the link between a program and the operating system. Each programming language has its own method of identifying the files a program uses. However, all languages specify an "external name" or ddname.

The z/OS operating system uses this "external name" to associate the file used by a program with a dataset described by a DD statement. Although this two-step approach may appear to be inefficient; it provides flexibility in assigning files. By changing a DD parameter, it is possible to direct an output file to the printer, tape, or disk. Accordingly, it will not be necessary to change and recompile the program to redirect its output.

Example:

How different languages specify ddname to work with this JCL.

```
//PRTSTEP EXEC PGM=PRNTCARD  
//CARDIN DD ...  
//PRINTOUT DD ...
```

COBOL Programming Language

In COBOL, the names are defined in the Environment Division:

```
ENVIRONMENT DIVISION.
```

```
INPUT-OUTPUT SECTION.
```

```
FILE-CONTROL.
```

```
    SELECT CARD-FILE          ASSIGN TO CARDIN.  
    SELECT PRINTER-OUTPUT    ASSIGN TO PRINTOUT.
```

This COBOL program uses two files. The ASSIGN clause specifies the ddnames, CARDIN and PRINTOUT, which the z/OS operating system uses to locate the datasets for the program.

PL/1 Programming Language

In PL/1, specify the ddname as part of the OPEN statement.

```
    OPEN CARD-FILE(CARDIN)      INPUT;  
    OPEN PRINTER-OUTPUT(PRINTOUT) OUTPUT;
```

Assembler Programming Language

In Assembler, the ddname is an operand of the DCB macro.

The equivalent assembler statements are:

```
    CARDFILE DCB DDNAME=CARDIN ...  
    PRINTER  DCB DDNAME=PRINTOUT ...
```

FORTRAN Programming Language

The structure of FORTRAN limits the ddname options.

FORTRAN specifies I/O devices by a one or two-digit logical address as part of READ and WRITE statements.

```
    READ (5,100) N,X  
    WRITE (6,500) TOTL
```

The logical address is a portion of the ddname:

```
    //PRTSTEP EXEC PGM=PRNTCARD  
    //FT05F001 DD ...  
    //FT06F001 DD ...
```

8 Spooled Datasets

DD statements have many parameters which can be used for describing datasets on tape and disk. These parameters can be used with slow I/O devices like card readers and printers.

The z/OS operating system uses the spool to hold jobs awaiting execution and to control the printing of the JCL listing. It can also use the spool to store data files for a program.

Input data is read from the card reader and stored on the spool before the program executes. Later, as the program executes, it reads the file from the spool much faster than it could from a card reader.

Similarly, files to be printed or punched are also kept on the spool. These files are printed after the program finishes. This permits a program to write many reports at once; even though only one printer is available to print them.

On the newer z/OS systems, the location of datasets is determined by the SMS: Storage Management Services subsystem.

8.1 * Parameter

The * parameter describes an in-stream dataset. An in-stream dataset is a group of data records which are included in the job stream. A DD * statement precedes the data in order to differentiate it from the JCL.

The * parameter is positional and must immediately follow the DD operator:

```
//INPUT DD *
```

The data records for the file must immediately follow the DD * statement. All subsequent records are part of the data file until one of these three things occurs:

- A delimiter (/) is read.
- Another JCL (//) statement is read.
- The reader runs out of records.

Example:

This in-stream dataset is terminated by a delimiter statement.

```
//INPUT DD *  
data record 1  
data record 2  
.  
.  
.  
data record n  
/*  
//NEXT DD ...
```

By using a different ddname for each file, any number of in-stream datasets in a step can be used.

8.2 SYSOUT Parameter

The SYSOUT parameter assigns a dataset to be printed or punched to an output class.

The format is:

```
SYSOUT=class
```

The class can be an alphanumeric character (A-Z, 0-9) or *. The output classes set up at an installation will be the same as that with the MSGCLASS parameter on the JOB statement.

Example:

```
//REPORT DD SYSOUT=L
```

This directs the z/OS operating system to store the print file called REPORT in output class L. If the installation uses the output class definitions, REPORT will be directed to a laser printer.

Example:

```
SYSOUT=*
```

An asterisk (*) in the SYSOUT parameter directs the z/OS operating system to write the output to the same class specified for MSGCLASS on the JOB statement.

Using SYSOUT=* simplifies changing the output classes for the entire job. Only the MSGCLASS parameter on the JOB statement needs to be changed. Otherwise, it would be necessary to change each individual SYSOUT parameter.

Example:

```
//TESTJOB    JOB T22,WILSON,  
//          MSGCLASS=T  
//PRTEST    EXEC PGM=PRINTGEN  
//PRINT1    DD SYSOUT=F  
//PRINT2    DD SYSOUT=*  
//PRINT3    DD SYSOUT=*
```

The JCL listing and files PRINT2 and PRINT3 are all written to output class T, as specified by MSGCLASS. PRINT1 is specifically written to class F.

By changing the MSGCLASS from T to A, PRINT2 and PRINT3 are also changed to class A. PRINT1 still goes to class F.

8.3 DUMMY Parameter

In some instances there may be a requirement not to use a dataset. A program may have defined several input files, but only have data for a single file. Or there may be a requirement not to print a long printout created by the program. However, when a program executes, it expects to use all of its files.

The DUMMY parameter allows the program to execute, but not use a particular dataset. DUMMY is a positional parameter and must be the first parameter on the DD statement.

In order to test a program, without printing the output file, use the DUMMY parameter on the output DD statement.

Example:

```
//STEP1      EXEC PGM=LOADCDS  
//CARDIN     DD *  
             (input records)  
//PRINTOUT   DD DUMMY
```

The input file, CARDIN, will be read when the program LOADCDS executes; the PRINTOUT file will not be created.

9 Utility Programs

The z/OS operating system provides a number of utility programs to perform common tasks, such as copying files. These utilities are available on all systems.

Most utilities operate in a similar manner. This consistency makes them easy to learn and interpret.

The JCL statements common to most utilities are:

Statement	Explanation
EXEC	The statement which names the utility program.
SYSIN	The DD statement defining control statements for the utility.
SYSPRINT	The DD statement indicating where utility messages should be printed.
SYSUT1	The DD statement that points to the input dataset.
SYSUT2	The DD statement that points to the output dataset.

9.1 IEBGENER Utility

The IEBGENER utility is used for copying one dataset to another.

Example:

This JCL copies an in-stream dataset to the printer.

```
//COPY      EXEC PGM=IEBGENER
//SYSIN     DD DUMMY
//SYSPRINT  DD SYSOUT=*
//SYSUT1    DD *
            [data]
//SYSUT2    DD SYSOUT=*
```

The EXEC statement designates the program IEBGENER to be executed. IEBGENER copies the input defined in SYSUT1 DD statement to the SYSUT2 DD statement. In this case, the output goes to the SYSOUT class A.

No control statements are needed to do a straight copy, but IEBGENER still expects the SYSIN DD statement to be present. Accordingly, specify a DUMMY dataset for SYSIN.

IEBGENER produces messages indicating whether or not execution has been successful. These messages are written to the SYSPRINT DD statement, which is printed to the same SYSOUT class as the MSGCLASS parameter on the JOB statement link to the JOB card.

SYSUT1 describes the input dataset. The DD * indicates that the data records are part of the job stream and immediately follow.

10 JCL Listings

For every job that is run, a listing is produced showing what has transpired while the job was in the system.

JCL listings typically include:

- Separator sheet
- JES log
- JCL statements
- Allocation messages
- Termination messages
- Utility messages

Listings vary in format and will be specific to an installation.

10.1 Separator Sheet

This is a typical separator sheet.

JOB #6001A

Figure 1 - 1. Separator Page

10.2 JES Log

Example: JES Log

```
IAT6140 JOB ORIGIN FROM GROUP=ANYLOCAL, DSP=IR , DEVICE=INTRDR , 0000
08:27:20 ---- IAT6853 THE CURRENT DATE IS WEDNESDAY, 12 JAN 2011 ----
08:27:20 IAT2000 JOB #6001A (JOB44395) SELECTED SY52 SRVCLASS=BATCH
08:27:20 TSS7000I #6001 Last-Used 12 Jan 11 07:59 System=5100 Facility=TSO
08:27:20 TSS7001I Count=02139 Mode=Impl Locktime=None Name=INSTRUCTOR1
08:27:20 SSA001I #6001A JOBNO=44395 ASID=031E SYSTEM=SY52 SYSR1=X1C1T1
08:27:20 IEF403I #6001A - STARTED - TIME=08.27.20 DATE=01/12/2011.012
08:27:21 IEF404I #6001A - ENDED - TIME=08.27.21 DATE=01/12/2011.012
```

Figure 1 - 2. JES Job Log

The JES job log provides a summary of the job's activity in the system. The format of the log differs slightly between JES2 and JES3. However, both contain similar information.

Time stamps indicate the time the job enters the system, begins executing, and so forth. The log includes all messages sent to the operator about the job. The log includes statistics on the JES activity for the job.

10.3 JCL Statements

```
//#6001A JOB NOTIFY=#6001                                00010002
//*   FIRST STEP TO COPY A DATASET                      00011001
//STEP01 EXEC PGM=IEBGENER                              00020001
//SYSPRINT DD SYSOUT=*                                  00030001
//SYSIN DD *                                            00040001
//SYSUT1 DD DSN=#6001.BENEFIT.DATA,DISP=SHR            00050002
//SYSUT2 DD DSN=#6001.BENEFIT.BK,DISP=(,CATLG),        00060002
//                                                    00070002
  1 //#6001A JOB NOTIFY=#6001                          00010002
    //*   FIRST STEP TO COPY A DATASET                  00011001
    2 //STEP01 EXEC PGM=IEBGENER                        00020001
    3 //SYSPRINT DD SYSOUT=*                            00030001
    4 //SYSIN DD *                                      00040001
    5 //SYSUT1 DD DSN=#6001.BENEFIT.DATA,DISP=SHR      00050002
    6 //SYSUT2 DD DSN=#6001.BENEFIT.BK,DISP=(,CATLG),  00060002
    //                                                    00070002
      SPACE=(TRK,(5,1)),UNIT=SYSDA
```

Figure 1 - 3. JCL Statements

10.4 JCL Statement Listing

The printout lists the JCL statements submitted in the job. Each statement is numbered in the left column. If any errors have been found, they are listed at the bottom of the listing with the number of the appropriate statement.

10.5 Allocation and Termination Messages

This is an allocation and termination message.

```

TSS7000I #6001 Last-Used 12 Jan 11 07:59 System=5100 Facility=TSO
TSS7001I Count=02139 Mode=Impl Locktime=None Name=INSTRUCTOR1
IEF236I ALLOC. FOR #6001A STEP01
IEF237I JES3 ALLOCATED TO SYSPRINT
IEF237I JES3 ALLOCATED TO SYSIN
IGD103I SMS ALLOCATED TO DDNAME SYSUT1
IGD101I SMS ALLOCATED TO DDNAME (SYSUT2 )
        DSN (#6001.BENEFIT.BK                )
        STORCLAS (WRKSTOR) MGMTCLAS (MCWRKSTD) DATACLAS (          )
        VOL SER NOS= WRK009
IEF142I #6001A STEP01 - STEP WAS EXECUTED - COND CODE 0000
IEF285I  #6001.#6001A.JOB44395.D000000A.?          SYSOUT
IEF285I  #6001.#6001A.JOB44395.D0000009.?          SYSIN
IGD104I #6001.BENEFIT.DATA                          RETAINED, DDNAME=SYSUT1
IGD104I #6001.BENEFIT.BK                            RETAINED, DDNAME=SYSUT2
IEF373I STEP/STEP01 /START 2011012.0827
IEF374I STEP/STEP01 /STOP 2011012.0827 CPU      OMIN 00.01SEC SRB      OMIN 00.00SEC VIRT  420K SYS
268K EXT      12K SYS      9852K
IEF375I JOB/#6001A /START 2011012.0827
IEF376I JOB/#6001A /STOP 2011012.0827 CPU      OMIN 00.01SEC SRB      OMIN 00.00SEC

```

Figure 1-4. Allocation and Termination Messages

Allocation messages for each step show the location of the datasets needed by that step. There is a message for each DD statement in the step. There also may be messages for other datasets the system needs to run the program.

Termination messages are produced at the end of each step. The condition code indicates whether or not the program completed successfully. Messages present the disposition of all the datasets allocated for that step. Termination messages present the system resources, such as virtual storage and processor time, used by the step.

At the end of the job, additional messages provide similar information on the job as a whole.

10.6 Utility Messages

Some utility programs produce messages which also appear in the listing. This message has been produced by the IEBGENER utility.

```
DATA SET UTILITY GENERATE
```

```
PAGE 0001
```

```
PROCESSING ENDED AT EOD
```

Figure 1-5. IEBGENER Utility Messages

Messages generated by IBM utility programs begin with a line identifying the utility program. Additional messages follow which provide information on the execution of the program.

"PROCESSING ENDED AT EOD", indicates that IEBGENER has reached the end of the input data.