Chapter 22: Job Control Language

This chapter presents a discussion of **JCL** (Job Control Language) as used for jobs run on a modern IBM mainframe running a descendant of the OS operating system, such as z/OS.

First, we must define the term "job". A job is a unit of work for the computer to execute. The job comprises identification statements, control statements, possibly program text, and usually data. There are conventions to label the statements that are not program text and data so that the Control Program part of the Operating System can determine which is what.

The paradigm for a job is a sequence of cards, each card with one statement. The standard card type is the IBM 80 column card, an example of which is shown below. The use of these cards persisted well into the time at which programs and data could be entered through a computer terminal. Today, in classes associated with this textbook, we skip the card input and create text files for submission as jobs. Just remember that each line of text should be imagined as being the content of an 80 column card.

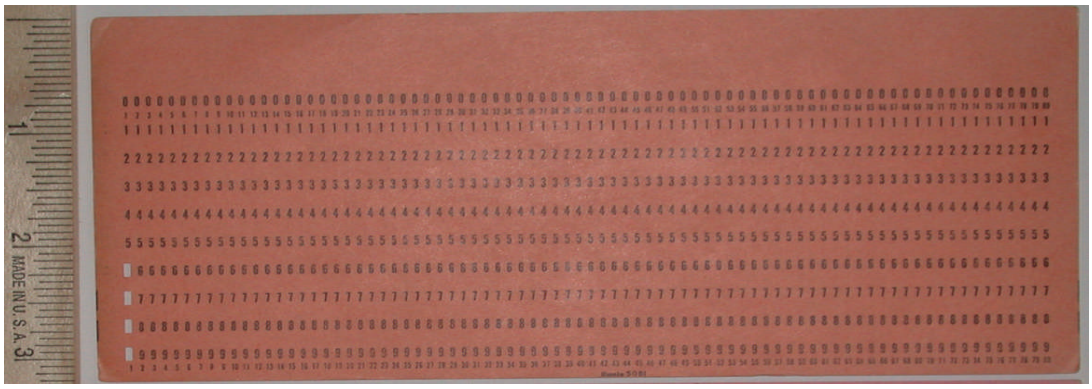


Photo by the author of a card in his collection

The card pictured above is a "6/7/8/9 card" used on a CDC–7600. This was a control card used to indicate the end of a specific job. In modern terms, a "7/8/9 card" would have been an EOD (End of Data) and a "6/7/8/9 card" an EOF (End of File). The 7/8/9 cards were green and the 6/7/8/9 cards were orange; this as a convenience to the programmer. The only computer–readable data on any card is found in the pattern of column punches.

The transition from card input of jobs to other means was driven by the simple inconvenience of handling boxes containing hundreds of cards. The key feature that facilitated that change was the introduction of system disk drives big enough to store significant amounts of user programs and data. This change was not driven by hardware only; it was some time after the introduction of disk drives that the software designers were able to develop a stable operating system based on the use of such drives. Your instructor recalls using a Xerox Sigma–7 while at Vanderbilt University in the 1970's, a time during which the operating system was being debugged. We called the machine the "Yo–Yo 7" as it was constantly down and up.

The first step in transitioning from card input was the ability to catalog a card deck on a disk file maintained by the computer center. Though the jobs remained card based, they became very short: access this file, change these statements, add these statements, and then run. Soon thereafter, the cards went away.

Copyright © 2009 by Edward L. Bosworth, Ph.D.

Next, it is important to dispel a misunderstanding that would be almost comical, had it not actually occurred during the teaching of a course based on this textbook. We begin by considering the first few lines of a program that your author assigns as a first lab.

```
//KC02263R JOB (KC02263),'ED BOZ',REGION=3M,CLASS=A,MSGCLASS=H,
// NOTIFY=KC02263,MSGLEVEL=(1,1)
//*
//*
//*
//FFFPROC JCLLIB ORDER=(TSOEFFF.STUDENT.PROCLIB.ASM)
//JESDS    OUTPUT PAGEDEF=V06483,JESDS=ALL,DEFAULT=Y,CHARS=GT15
//STEP1    EXEC PROC=HLLASM
//ASM.SYSIN   DD *
```

The above text is the block of job control language that precedes the text of the first assembler language program. Note that many of the lines begin with "**//**". Several students decided that these mandatory lines were optional, since they were obviously comments.

The structure of a comment in either a programming language or an execution control statement depends on the language or operating system. It is peculiar to that system. The fact that the "**//**" character sequence introduces in–line comments in both C++ and Java does not imply similar functioning in other situations. In IBM Job Control Language, comments are prefixed by "**//***", with the asterisk being very significant.

The Job Control Language

There are six types of job control statements that will interest us at this time. These are:

JOB	This marks the beginning of a job. It gives the user identification, accounting information, and other site–specific data.
EXEC	This marks the beginning of a job step by specifying a program or procedure to be executed.
DD	This request the allocation of an I/O device and describes the data set on that device. It must use the logical device name from the program.
//*	This is a comment in the job control language.
/*	This terminates an input stream data set.
//	This can be used to mark the end of a job.

Logical and Physical Devices

One of the advantages of the structure of the JCL is the ability to define a logical device using a DCB macro within the code, and use the DD control statement to link that logical device to an actual physical device. With the DCB, the code specifies the logical properties of a device. For example a logical printer might be described as PS (Physical Sequential) with record length of 133 bytes (one control character and 132 characters to be printed). The DD statement might then associate this logical device either with the standard output stream or with a dedicated disk file that can be saved and accessed by another job.

Much of this is discussed in chapters 5 and 6 of the IBM Redbook ***Introduction to the New Mainframe: z/OS Basics*** [R_24].

Copyright © 2009 by Edward L. Bosworth, Ph.D.

The Job Card
This identifies the beginning of a job. It must include a name to associate with the job. For use in our classes, that name is most often the user ID. The name must begin in column 3 of the "card", following the "**//**" characters. Remember that none of this is free–form input.

In general, the format of the JOB statement starts as follows.
```
//name JOB (account number),programmer name
```

Consider our example from the listing of a lab exercise.
```
//KC02263R JOB (KC02263),'ED BOZ',REGION=3M,CLASS=A,MSGCLASS=H,
```

The user name consists of from one to eight alphanumeric characters, with the first one being alphabetic. The standard for our course is the user ID with a single letter following it. The job card above shows a user ID of **KC02263**, with the letter "R" appended.

The next entry in this statement is the keyword "**JOB**" identifying this as a JOB card.

This is followed by the account number in parentheses. For our student use, the account number is the same as the user ID. This is followed by the programmer name, which is enclosed in quotes as the name contains a space.

The next entry, **REGION=3M**, specifies the amount of memory space in megabytes required by the step. This could have been specified by **REGION=3072K**, indicating the same allocation of space. The two size options here are obviously "**K**" and "**M**" [R_25, page 16–4].

The entry **CLASS=A** assigns a job to a class, roughly equivalent to a run–time priority. According to R_25 [page 20-15] the "class you should request depends on the characteristics of the job and your installation's rules for assigning classes". This assignment works.

The entry **MSGCLASS=H** assigns the job log to an output class [R_25, page 20-24]. Depending on the **MSGLEVEL** statement (see below), the job log will have various content.

The next line of text in the above example should be considered as a continuation of the job card, in that the information that is found there could have been on the job card.

```
// NOTIFY=KC02263,MSGLEVEL=(1,1)
```

The notify line indicates what user is to be given information about the execution of the job; the level of information is indicated by the integers associated with **MSGLEVEL**. The first number specifies which job control statements are to be printed in the listing. There are three possible choices.

0 Only the JOB statement is displayed. This is the default for many centers.

1. All job control statements are displayed including those generated from a cataloged procedure. This is the default for a student job. Note that a **cataloged procedure** is a sequence of control statements that have been given a name and placed in a library of cataloged procedures.
2. Only those job control statements appearing in the input stream are displayed.

The second number inside the parentheses specifies whether or not the I/O device allocation messages are to be printed. A 1 (the default) indicates that all allocation and termination messages are to be printed, regardless of how the job terminates.

Copyright © 2009 by Edward L. Bosworth, Ph.D.

The EXEC Statement
The execute statement begins a job step that is associated with the program name or procedure name that controls that step. Each EXEC can begin with an optional step name, which must begin in column 3 and be unique within the job.

There are three standard forms of the execute statement.

```
//step name   EXEC PGM=program name
//step name   EXEC PROC=procedure name
//step name   EXEC procedure name
```

The step name is optional, but if it exists it must be unique in the job. For example, we have this line in the job control language of our first lab assignment. This calls for the H–level assembler to be invoked. The procedure takes care of a number of steps that are required, and can be mechanically created.

```
//STEP1    EXEC PROC=HLLASM
```

In some more advanced JCL, there is a control logic that requires step names. In this example, we assign names just to show that we can do that.

The PGM option is rarely used by students, who commonly use cataloged procedures. This author views stored procedures in the same light as programming macros; they are predefined sets of statements that have proven useful in the past.

The second and third lines are equivalent, indicating that the default is to execute a cataloged procedure. This expands into a sequence of program EXEC and DD statements.

Here is an example of the ASMFC cataloged procedure [R_09, page 384]. This is given without explanation in order to show the expansion of a very simple cataloged procedure.

```
//ASM       EXEC PGM=IEUASM,REGION=50K
//SYSLIB    DD   DSNAME=SYS1.MACLIB,DISP=SHR
//SYSUT1    DD   DSNAME=&SYSUT1,UNIT=SYSSQ,SPACE=(1700,(400,50)),     X
//               SEP=(SYSLIB)
//SYSUT2    DD   DSNAME=&SYSUT2,UNIT=SYSSQ,SPACE=(1700,(400,50))
//SYSUT3    DD   DSNAME=&SYSUT3,SPACE=(1700,(400,50)),                X
//               UNIT=(SYSSQ,SEP=(SYSUT2,SYSUT1,SYSLIB))
//SYSPRINT  DD   SYSOUT=A
//SYSPPUNCH DD   SYSOUT=B
```

There are a number of parameters to the EXEC statement, but none of these need concern us here. The student who is interested is referred to [R_25, Chapter 16].

The DD (Data Definition) Statement
Any data sets used by the program must be described in DD statements. These must follow the EXEC statement for the particular step in which the data sets are accessed. In the lab examples used with the course associated with this textbook, the DD statements follow the assembler procedure invocation and its associated program input.

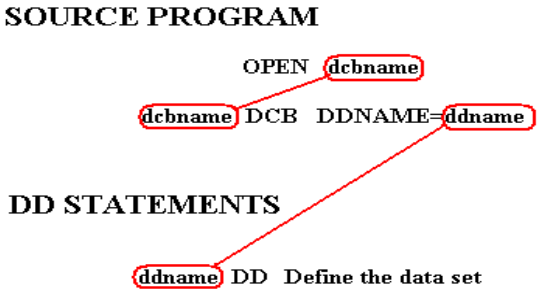
For more information, the reader should consult Chapter 6 of ***Introduction to the New Mainframe*** [R_24] or Chapter 12 of the ***MVS JCL Reference*** [R-25].

Copyright © 2009 by Edward L. Bosworth, Ph.D.

The general format of the DD statement is rather flexible, but all have this form.

```
//proc.ddname   DD    options
```

The first part of the name is the procedure step. In our programs, we use "GO" for this. The second part of the name is identical to that used in the DCB macro in the source program, and it further describes the data set referenced in that macro. In general, we have the following sets of relationships within the job.



Here is an example of the linkage between DCB and DD as found in our lab 1.

```
FILEIN   DCB   DDNAME=FILEIN,                                    X
               DSORG=PS,                                        X
               DEVD=DA,                                         X
               RECFM=FB,                                        X
               LRECL=80,                                        X
               EODAD=A90END,                                    X
               MACRF=(GM)
PRINTER  DCB   DDNAME=PRINTER,                                  X
               DSORG=PS,                                        X
               DEVD=DA,                                         X
               MACRF=(PM),                                      X
               LRECL=133,                                       X
               RECFM=FM
         END
/*
//GO.PRINTER DD SYSOUT=*
//GO.FILEIN DD *
```

What we have in the above example is a use of the standard input and output data streams. The input stream data set is simply the stream that includes the text of the program and the job control language. The "**DD *** " indicates that the stream is to be taken as the sequence of 80–character lines immediately following. This stream ends with "**/***".

The following represents the last lines in a job intended to print out the text of three lines. Note the three lines of input text immediately following the DD.

```
//GO.FILEIN DD *
LINE 1
LINE 2
LINE 3
/*
```

The statement "DD SYSOUT=*" indicates that the output associated with the ddname PRINTER is to be routed to the standard output stream, called SYSOUT.

Copyright © 2009 by Edward L. Bosworth, Ph.D.

The flexibility of this linkage between the DCB and DD statements is illustrated in the following fragment, taken from another lab exercise associated with this textbook. We have taken the above and changed only the DD statement. We have as follows:

```
PRINTER  DCB   DDNAME=PRINTER,                                        X
               DSORG=PS,                                             X
               DEVD=DA,                                              X
               MACRF=(PM),                                           X
               LRECL=133,                                            X
               RECFM=FM
         END
/*
//GO PRINTER DD DSN=KC02263.SP2008.LAB10UT,SPACE=(TRK,(1,1),RLSE),
//           DISP=(NEW,CATLG,DELETE)
//GO.FILEIN DD *
LINE 1
LINE 2
LINE 3
/*
```

The print output is now saved as a text file, called **SP2008.LAB1OUT** in the user area associated with the user **KC02263**, which was at the time your author's user ID. Neither the name "SP2008" nor the name "LAB1OUT" can exceed eight characters in length.

In this version of the **DD** statement, we use the **DSNAME** operand, abbreviated as **DSN**. This identifies the data set (disk file) name to be associated with the output and specifies a few options. The two we use are the disposition option and the space allocation option.

The data set disposition operand has the general form as follows.

DISP=(file status, normal disposition, error disposition)

The first terms indicates the status of the data set in relation to this job step. The options are:

OLD	An existing sat set is used as input only to this step.
SHR	An existing disk data set that can be shared with other jobs concurrently.
MOD	A partially completed sequential data set. New records to added at the end.
NEW	A new output data set is to be created for this job step.

The second term indicates the disposition of the data set in case of a normal termination of the process associated with the step. There are five options for this one.

KEEP	Keep the data set.
PASS	Pass the data set to a later job step.
DELETE	Delete this existing data set.
CATLG	Catalog and keep the data set.
UNCATLG	Remove this data set from the catalog, but keep it.

The third term specifies disposition in the case of an abnormal termination. The option PASS is not available, as it is presumed that an abnormal termination will be associated with corrupt data. Note that our JCL says **DISP=(NEW,CATLG,DELETE)**, indicating to create a new file and catalog it if the job terminates normally. If the job has an abnormal termination, just discard the file.

Copyright © 2009 by Edward L. Bosworth, Ph.D.

The space operand has the following format. It is used only for DASD (Direct Access Storage Device, read "disk") data sets.

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

The units term indicates the measure of storage space to be used. In order to understand this, one should review the architecture of a typical disk unit. The two options for this term are **`CYL`** (cylinder) and **`TRK`** (track). Our JCL has the option **`SPACE=(TRK,(1,1),RLSE)`**, indicating that one track is to be allocated initially for our data set and that additional disk space is to be allocated one track at a time when the existing allocation is exhausted.

The **`RLSE`** option indicates that the unused space on the DASD (disk drive) is to be released and made available for data storage by other programs when this program terminates and the data set is closed. [R_25, page 12–12].

One option worth mention just for historical reasons is the LABEL option. This was used when accessing data sets on magnetic tape, either 7–track or 9–track. The label was an identifier assigned to an individual physical tape. It was physically written on the label of the tape (to be read by the computer operator) and written in the header record of the tape (to be read by the Operating System). This option would insure that the correct tape was mounted, so that the desired data (and not some other) would be processed.

The reader who is interested in tape labels is referred to a few references, including [R_02, page 449; R_24, page 203, and R_25, chapter 12].

Copyright © 2009 by Edward L. Bosworth, Ph.D.