JCL Coding Requirements

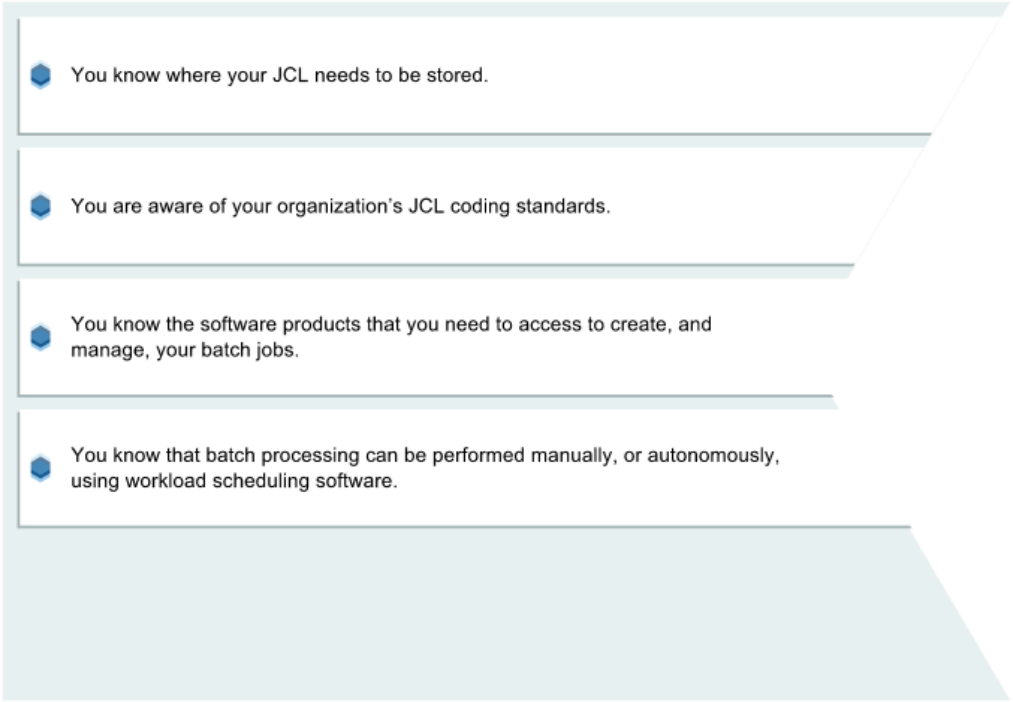
Objectives

JCL Coding Requirements

Before beginning to write any JCL code, you need to become familiar with the formatting of JCL statements, and the consequences should these not be followed. In this module you will look at the syntax, and format of JCL statements generally and will need to combine these items with your own organizational JCL standards.

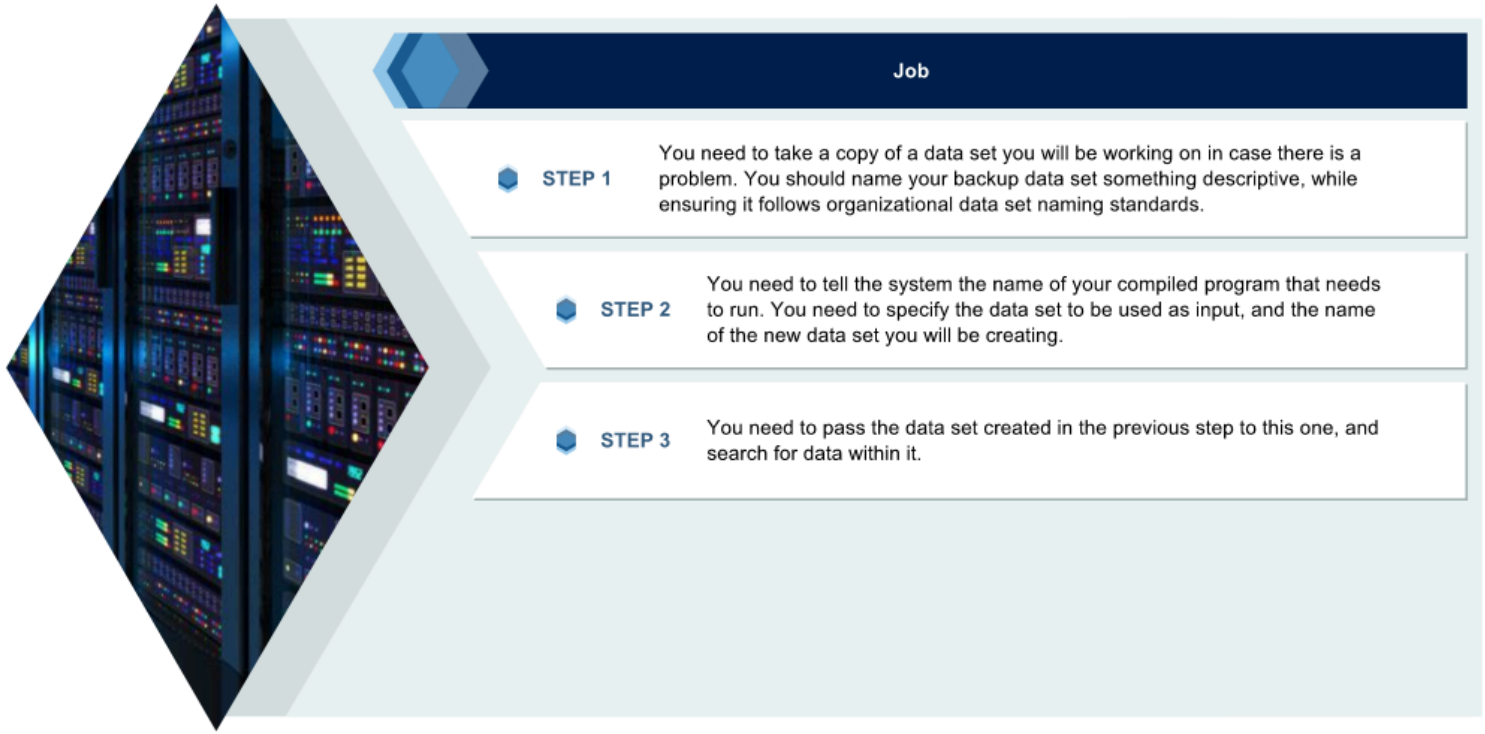
At the end of this module, you should be able to:

- Describe the Formatting Requirements of a JCL Statement
- Identify Common JCL Coding Errors



In the previous module you saw that there were several items you needed to think about before even laying your eyes on JCL.

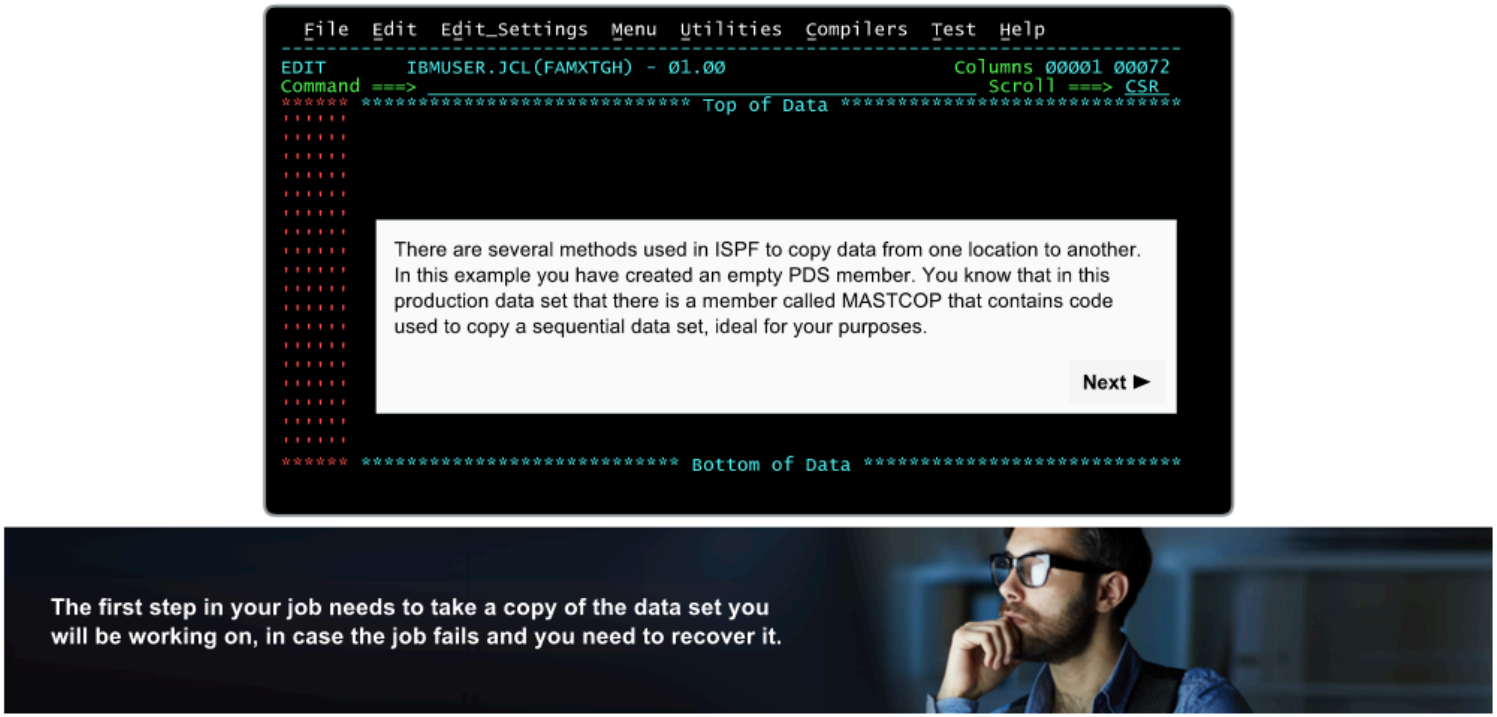
Now that these have been addressed, it is time to take a high-level look at the building blocks that form the structure of JCL.



Before writing any JCL you need to have a plan of the batch processing you need to perform. For example, you may need to do the following:

- Take a backup of a data set before you work on it
- Run a COBOL program you have been working on, against the data set you copied
- Create a report that identifies discrepancies between the result of your program, and another set of data

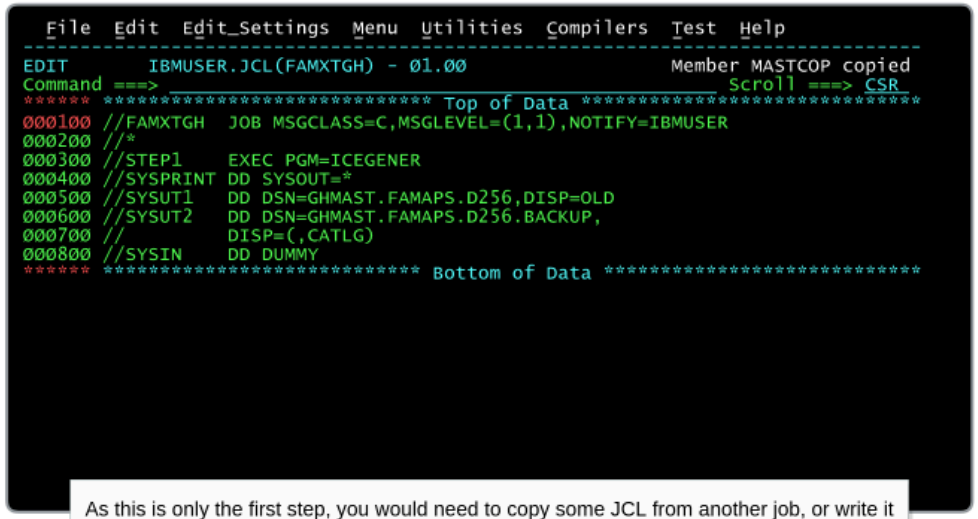
All these tasks can exist within a single batch job and are broken up into steps.



While you could write all the JCL required for the job on the previous page from scratch, the reality is that you are more likely to copy JCL from a similar job, or part of one, into a new PDS mem and make general changes before submitting it. Your vendor may also supply sample libraries containing basic JCL for tasks you may need to perform.

Why do you need to know all this information about JCL code? The answer is that you need to understand the instructions you are passing to the system, especially when jobs fail, and you nee diagnose system messages.

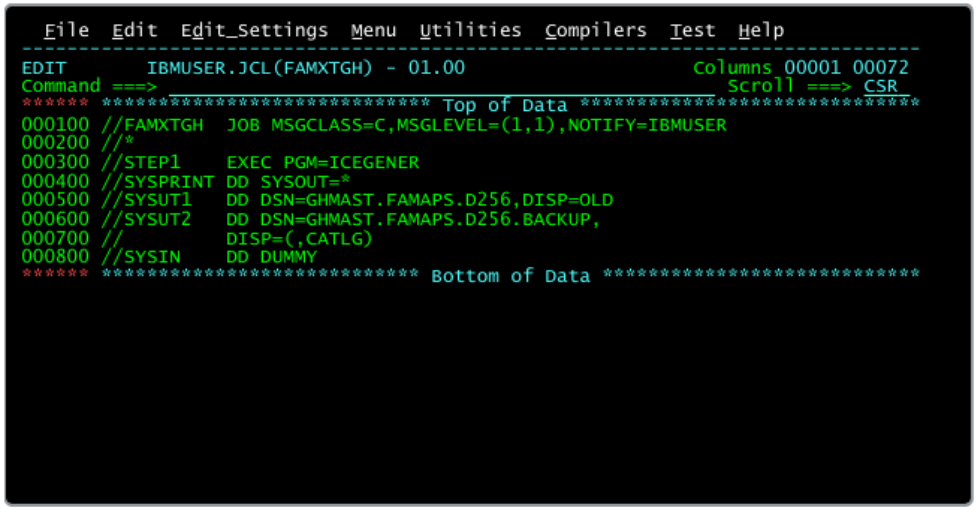
Click **Next** to see how JCL is copied into this empty PDS member and modified.



While you could write all the JCL required for the job on the previous page from scratch, the reality is that you are more likely to copy JCL from a similar job, or part of one, into a new PDS meml and make general changes before submitting it. Your vendor may also supply sample libraries containing basic JCL for tasks you may need to perform.

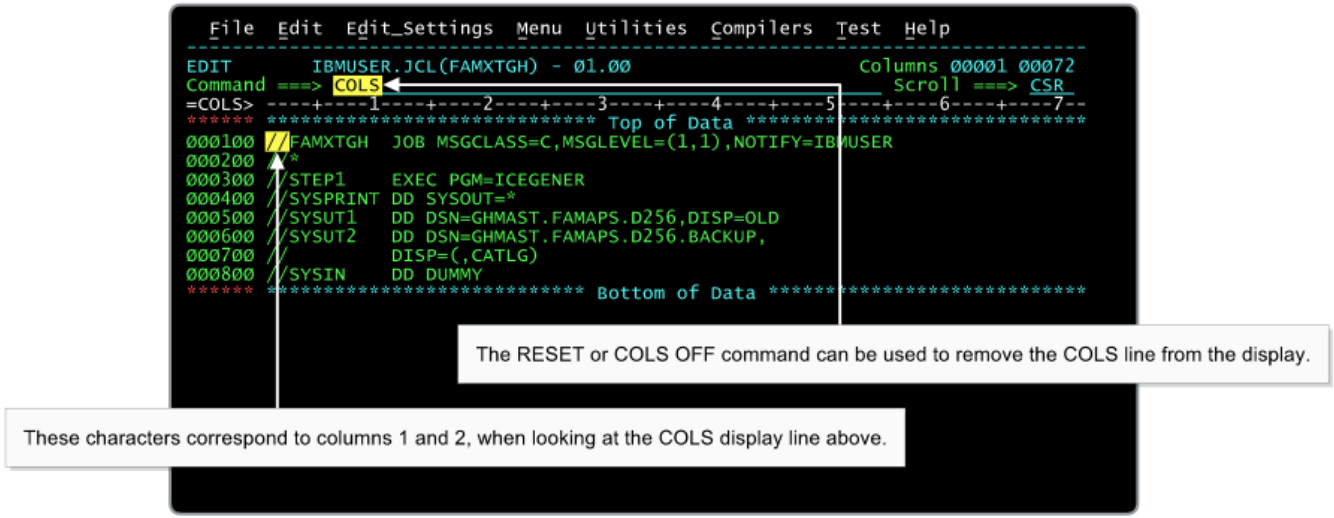
Why do you need to know all this information about JCL code? The answer is that you need to understand the instructions you are passing to the system, especially when jobs fail, and you neec diagnose system messages.

Click **Next** to see how JCL is copied into this empty PDS member and modified.



The JCL from the previous page is a simple example, where there is only a single step running a copy program.

While this JCL may look confusing to someone who has never worked with this code before, by the end of these JCL courses you will be able to confidently identify the purpose of these statem



Now you will begin dissecting the JCL to see what is required.

JCL statements begin with double slash (//) characters appearing in columns one and two. This is how the system is able to interpret the data that you submit to the system is JCL. There are a f JCL rules that dictate the column in which information can be placed, therefore it is usually a good idea to enter the ISPF edit command COLS in the command line, to display this information at top of your data.

Note that having said that JCL statements must begin with double slash (//) characters, there are some exceptions, but these are discussed in later courses.

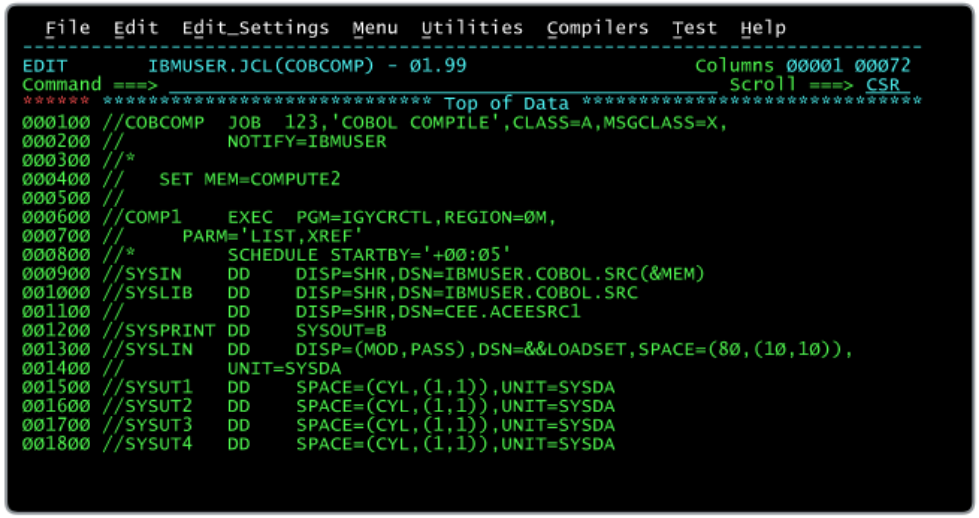
```
  File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
------------------------------------------------------------------------------
EDIT       IBMUSER.JCL(AGENER) - 01.05                  Columns 00001 00072
Command ===> ____________________________________________ Scroll ===> CSR
****** ***************************** Top of Data ******************************
000100 //AGENER   JOB MSGCLASS=C,MSGLEVEL=(1,1)
000200 //*  STEP 1 - TEST GDG BASED CONCATENATION
000300 //STEP1    EXEC PGM=IEBGENER
000400 //SYSPRINT DD SYSOUT=*
000500 SYSUT1   DD DSN=IBMUSER.PEN.DAILY.TRANS(0),DISP=SHR
000600 //SYSUT2   DD SYSOUT=*
000700 //SYSIN    DD DUMMY
000800 //         IF STEP1.ABEND=TRUE THEN
```

```
  Display  Filter  View  Print  Options  Search  Help
------------------------------------------------------------------------------
SDSF OUTPUT DISPLAY AGENER   JOB09308  DSID     2 LINE 19      COLUMNS 02- 81
COMMAND INPUT ===>                                           SCROLL ===> CSR
        12 SYSOUT SPOOL KBYTES
      0.00 MINUTES EXECUTION TIME
     1 //AGENER   JOB MSGCLASS=C,MSGLEVEL=(1,1)
       //*  STEP 1 - TEST GDG BASED CONCATENATION
     2 //STEP1    EXEC PGM=IEBGENER
     3 //SYSPRINT DD SYSOUT=*
     4 //SYSIN     DD *                GENERATED STATEMENT
     5 //SYSUT2   DD SYSOUT=*
     6 //SYSIN    DD DUMMY
     7 //         IF STEP1.ABEND=TRUE THEN
```

The message you receive as a result of forgetting to code // will be slightly different depending on where it is located. In this example, there are no // characters on line 000500, and when this is submitted, the system thinks that you are attempting to pass raw data to the system, and it has generated a //SYSIN DD * statement.

◄ Previous Restart

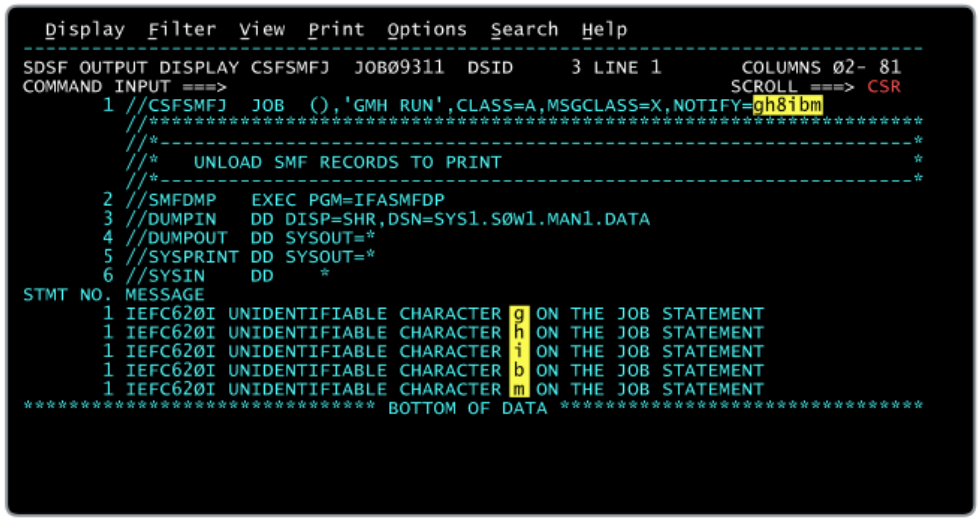
Writing JCL for the first time, it can be easy to forget to code the double slash (//) characters. In the example here, // is missing from the start of line 000100.



A common mistake for new JCL users is to submit a job that contains a line where the only data is //. This type of statement is called a null statement and indicates that this is the end of your JC

You can see in the example here that the user wanted to remove the content from a JCL statement, however left // by itself on line 000500. Even though JCL statements appear after this line, th system will not recognize them.

Click Play to see how the system interprets this code.

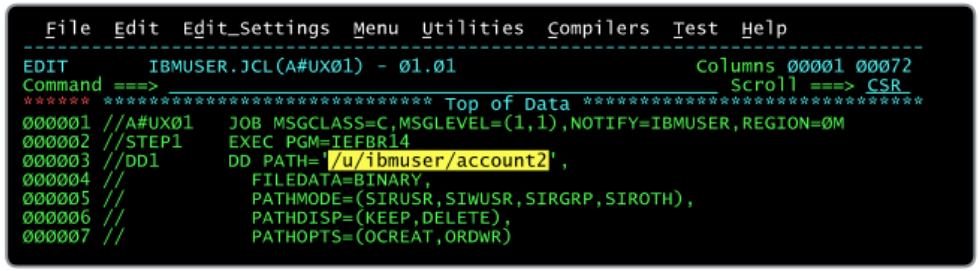


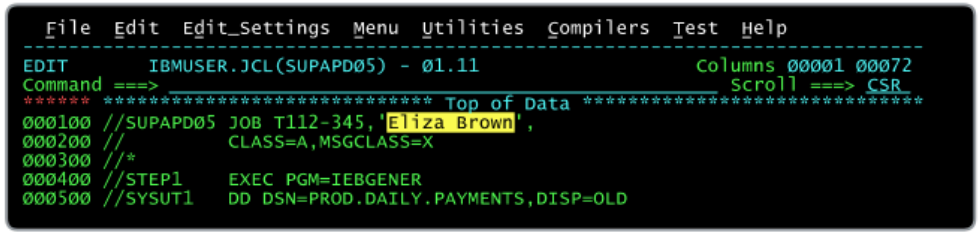
You can see the messages at the bottom of this job output, indicating that the system could not interpret the lowercase characters.

Another thing you will notice with JCL is that it is coded in uppercase characters. In fact, your edit profile - displayable using the PROF command in the command line - may show that CAPS is to ON, meaning that any data you enter is automatically converted to uppercase when you press the Enter key.

In the example shown here, the profile shows as CAPS OFF allowing lowercase characters to be accepted. The person updating this job, accidentally typed in their own user ID at the end of lin 000001 in lowercase characters.

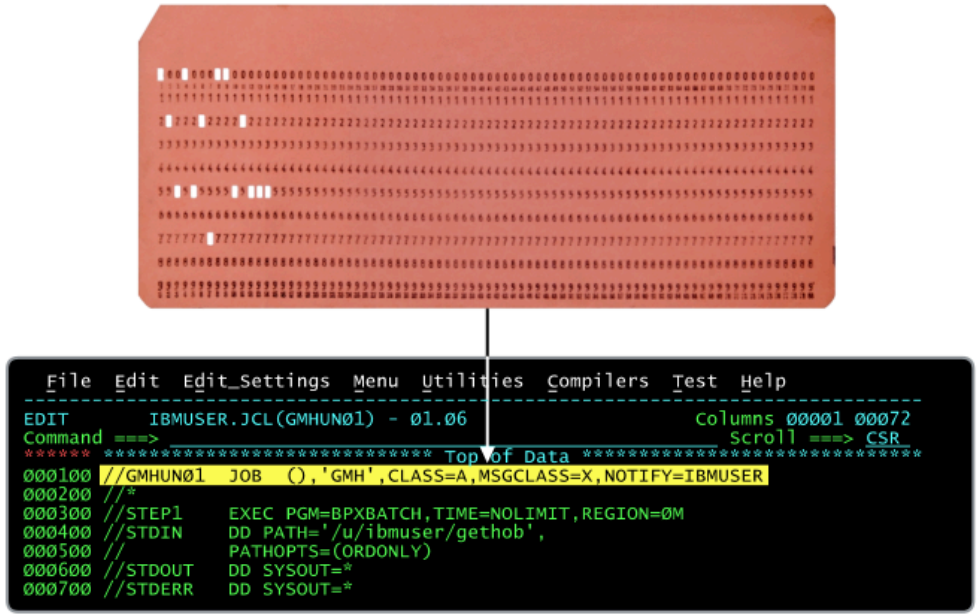
Click **Play** to see what happens when this job is submitted for processing.





As you will see throughout these courses there will be some exceptions to many of the JCL rules. In relation to the use of case in JCL, there may be situations where lowercase characters are required. To facilitate this, you should change your profile to CAPS OFF and ensure that any lowercase characters are enclosed in single quotes.

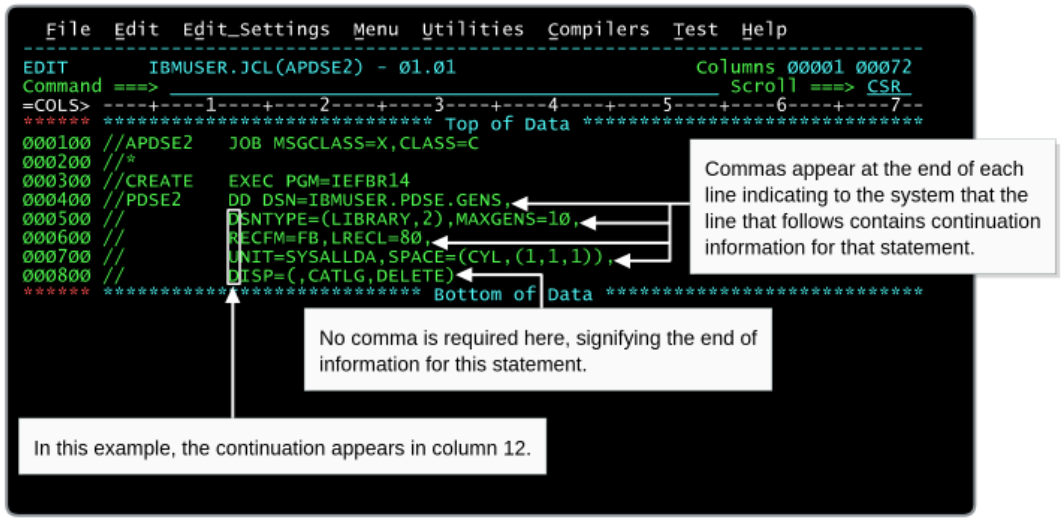
In the top example shown here, a z/OS UNIX file is being referenced, and this needs to appear in lowercase. The example at the bottom of this page shows a name, using both uppercase and lowercase characters, that is enclosed in single quotes. This JOB statement parameter as you will see later allows you to specify a programmer's name to be linked to the job.



You may remember in the previous module that the partitioned data set you are using to store your JCL should have a record length of 80. This originates from punch cards that were used in the early days of computing, where these cards contained 80 columns.

Columns 73-80 of the punch cards, and also these columns in your displayed job, are ignored when submitted. They were traditionally used for sequence numbers.

Click **Play** to see what happens if you code JCL in these columns.

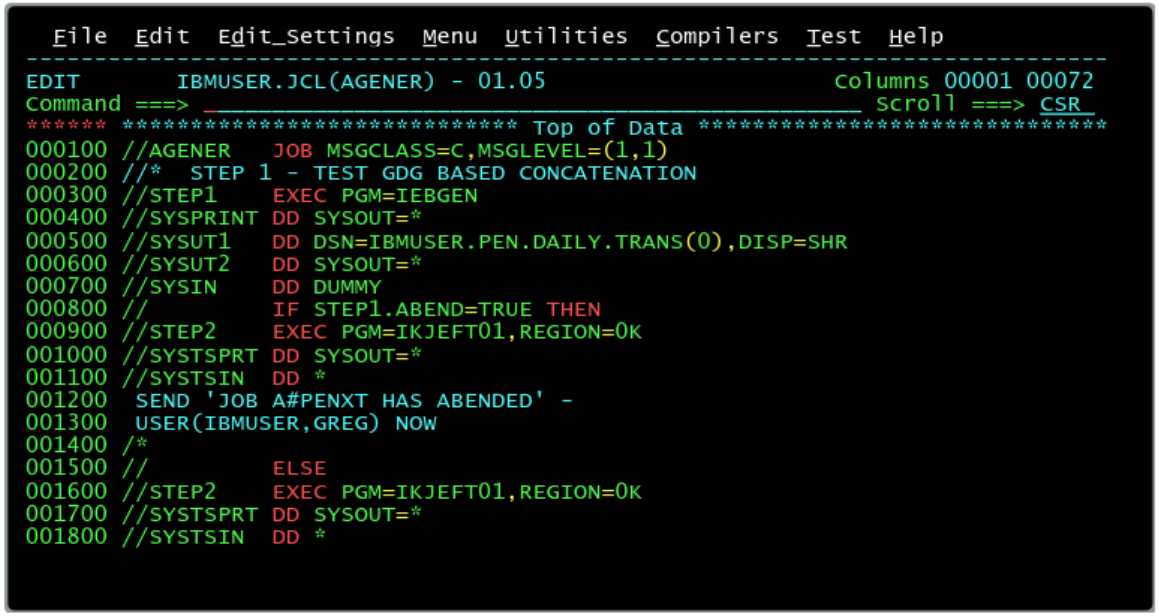


Click Next to see what happens if you do not continue a statement successfully.

Next ►

Since you cannot extend your JCL statement past column 72, how do you handle a JCL statement that contains lots of information? To continue a statement, you code a comma at the end of th parameter being specified on that line, and on the following line the double slash (//) characters must be in columns 1 and 2, and your continued information can appear anywhere between colu 4 and 16 (inclusive).

Often you will see for readability purposes that continued line data is aligned with previous lines.

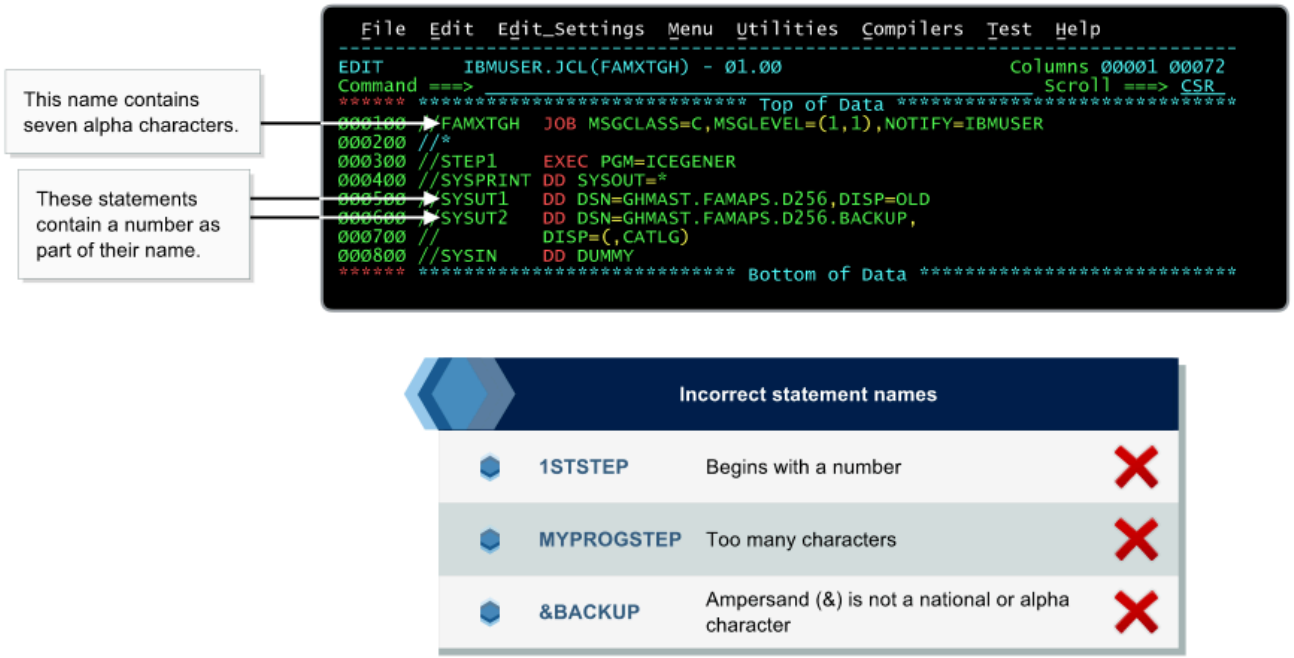


Step 2 of 2

The HILITE command has identified that the code is JCL and is able to provide coloring to differentiate components. For example, comments and raw data appear as turquoise, while statemen types are shown in red. The remainder of this course will use this highlighting feature when working with JCL code.

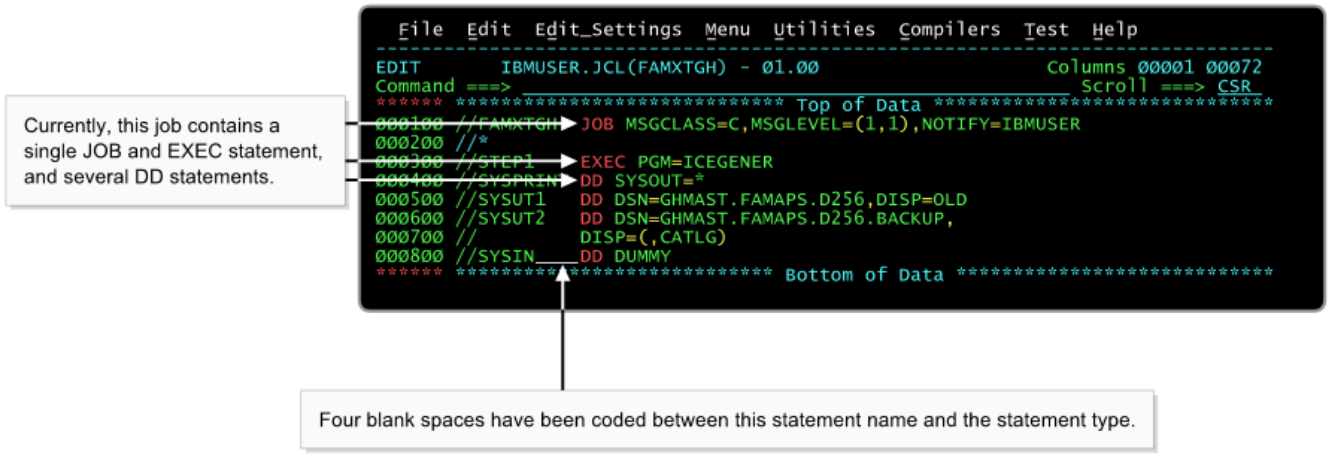
- **JOB** Attributes the system will use when processing the batch job.
- **EXEC** To specify programs that need to be run.
- **DD** For identifying input and output requirements for the program.

Now that you have much of the JCL formatting rules covered, it is time to take a quick look at the general syntax of information displayed in JCL statements. Do not worry, as all this information covered in more detail in later courses.



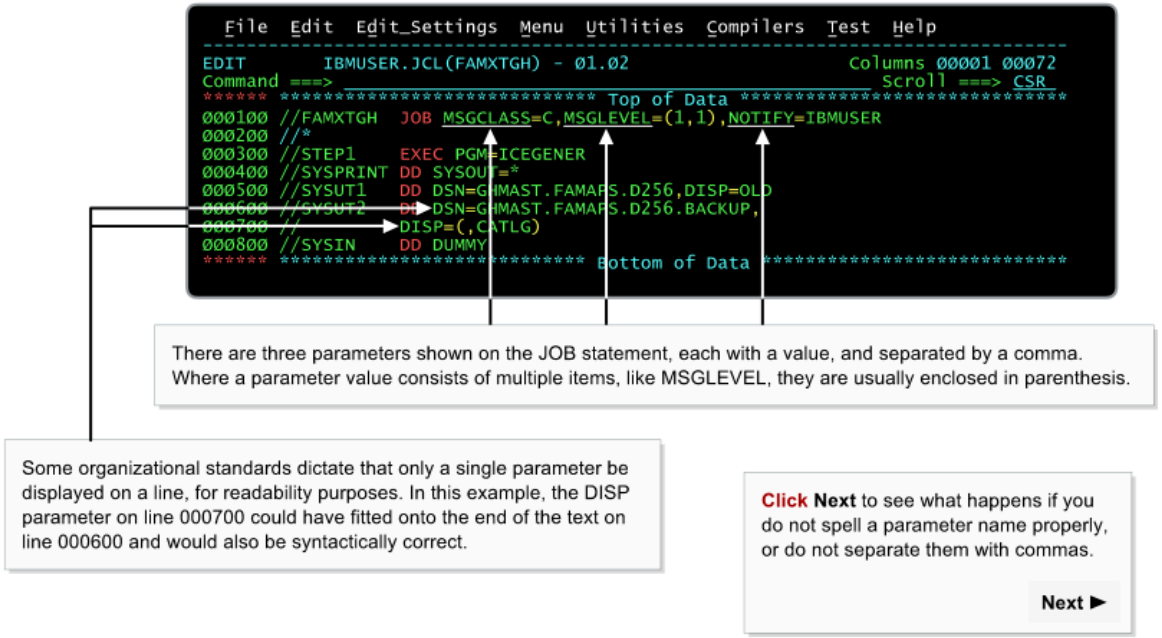
When creating JCL, you will need to tell the system the type of statement you are providing, and in most cases, provide a name for that statement. The name you provide for each statement is c to eight characters and appears immediately after the double slash (//) characters. This name must start with an alpha or national character ($, #, @), while remaining characters can also contai numbers.

As you will see later, depending on the statement type, this name could be one of your choice, or may need to be specific.



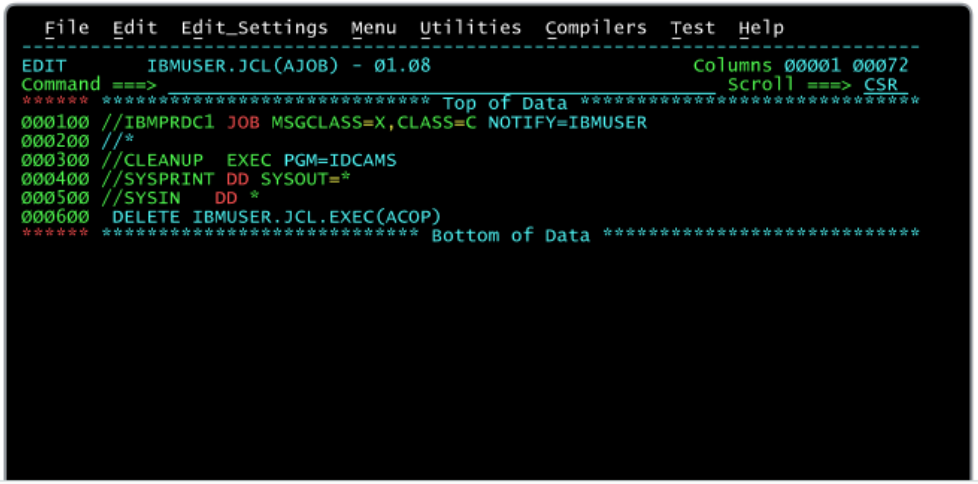
Associated with the name is the statement type. Common statements used include JOB, EXEC, and DD, which are discussed in detail in the courses that follow. At least one space must be cod following the name before this statement type is entered.

If you are working on existing JCL you will often see several spaces between the name and the statement type. This is for readability purposes as it aligns important information.



Every statement will have some parameters that describe requirements, or attributes, to be associated with that statement. There may be many associated with that statement, though in reality you are only likely to use a subset of them regularly. If parameters are specified, at least one space must follow the statement type - JOB, EXEC, or DD - before they are entered.

Where there are multiple parameters for a statement, they must be separated by commas, and if the parameter itself contains a space, it needs to be enclosed in single quotes.

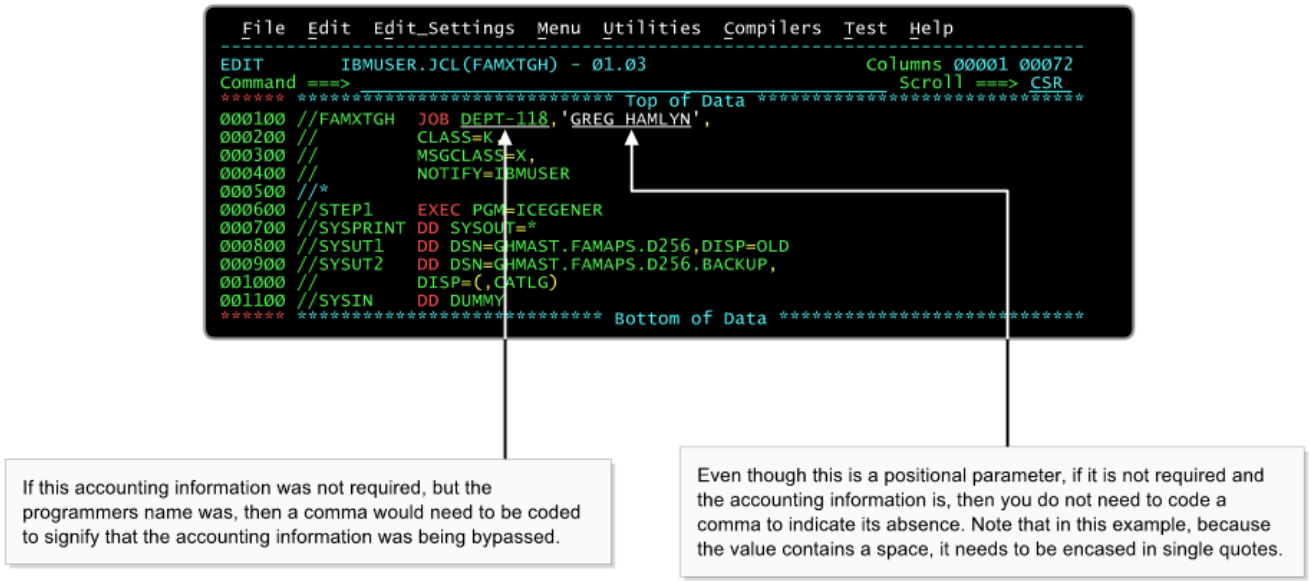


In this example, a space was left between the CLASS and NOTIFY parameters on line 000100, rather than a comma. The system interprets the information following the space as a comment. Note that in this instance the HILITE coloring assists you in identifying this issue, as NOTIFY and its value are displayed in turquoise.

◄ Previous Restart

Every statement will have some parameters that describe requirements, or attributes, to be associated with that statement. There may be many associated with that statement, though in reality are only likely to use a subset of them regularly. If parameters are specified, at least one space must follow the statement type - JOB, EXEC, or DD - before they are entered.

Where there are multiple parameters for a statement, they must be separated by commas, and if the parameter itself contains a space, it needs to be enclosed in single quotes.



The parameters for each statement are separated into positional and keyword.

If used, a positional parameter must appear in a specific area of the code, and if it is not required, a comma is often used to indicate it being bypassed, although this is not always the case. In this example some parameters have been added to the JOB statement. These are discussed in detail in a later course, but for the purpose of this exercise the department accounting information and programmers name that appear on line 000100 are both positional parameters.

```
 File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
--------------------------------------------------------------------------------
EDIT       IBMUSER.JCL(FAMXTGH) - 01.04                Columns 00001 00072
Command ===> _____________________________________________ Scroll ===> CSR
****** ***************************** Top of Data ******************************
000100 //FAMXTGH  JOB 'GREG HAMLYN',
000200 //          CLASS=K,
000300 //          MSGCLASS=X,
000400 //          NOTIFY=IBMUSER
000500 //*
000600 //STEP1     EXEC PGM=ICEGENER
000700 //SYSPRINT DD SYSOUT=*
000800 //SYSUT1    DD DISP=OLD,DSN=GHMAST.FAMAPS.D256
000900 //SYSUT2    DD DSN=GHMAST.FAMAPS.D256.BACKUP,DISP=(,CATLG)
001000 //SYSIN     DD DUMMY
****** **************************** Bottom of Data ****************************
```

Keyword parameters are more common and can appear in any order within the statement, following the statement type. Their name is followed by an equals (=) sign and then the value assigned to that keyword parameter.

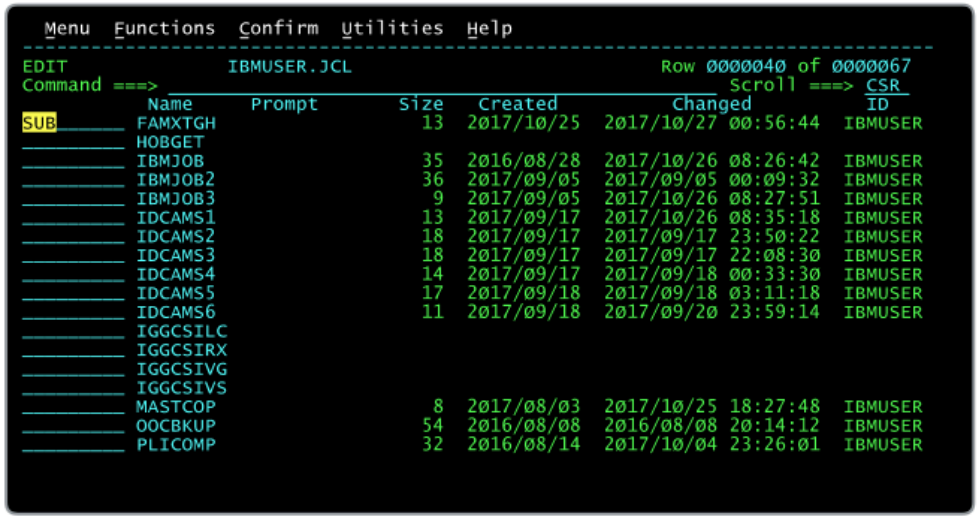
In this example, line 000800 shows a DISP parameter first, and then a DSN parameter. On the line after this, these two parameters appear in the opposite order. As DISP and DSN are keyword parameters this coding is acceptable.

```
 File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
--------------------------------------------------------------------------------
EDIT       IBMUSER.JCL(FAMXTGH) - 01.04                Columns 00001 00072
Command ===> _____________________________________________ Scroll ===> CSR
****** ***************************** Top of Data ******************************
000100 //FAMXTGH  JOB 'GREG HAMLYN',
000200 //          CLASS=K,
000300 //          MSGCLASS=X,
000400 //          NOTIFY=IBMUSER
000500 //*
000600 //*  THE FIRST STEP IS USED FOR RECOVERY PURPOSES
000700 //*  CHANGE THE SYSUT1 AND SYSUT2 DATA SETS AS REQUIRED
000800 //*
000900 //STEP1    EXEC PGM=ICEGENER
001000 //SYSPRINT DD SYSOUT=*
001100 //SYSUT1   DD DISP=OLD,DSN=GHMAST.FAMAPS.D256   * INPUT DATA SET *
001200 //SYSUT2   DD DSN=GHMAST.FAMAPS.D256.BACKUP,DISP=(,CATLG)
001300 //SYSIN    DD DUMMY
****** **************************** Bottom of Data ****************************
```

Note that comments are not interpreted by the system but are simply accepted as they are coded.

As with any coding exercise, it is always best practice to include comments, so that if another person needs to use the JCL, they are made aware of what its purpose is, and what changes they will need to make before using it. Sometimes a comment is also used to break up the JCL code, so it is easier to read.

There are two ways of coding comments in JCL. The more common method is to code a //* statement such as on lines 000500 to 000800. Any text that then appears after this is considered a comment. Another method is to leave at least one space at the end of a line and type your comment, such as on the end of line 001100.



This screen is a list of PDS members accessed using the ISPF 3.4 option. If no editing is required to the JCL, then a SUB, or SUBMIT, command can be entered directly against the PDS member.

◄ Previous Next ►

Although you have a long way to go before submitting your own batch job, it is an opportune time to look at the various methods used to perform this task.

The most common method is for you to type SUBMIT, or SUB, into the ISPF edit command line shown here.

- A maximum of eight characters can be specified for a statement name.
- A statement name needs to begin with an alpha or national character.
- There are two types of parameters that can be specified on a JCL statement – positional and keyword.
- There are two different ways of coding a comment in JCL.
- The SUBMIT command is used to send the JCL to the operating system.

In this section you have seen how specific you sometimes need to be when coding JCL. A simple missing comma or a misspelled parameter can prevent your job from running successfully or running at all. Having said that, many syntax-related errors can be quick to identify and resolve.

The courses that follow this will provide a more in-depth look at the major JCL statements, and offer you many opportunities to code your own JCL statements.

Summary

JCL Coding Requirements

Before beginning to write any JCL code, you need to become familiar with the formatting of JCL statements, and the consequences should these not be followed. In this module you looked at the general syntax and format of JCL statements.

You should now be able to:

- Describe the Formatting Requirements of a JCL Statement
- Identify Common JCL Coding Errors