Table of Contents:

Introduction to Batch Processing

Enterprise Data Processing

Mainframe Data Processing

Data on the mainframe does not come in one shape. For example, every record within a data set containing daily transactions will need to be processed, while other data sets provide the ability to update individual records only.

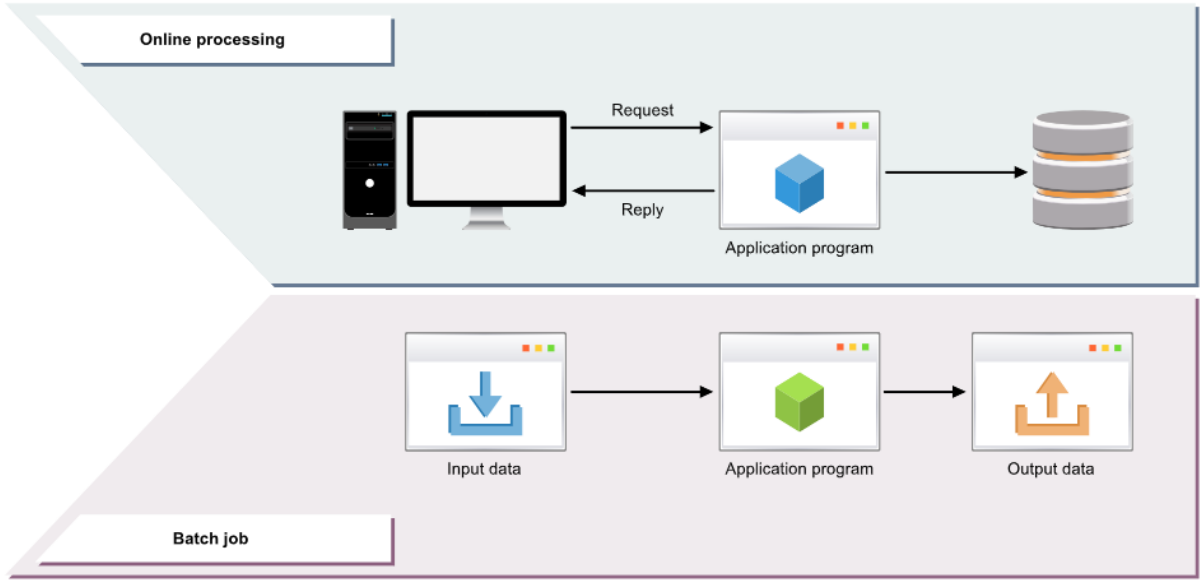
Data Types:

- Partitioned Data Set
    - Contains individual members.
    - An example of this is individual compiled programs residing in a production partitioned data set.
- Sequential Data Set
    - Contains data that is stored sequentially, one record following the next.
    - An example of this would be performance data that will later be printed or a list of transactions that will be applied to a master file.
- VSAM Data Set
    - There are several types of VSAM data set that can be created in a z/OS environment.
    - These are more complex than other types of data sets as they can consist of indexes or keys, to access and retrieve data records.
    - An example of data that would be stored in a VSAM structure would be system catalogs or customer data containing fields such as name, or ID that need to be referenced using that information as a key.
- z/OS UNIX File
    - JCL can be used to run shell scripts or z/OS UNIX application programs against z/OS UNIX data.
    - A wide range of data can be stored in several types of z/OS UNIX files.
- Database File
    - They provide many features and capabilities that can be applied to data stored within them.
    - They are generally used where there is a large amount of data and specific information needs to be accessed quickly.
    - An example of data that would be stored in a database table would be results of research experiments or records of organizational purchases.

Online and Batch Processing

Due to the mainframe containing several types of data with numerous people needing access to it the z/OS needs a way to manage this.

There are two methods for how this can be achieved - online and batch



With online processing you simply enter a request from a screen and press Enter, the relevant data is accessed, and a response is returned to your screen. Online processing is usually reserved for simple, quick tasks.

Batch processing is often used when large amounts of data need to be accessed and worked on, usually at a predetermined time, unlike online processing which is immediate. Batch processing uses JCL to identify the programs to be run, what is to be used as input data, and what needs to be produced as a result.

Benefits of Batch Processing

If online processing can perform tasks instantly, why is there a need for batch processing? This is due to how the cost to make resources available to that extent is not yet economically viable. Hence, batch processing is the perfect solution for situations where there is repetitious, high volume work.

- Updates to data can be performed at a time that is suitable to the organisation.
- It is suitable for large amounts of repeated work i.e. master file updates.
- Dollar costs per workload are significantly less because of the characteristics above.
- Less user interaction is required to schedule and run batch jobs.

Working with Batch Processing

Depending on one's role, our relationship to batch processing is likely to differ. However, regardless of its application if the JCL is not coded correctly, it can have some drastic consequences.

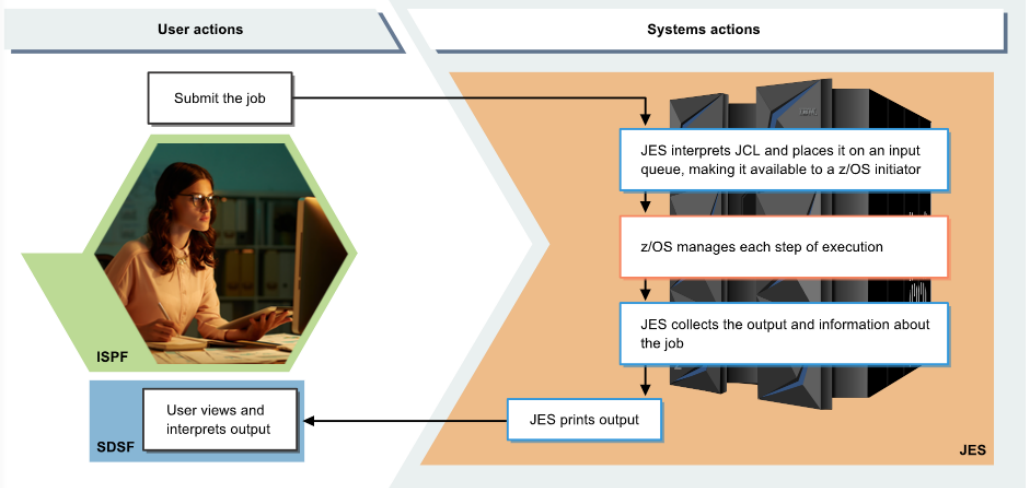
Examples:

- An incorrect version of a master file is being used as input, resulting in overpayment notices being sent to some customers.
- Some parameters that you passed to a sort program have accidentally deleted data that is normally used as input to a later program you have to run.
- The management report that is supposed to be created from your batch job is not being produced.

Running a Batch Job

Understanding how the process works allows one to better diagnose JCL error messages at a later stage.

- Submit the Job
    - JES interprets JCL and places it on an input queue, making it available to a z/OS initiator.
- z/OS manages each step of execution
    - Required resources are allocated - programs, memory, files
    - Resources are freed when the program is finished
- Output time
    - JES collects the output and information about the job
    - JES prints output
- User views and interprets output



Other terminology:

- ISPF - to access the JCL code that is used for a batch job
- JES - you will need some background on how JES, JES2, or JES3 is going to handle your submitted batch job
- SDSF or similar output viewing software - to display the results following the completion of your batch job

Batch Job Automation

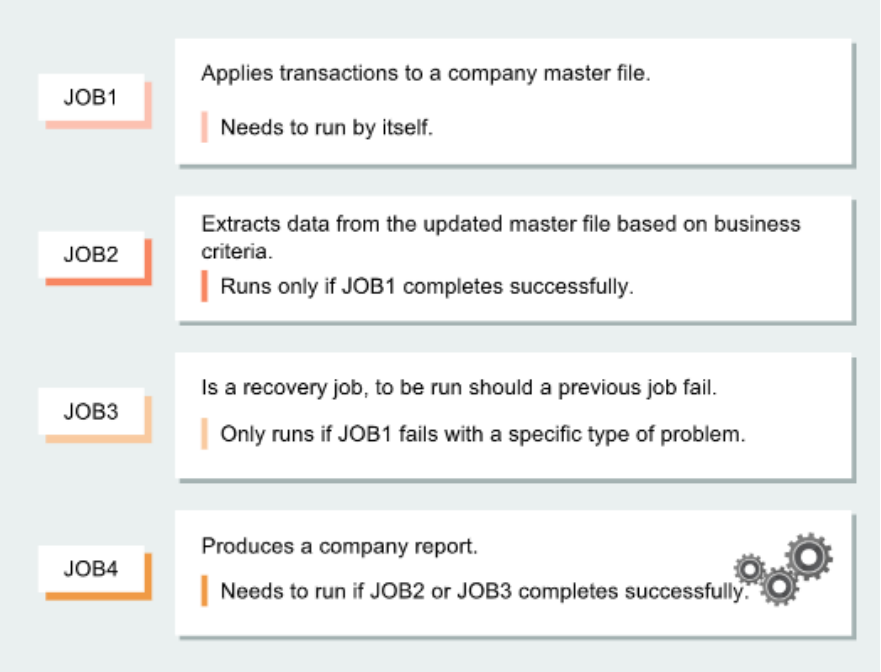
Many larger organizations have implemented automated batch scheduling software that is capable of handling complex batch job scheduling requirements. For example, batch processing may need to run at a specific time, once other data has become available, when a resource becomes free, or simply triggered once another batch job has completed.

Some examples of scheduling products used:

- BMC Control-M Product
- IBM Workload Scheduler
- CA Workload Automation - CA 7 Edition

Batch Job Scheduling

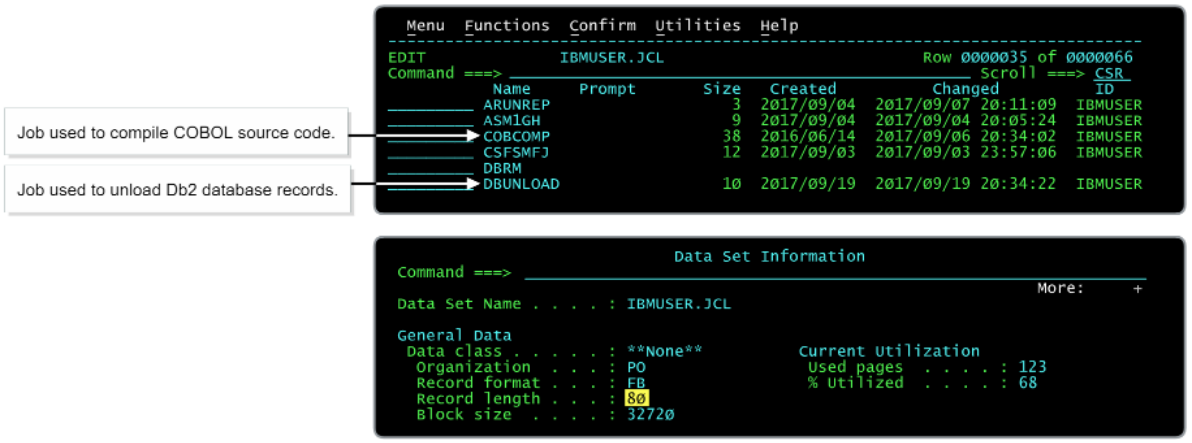
Since z/OS 2.2, we can create simple scheduling for batch jobs within the batch job itself. The JCL code is referenced by JES which can run jobs at specific times or after other jobs have completed.




```
 File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
------------------------------------------------------------------------------
EDIT       IBMUSER.JCL(XRGRP) - 01.03                  Columns 00001 00072
Command ===> ____________________________________________ Scroll ===> CSR
****** ***************************** Top of Data ******************************
000100 //XRGRP   JOBGROUP (MK112),HARRIS,ERROR=(ABEND),ONERROR=STOP
000200 //TRANAPY GJOB
000300 //XTRCT   GJOB
000400 //         AFTER NAME=TRANAPY,WHEN=(RC=0)
000500 //XRSTORE GJOB
000600 //         AFTER NAME=TRANAPY,WHEN=(RC=8)
000700 //XREPORT GJOB
000800 //         AFTER NAME=XTRCT
000900 //         AFTER NAME=XRSTORE
001000 //XRGRP   ENDGROUP
001100 //TRANAPY JOB MSGCLASS=X
001200 //         SCHEDULE JOBGROUP=XRGRP,
001300 //         HOLDUNTL='+03:00'
001400 //APPLY   EXEC PGM=TRNASS
001500 //XTRCT   JOB MSGCLASS=X
001600 //         SCHEDULE JOBGROUP=XRGRP
001700 //EXTRACT EXEC PGM=TRNXRT
001800 //XRSTORE JOB MSGCLASS=X
```

Batch Processing Prerequisites

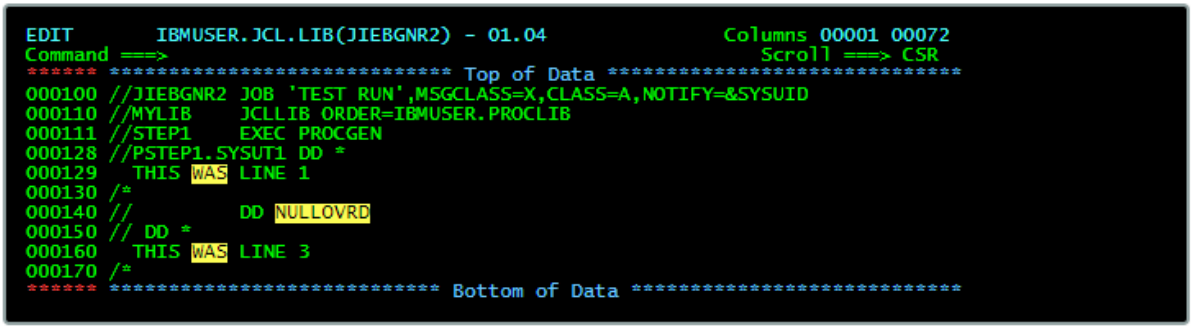
The data set used for storing batch jobs is the partitioned data set (PDS) This is because it contains individual members, thus, making it an ideal location to store all batch jobs belonging to you or your group.



JCL z/OS 2.4 Update

There are only minor changes made to JCL with the introduction of z/OS 2.4, with one being included here.

A new NULLOVRD parameter can be used when performing overrides of DD * or DD DATA statements.



Coding Requirements

JCL Coding Rules

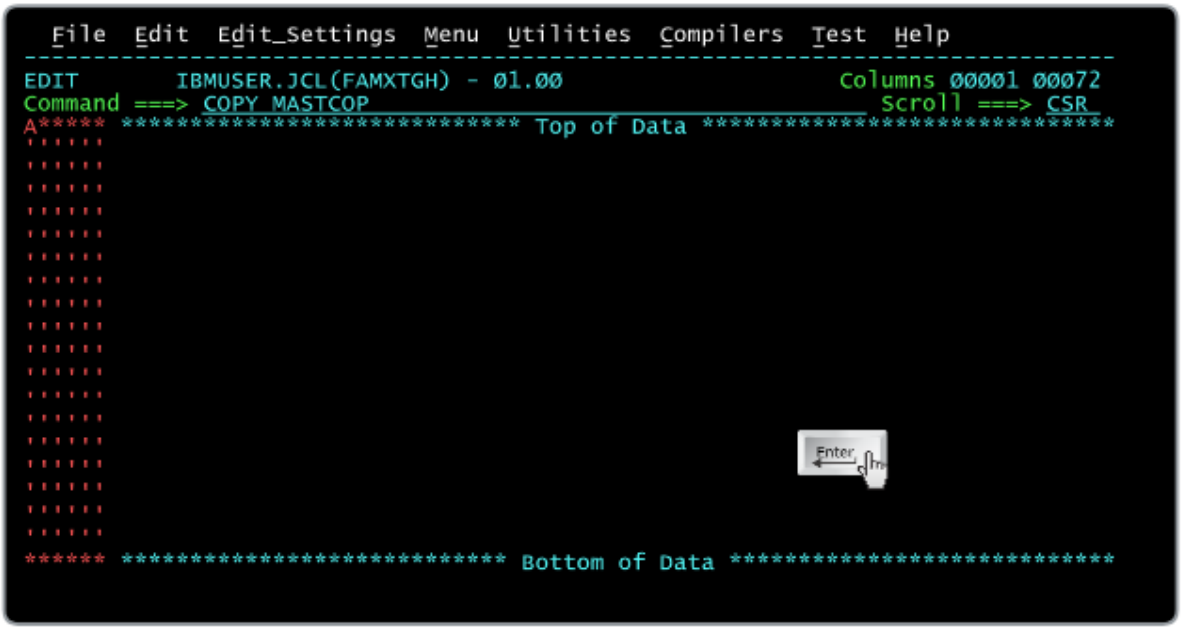
Planning Processing Requirements

Before writing any JCL, one must first plan out the batch processing that is to be performed.

- 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

Reality of JCL Coding

It's extremely unlikely that one will ever write out JCL for a job from scratch as it is much easier to copy it from a similar job and modify it.

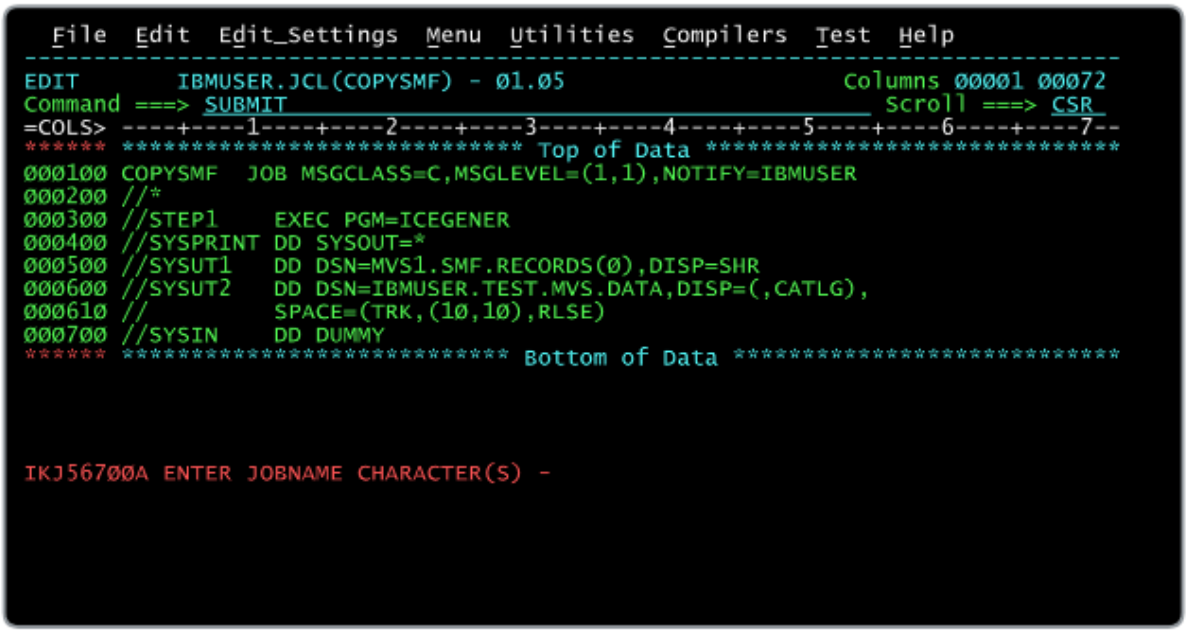


How to copy JCL:

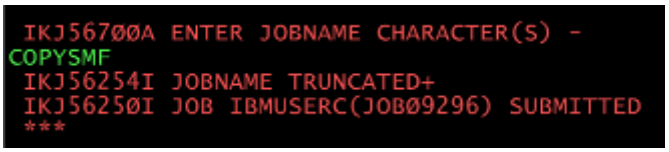
- Create an empty PDS member
- Type Copy [Other file's name] into the command line
- Place an A in the empty PDS member's line right at the top then hit enter
- Make your changes then submit it once you're ready

Identifying JCL code

JCL statements begin with a double slash within columns 1 and 2. This is how the system is able to interpret the data that is submitted to the system as JCL. Do note that there are, however, some exceptions to this double slash rule.



In the example above, the system cannot determine the name of the job because it does not recognise the first line due to the lack of double slash (//) at the beginning of the line.

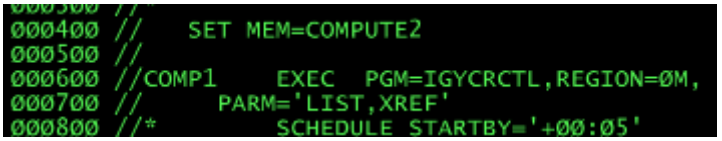


What the initial prompt did not not mention was that it will use one's user ID (IBMUSER) as part of the job name and that one needs to add characters to append to it. Since the name of the job can only be eight characters, it has truncated the response using the C only.

The empty // curse

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 JCL.

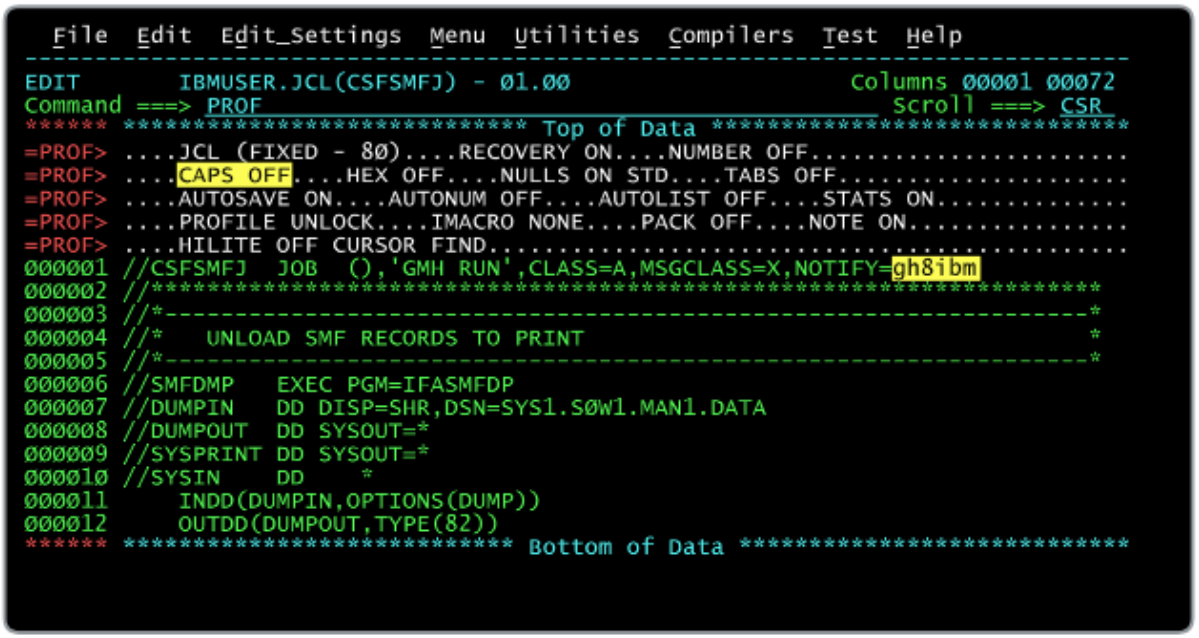
In the example here, the user wanted to remove the content from a JCL statement, but left // by itself on line 000500. Even though JCL statements appear after this line, the system will not recognize them.



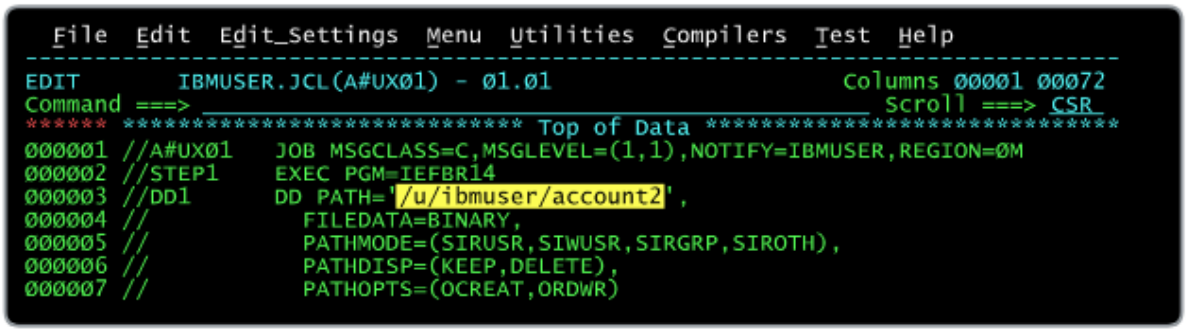
Uppercase Characters

JCL is coded in uppercase characters and you can use the PROF command in the CLI to check whether you have caps set to on or off.

- This can be fixed by using CAPS ON in the command line to automatically have your input converted to uppercase whenever you hit enter.



However, not everything has to be in uppercase; there are a few exceptions where you'll need to use lowercase characters enclosed within single quotes. The example shown below showcases this concept through a z/OS UNIX file being referenced.



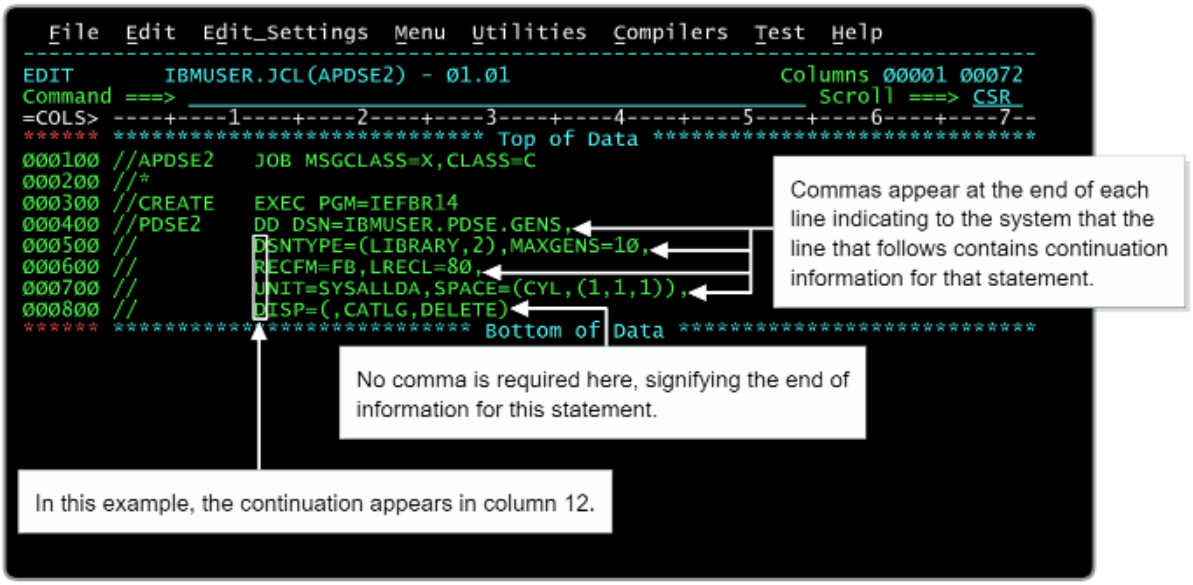
Record Length

Within the PDS member that stores one's JCL code, it has a record length of 80 - based on punch cards from the early days of computing. Columns 73 - 80 are ignored when a job is submitted as they were traditionally used for sequence numbers. Thus, avoid code that overruns into these columns to prevent errors popping up when the job is submitted.

Continuing a statement

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 the 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 columns 4 and 16 (inclusive).

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

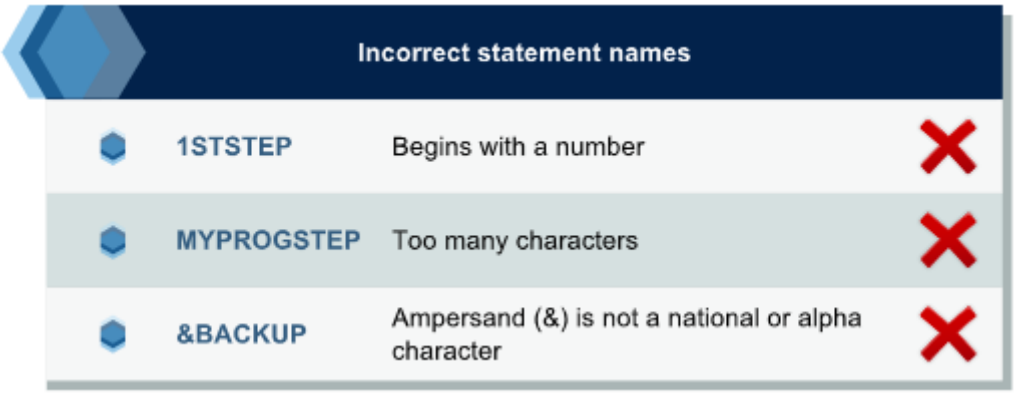


Additionally, one should avoid putting any code in column 72 itself as traditionally this column was used to indicate a continuation but today is generally no longer used for this purpose and needs to be blank.

Statement Breakdown

Statement Names

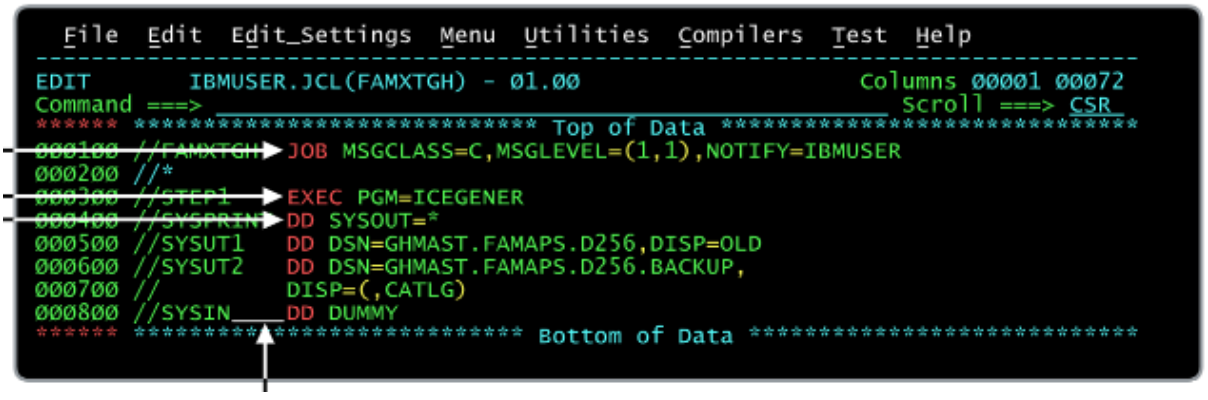
When creating JCL, you will need to tell the system the type of statement you are providing, and in most cases, provide a name for that statement. The name you provide for each statement is one to eight characters and appears immediately after the double slash (//) characters. This name must start with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



Statement Types

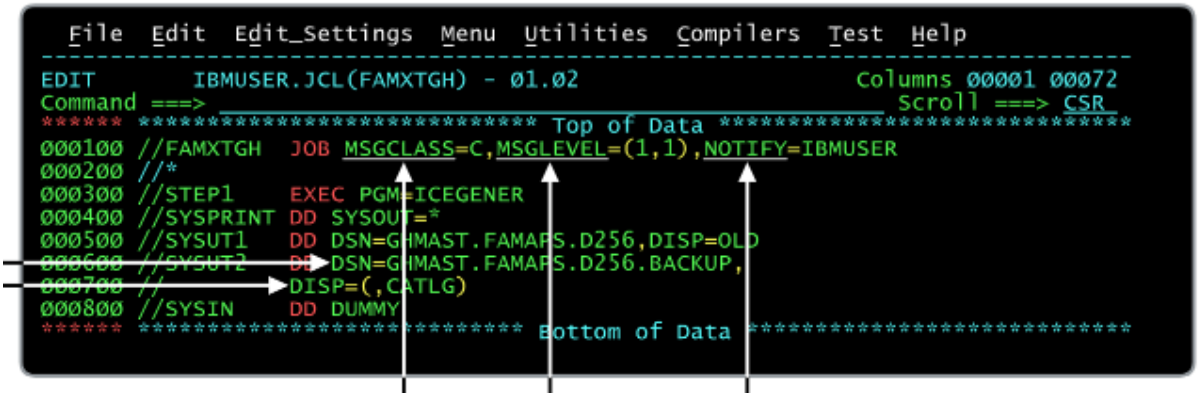
Associated with the name is the statement type. Common statements used include JOB, EXEC, and DD (as seen in the image below). At least one space must be coded following the name before this statement type is entered.

If you are working on an 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.



Parameters

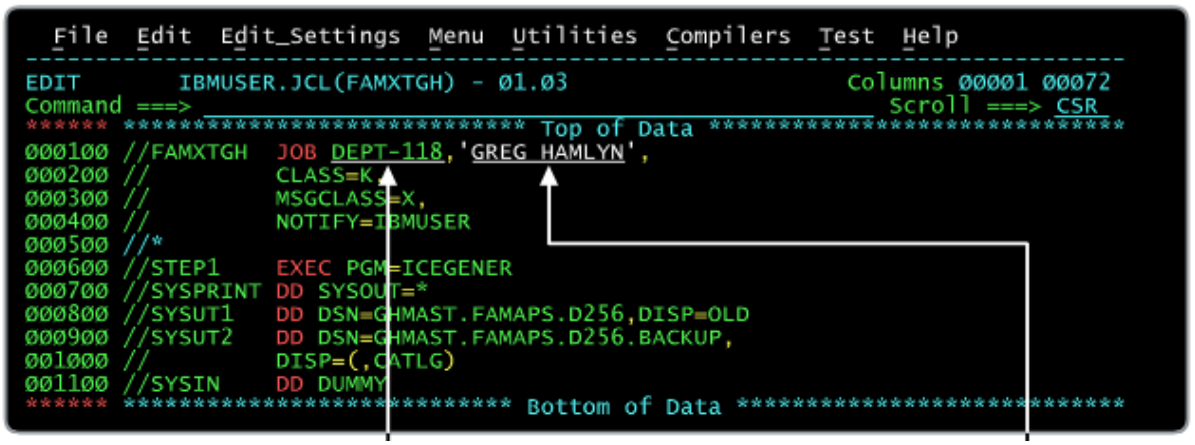
Every statement will have some parameters that describe requirements, or attributes, to be associated with that statement. There may be many associated with that statement, though in reality you are only likely to use a subset of them regularly. If parameters are specified, at least one space must follow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

Positional Parameters

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.



Example:

- Left arrow: If this accounting information was not required but the programmer's name was, then a comma would need to be coded to signify that the accounting information was being bypassed.
- Right arrow: Even though this is a positional parameter, if it is not required and the accounting information is, then you do not need to code a comma to indicate its absence. Note that in this example, because the value contains a space, it needs to be encased in single quotes.

Keyword Parameters

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.

```
 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 ****************************
```

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.

Comments

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.

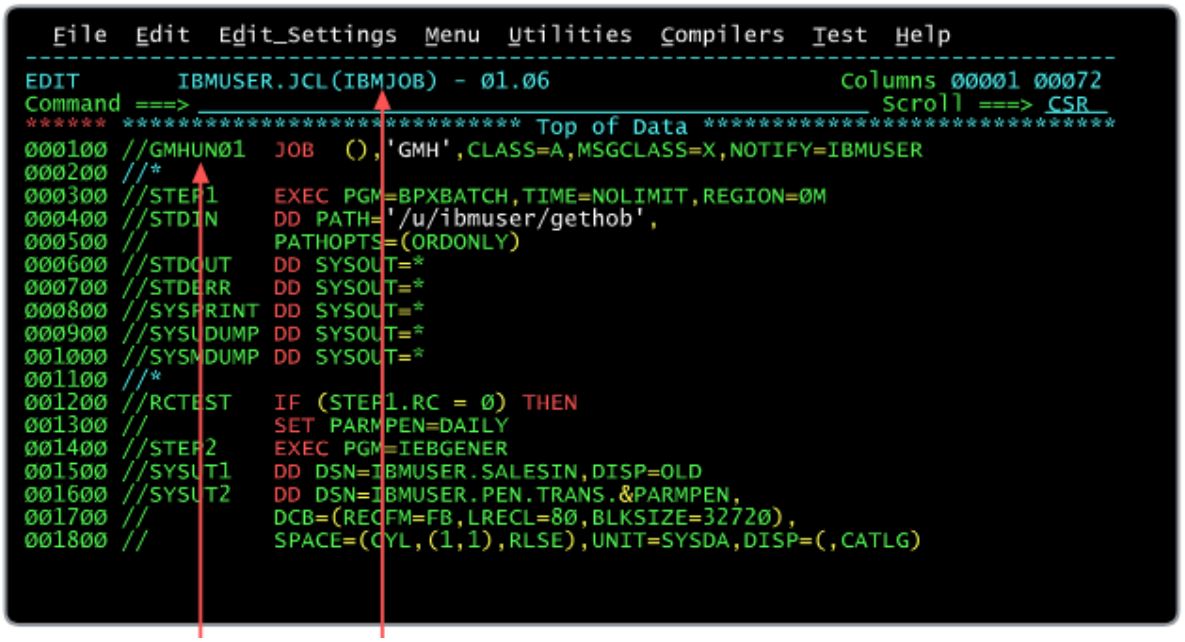
```
 File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
-------------------------------------------------------------------------------
EDIT       IBMUSER.JCL(FAMXTGH) - 01.04                 Columns 00001 00072
Command ===> ____________________________________________ Scroll ===> CSR
****** ***************************** 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 ****************************
```

JOB Statement Basics

Statement Requirements

JOB Statement Importance

The JOB statement is typically the first statement encountered in your job, it is used to specify attributes to be associated with your job when it is submitted to the system. The JOB statement is important and if not coded correctly, can have major ramifications on the success of your job.



The name you specify on this statement is the one that the system uses to reference your job when it is submitted. The PDS member name you are using to store your JCL can be the same or different to the job name and has no relationship with it.

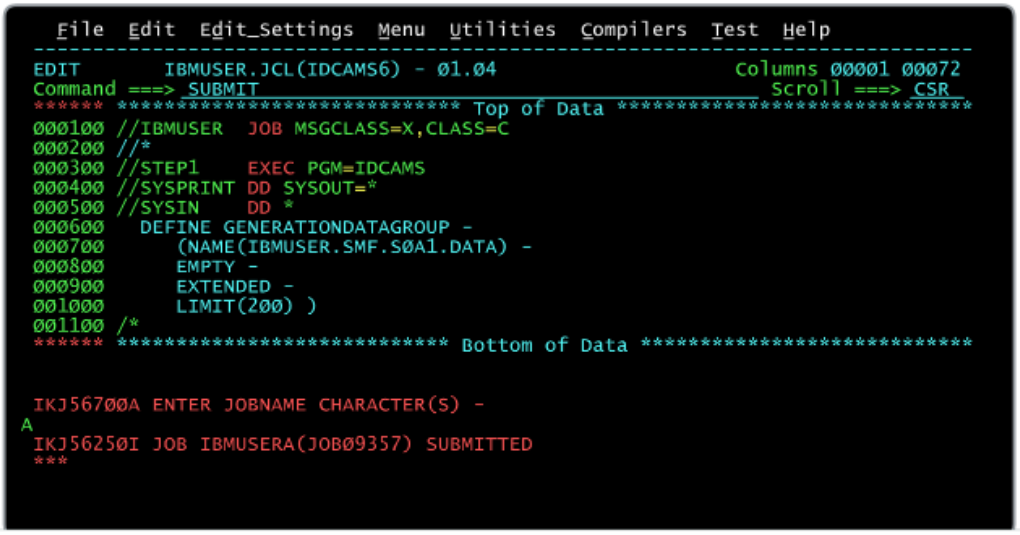
JOB Name Standards

Mentioned previously in a section above, a statement name needs to meet specific rules - it should be between one and eight characters in length and begin with an alpha or national character. It can contain a number as long as it is not the first character.

Each organization will probably have its own standards regarding job names. This allows the organization to more clearly identify to whoever is looking at the job, what group, or individual it belongs to.

Running a JOB Multiple Times

To submit a job several times, for testing purposes for example, you can code your user ID as the job's name. When you submit your job, it will prompt you for a character to be added to the end of this name before it is submitted.

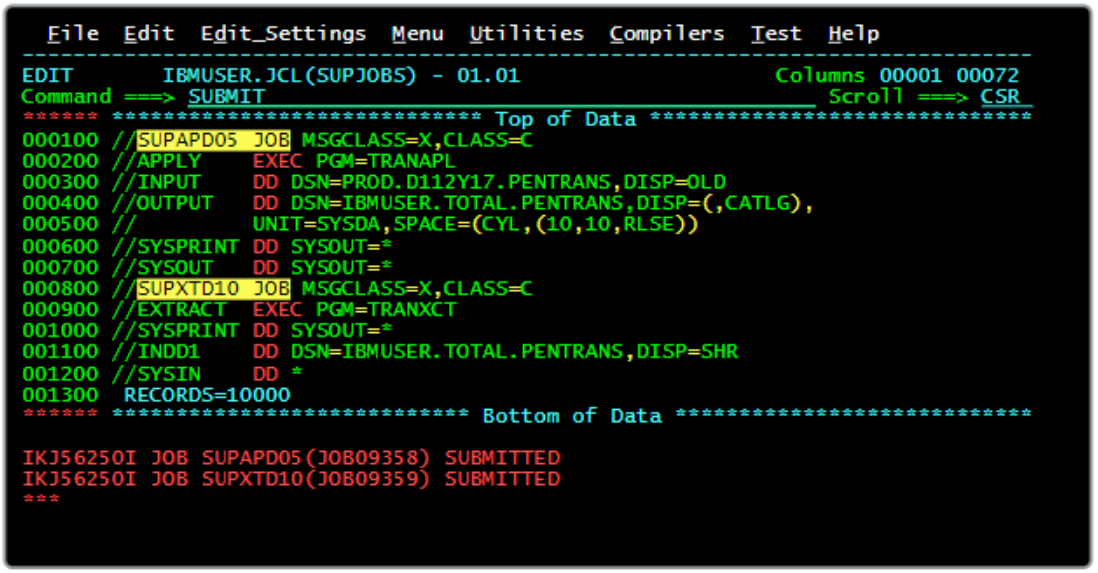


The A character has been appended to IBMUSER, which was specified in the JOB statement, to create the job name that the system will use. You can now use the same job, changing the name of the generation data set to S0B1, and when you submit the job you can use B for the appended job name character. When you check the output from these jobs, it is easy to determine which output belongs to which job.

Note that to use this option, one's user ID needs to be a maximum of seven characters, because a maximum of eight characters are allowed for a job name.

Multiple JOB Statements

Normally, there is just a single JOB statement followed by one or more steps used to execute programs. However, in some situations there might be several JOB statements coded within a PDS member. When submitted to the system, it will detect the JOB statement and submit the statements following it as a separate job.



Incorrect JOB Name

Depending on the problem, the system may prompt you for some input that it will use with system defaults, to build a JOB statement for you, or it may fail indicating that it has an invalid name. In this example it failed because the job name did not start with an alpha or national character.

```
  File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
--------------------------------------------------------------------------------
EDIT       IBMUSER.JCL(IBMJOB3) - 01.05                Columns 00001 00072
Command ===> ____________________________________________ Scroll ===> CSR
****** ***************************** Top of Data ********************************
000100 //1DAYMOD  JOB  ,'GMH',CLASS=C,MSGCLASS=X,NOTIFY=IBMUSER
000200 //*
000300 //          SET WDAY=THURS
000400 //STEP1     EXEC PGM=IEBGENER
000500 //SYSUT1    DD DSN=IBMUSER.JSONDATA,DISP=SHR
```

```
SDSF OUTPUT DISPLAY 1DAYMOD  JOB09360  DSID     2 LINE 0       COLUMNS 02- 81
COMMAND INPUT ===>                                            SCROLL ===> CSR
3.14.27 JOB09360 ---- MONDAY,     30 OCT 2017 ----
3.14.27 JOB09360  IRR010I  USERID IBMUSER  IS ASSIGNED TO THIS JOB.
3.14.27 JOB09360  IEFC452I INVALID - JOB NOT RUN - JCL ERROR  975
----- JES2 JOB STATISTICS ------
. . .
        1 //1DAYMOD  JOB  ,'GMH',CLASS=C,MSGCLASS=X,NOTIFY=IBMUSER
          //*
        2 //          SET WDAY=THURS
. . .
STMT NO. MESSAGE
        1 IEFC662I INVALID LABEL
******************************** BOTTOM OF DATA ********************************
```

Statement Positional Parameters

With a correct job name and type of statement defined, you now need to look at the types of parameters that can be coded on a JOB statement.

There are two positional parameters that can be specified on a JOB statement - **accounting information**, and the **programmer's name**. As discussed previously, positional parameters, if used, need to appear in a specific order.
The value for these two parameters may be enforced through organizational standards, therefore supplying them in the JOB statement could be optional or mandatory.

```
  File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
--------------------------------------------------------------------------------
EDIT       IBMUSER.JCL(FAMXTP15) - 01.07               Columns 00001 00072
Command ===>                                             Scroll ===> CSR
****** ***************************** Top of Data ********************************
000100 //FAMXTP15 JOB (170A80),'GINA HARRIS',
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 DSN=GHMAST.FAMAPS.D256,DISP=OLD
001200 //SYSUT2    DD DSN=GHMAST.FAMAPS.D256.BACKUP,DISP=(,CATLG)
001300 //SYSIN     DD DUMMY
****** ***************************** Bottom of Data ****************************
```

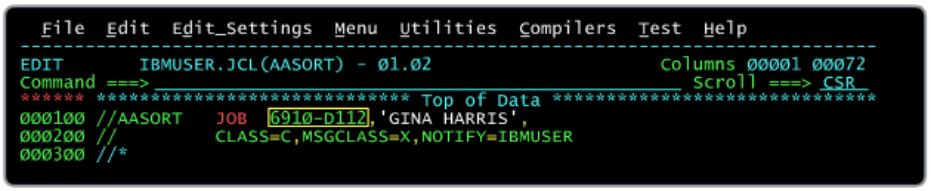
Accounting Information:

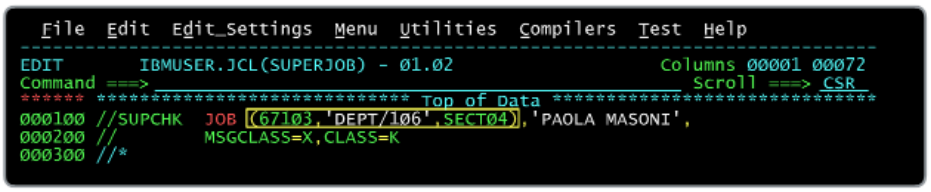
The accounting information parameter is used generally for billing, statistical, or performance-related purposes. The use of this parameter is likely to be dictated by an organization's standards. If used, this parameter must appear before any other JOB statement parameters.

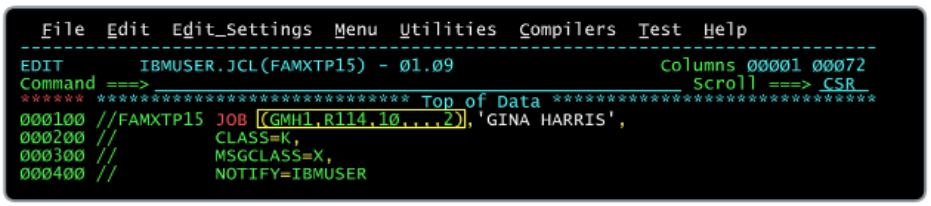
The following are some rules when coding this parameter:

- If more than one subparameter is required, you must code the sub parameters within parentheses. You may also see a single parameter here enclosed in parentheses which is also acceptable.
- If sub parameters contain special characters, with the exception of hyphens, they must appear within apostrophes.
- It cannot exceed 143 characters in length.

The accounting information parameter can consist of two sub parameters - the account number, and more granular details associated with the account. You may also see JES2 accounting information supplied here, although this is not commonly used by organizations.







Examples:

- Image 1 - This is an example of an accounting code defined by the organisation. Even though it contains a hyphen, it does not need to be enclosed in parentheses or apostrophes, although syntactically this is allowed.
- Image 2 - This is an example wherein the organization has defined their accounting information for a job must contain a code, department number, and section name. As the department number contains a slash (/), it needs to be enclosed in apostrophes. As there are three sub parameters altogether, they are enclosed in parentheses.
- Image 3 - When using this parameter to supply JES2 accounting information, a range of sub parameters can be specified. These are positional sub parameters, in this example there are several commas entered to indicate values being bypassed.

Programmer's Name:

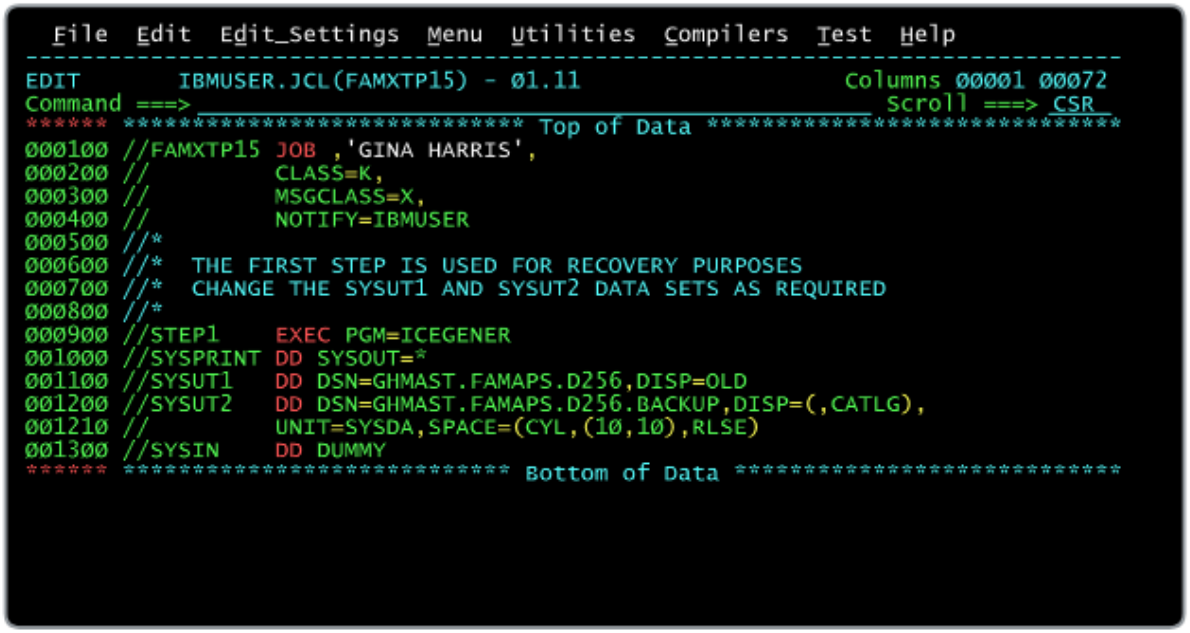
The programmer's name is another optional parameter that may be required by your organization. This field helps to provide identification about the owner of the job. This is often the name of an individual but could also be the name of your group. You can see here that the job's log at the bottom displays this information.

When coding the programmer's name parameter a maximum of 20 characters can be coded and single quotes are required if the name contains any special characters, other than hyphens (-) or periods (.).

Some or No Positional Parameters

If a positional parameter is not required, a comma is coded to signify its absence. This is required when another positional parameter immediately follows it. If no positional parameters are required at all for the statement, they can be omitted without any commas.

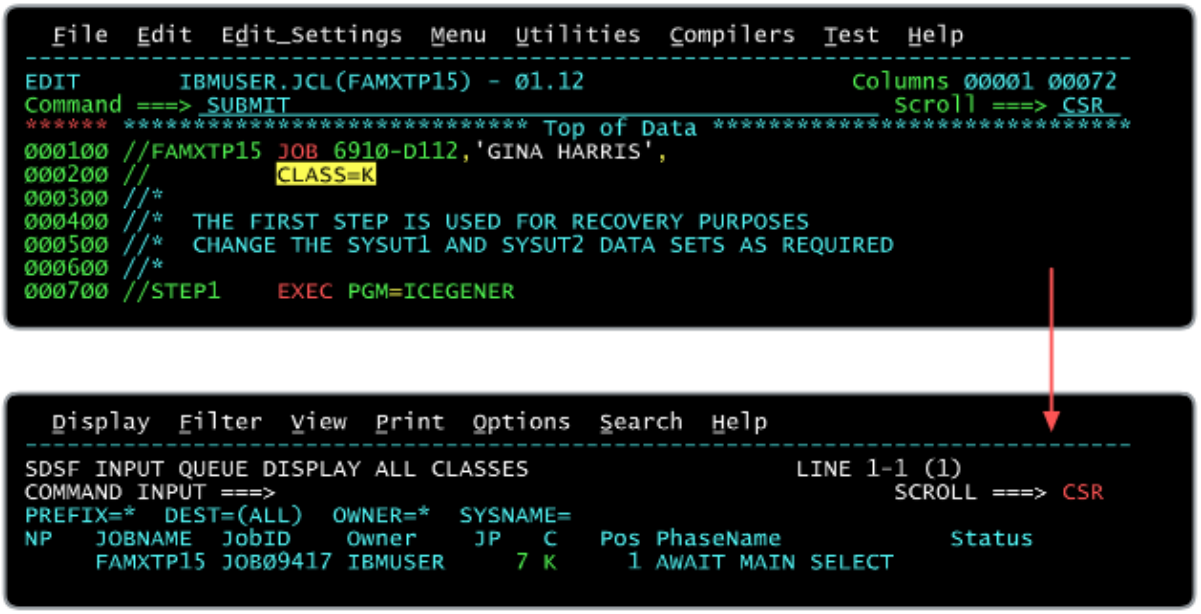
The example below is for when you omit the accounting information.



Statement Keyword Parameters

CLASS Parameter

During JES initialization, organizational job classes are defined with each containing their own characteristics. These classes can be a single number (0-9), letter (A-Z), or even a one to eight-character name. The example at the bottom of this page shows attributes assigned to K class jobs during JES initialization.

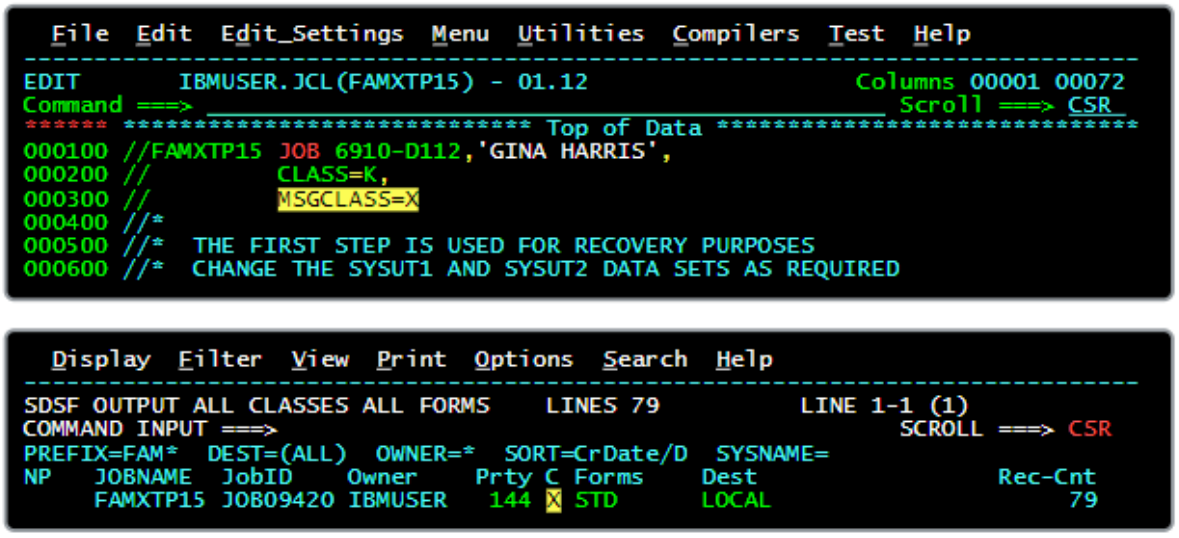


One of the items you would normally code on a JOB statement is a CLASS parameter and you can see in the screen at the top that the syntax is straightforward.

- If an invalid class is specified in the JOB statement the job will be flushed or cancelled and an error message will be displayed.

MSGCLASS Parameter

The MSGCLASS parameter is used to assign the job's log to an output class. This is what you look at in SDSF to determine whether your job ran successfully. Like the CLASS parameter, the attributes associated with output classes are defined during JES initialization.

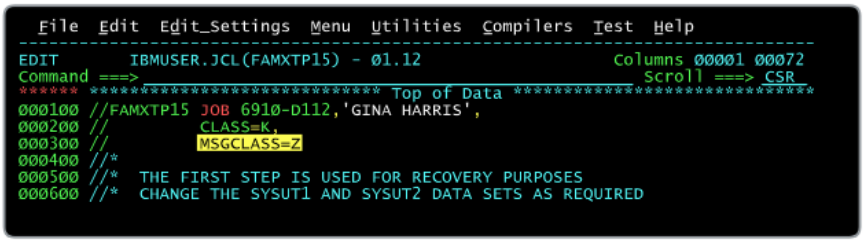


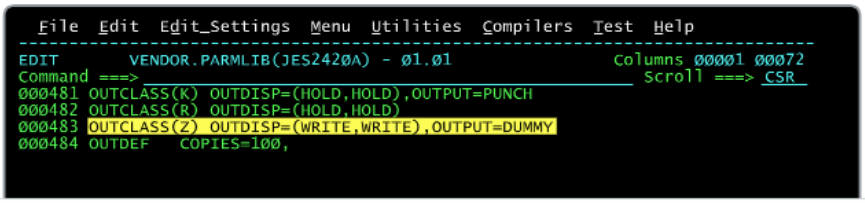
Note that you will again be likely to have organizational standards that specify the output class you should use.

- MSGCLASS=Z is often configured to automatically purge job log output on completion of the job.

If you do not code a MSGCLASS parameter then installation defaults will be used, so it is usually good practice to code this parameter so that you can find your output easily.

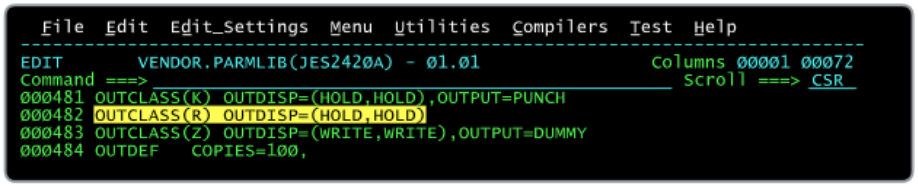
In the example from the previous page, JES initialization statements defined that X class output is not held, and this is why you found it on the SDSF output screen rather than the held output screen. This is not always going to be the case.

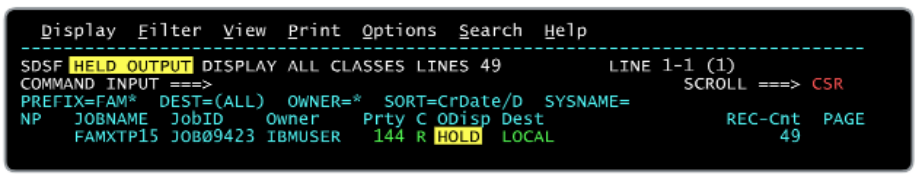




In this example, a MSGCLASS of Z is specified in the JCL. In the JES initialization parameters, notice that for this class it displays OUTPUT=DUMMY, which means that the output will not be available for viewing. You might want to use this class for your job if you are running it many times and do not need to ever view its job log.

◄ Previous Next ►





You can see in the JES initialization parameters at the top that R class output is assigned a HOLD status. If the job had a MSGCLASS of R you would have to search the SDSF held output screen to find it, not the normal output screens previously used.

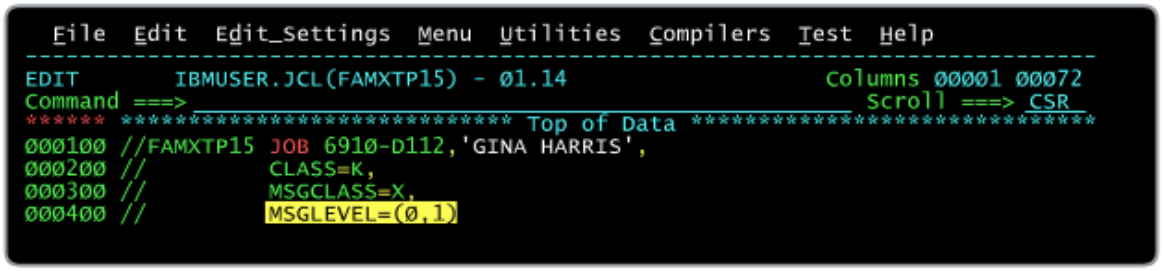
◄ Previous Restart

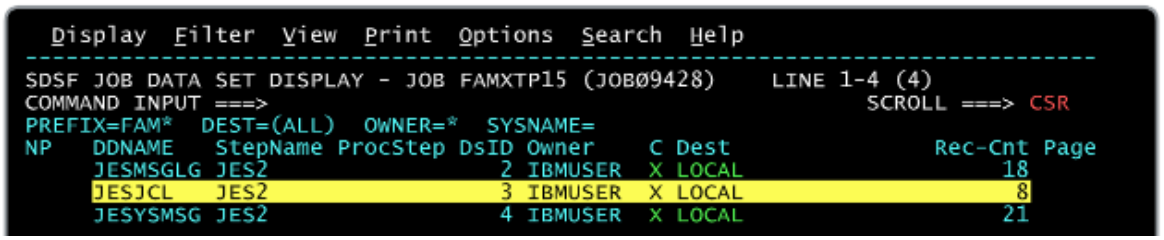
MSGLEVEL Parameter

A job's log output, which was discussed on the previous page, consists of a number of separate output components including the following:

- JES job log
- JCL statements
- Job-related JES and operator messages - system messages

There may be times when you do not need all of this information to be made available because it can clutter the screen making it difficult for you to find what really matters to you.





The MSGLEVEL parameter consists of two sub parameters. The first indicates which statement images to produce. The second sub parameter indicates which system messages to produce. The default is usually MSGLEVEL=(1,1), which will show all JCL statements and messages. A MSGLEVEL default may also be defined for each JES JOBCLASS definition.

In this example, specifying 0 for the first sub parameter will only produce the JOB statement and any comments that appear before the first step. You can see in the output produced that the number of records for the KCL messages have been reduced as a result of this.

```
 Display  Filter  View  Print  Options  Search  Help
-------------------------------------------------------------------------------
SDSF JOB DATA SET DISPLAY - JOB ARUNREP  (JOB09429)    LINE 1-7 (7)
COMMAND INPUT ===>                                     SCROLL ===> CSR
PREFIX=ARUN*  DEST=(ALL)  OWNER=*  SYSNAME=
NP    DDNAME   StepName ProcStep DsID Owner    C Dest              Rec-Cnt Page
      JESMSGLG JES2                 2 IBMUSER  C LOCAL                33
      JESJCL   JES2                 3 IBMUSER  C LOCAL                24
      JESYSMSG JES2                 4 IBMUSER  C LOCAL                67
```

```
 Display  Filter  View  Print  Options  Search  Help
-------------------------------------------------------------------------------
SDSF JOB DATA SET DISPLAY - JOB ARUNREP  (JOB0943Ø)    LINE 1-7 (7)
COMMAND INPUT ===>                                     SCROLL ===> CSR
PREFIX=ARUN*  DEST=(ALL)  OWNER=*  SYSNAME=
NP    DDNAME   StepName ProcStep DsID Owner    C Dest              Rec-Cnt Page
      JESMSGLG JES2                 2 IBMUSER  C LOCAL                33
      JESJCL   JES2                 3 IBMUSER  C LOCAL                 3
      JESYSMSG JES2                 4 IBMUSER  C LOCAL                67
```

Coding 2 for the first sub parameter will produce JCL and JES statements but not procedure statements. The output at the top did not have any MSGLEVEL parameter and you can see that it produced 24 records for the JESJCL part of the output. The same job when submitted with MSGLEVEL=(2,1) produced the output at the bottom showing only 3 records being produced.

```
 Display  Filter  View  Print  Options  Search  Help
-------------------------------------------------------------------------------
SDSF JOB DATA SET DISPLAY - JOB FAMXTP15 (JOB09425)    LINE 1-4 (4)
COMMAND INPUT ===>                                     SCROLL ===> CSR
PREFIX=FAM*  DEST=(ALL)  OWNER=*  SYSNAME=
NP    DDNAME   StepName ProcStep DsID Owner    C Dest              Rec-Cnt Page
      JESMSGLG JES2                 2 IBMUSER  X LOCAL                18
      JESJCL   JES2                 3 IBMUSER  X LOCAL                13
      JESYSMSG JES2                 4 IBMUSER  X LOCAL                21
```

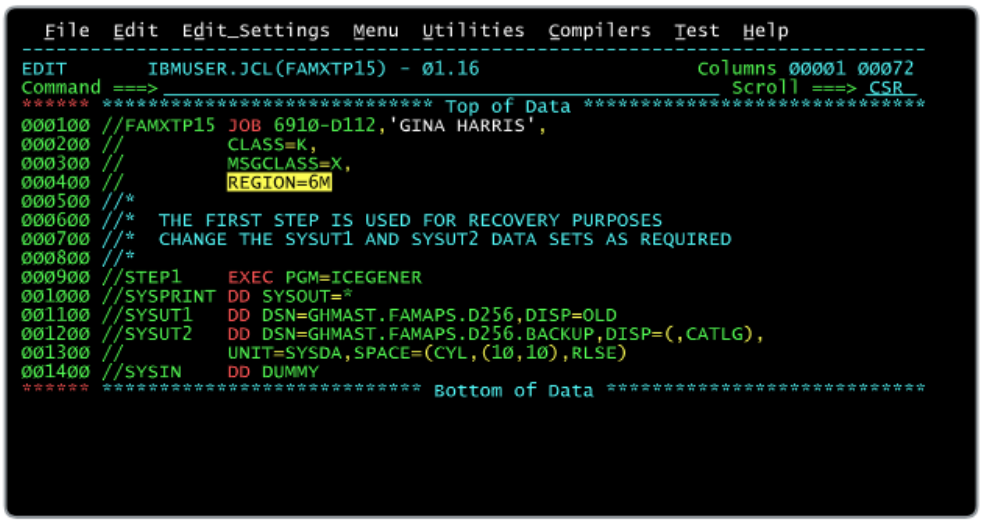
```
 Display  Filter  View  Print  Options  Search  Help
-------------------------------------------------------------------------------
SDSF JOB DATA SET DISPLAY - JOB FAMXTP15 (JOB09434)    LINE 1-4 (4)
COMMAND INPUT ===>                                     SCROLL ===> CSR
PREFIX=FAM*  DEST=(ALL)  OWNER=*  SYSNAME=
NP    DDNAME   StepName ProcStep DsID Owner    C Dest              Rec-Cnt Page
      JESMSGLG JES2                 2 IBMUSER  X LOCAL                18
      JESJCL   JES2                 3 IBMUSER  X LOCAL                13
      JESYSMSG JES2                 4 IBMUSER  X LOCAL                11
```

Coding 0 for the second sub parameter will produce only JCL messages, unless the job fails, in which case JES and operator messages will also be produced. In this situation SMS messages will also be shown, if SMS has caused the job to fail. The example at the top is showing all job log output, while the one at the bottom is produced using MSGLEVEL=(1,0).

REGION Parameter

Every program you run is going to need a different amount of memory to run. Many organizations will use a REGION parameter to cap this requirement.

Coding REGION=0M or REGION=0K on the JOB statement provides every program specified in your job with as much virtual storage - memory - as it requires. Depending on your requirements, you can code a specific maximum amount of memory that can be used, in megabytes (M) or kilobytes (K). If REGION is not specified, then a JES initialization default will take effect.

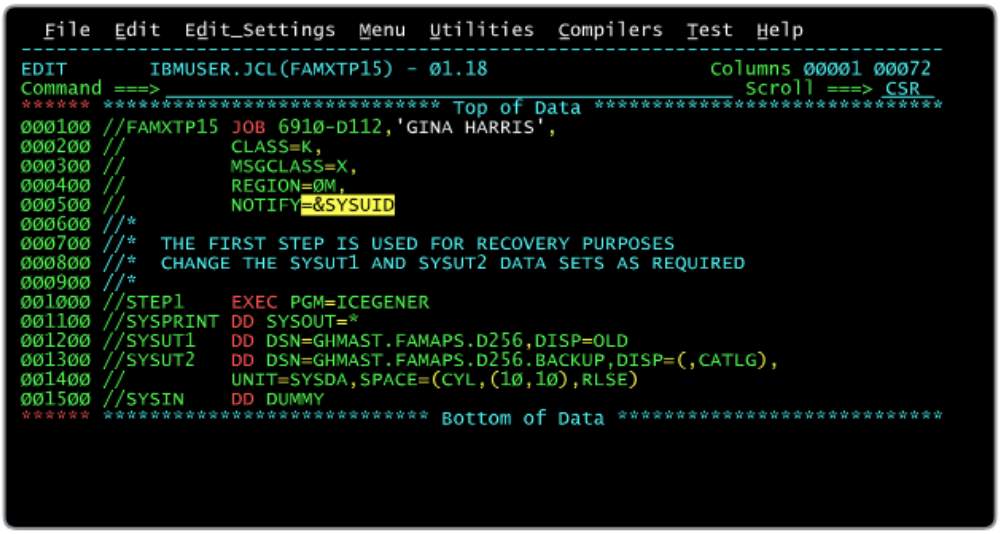


NOTIFY Parameter

When you submit your job to the system, most people will want to be notified once the job has completed, so it can be checked. This task is achieved using the NOTIFY parameter.

The syntax of this command is relatively straightforward. The NOTIFY value is a TSO user ID, usually the person submitting the job.

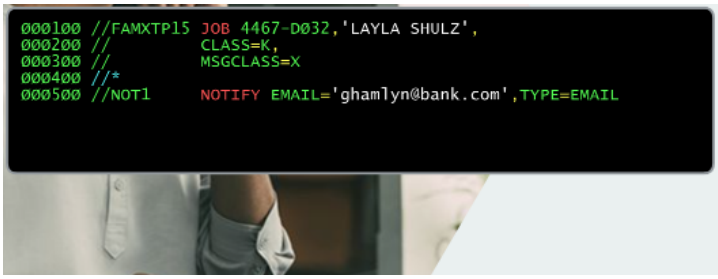
Use &SYSUID - for anyone to use your JCL instead of specifying a specific TSO UserID.



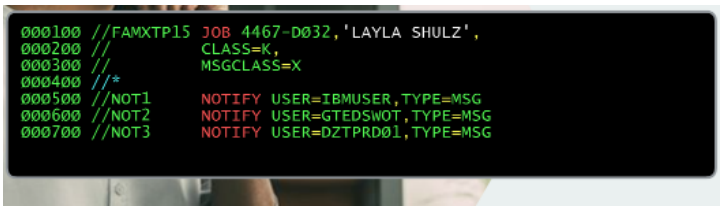
The NOTIFY parameter is quite simple but has a few drawbacks, including the following:

- If the user that the message is to be sent to is not logged on, they will not receive a job completion message until they next log on
- The NOTIFY parameter can only be used to send a message to a single user
- There is no default, so if you forget to code this parameter you will need to monitor the job using other methods, to determine if it has finished

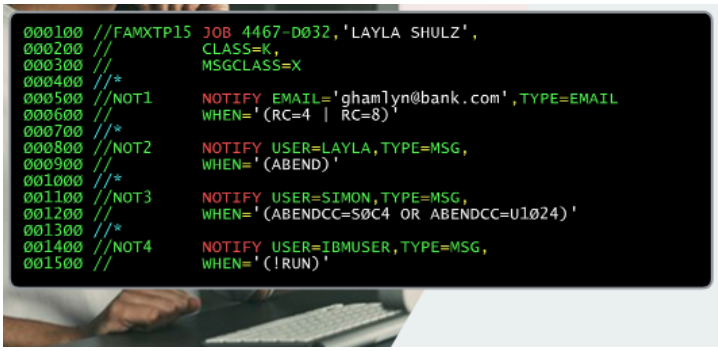
With z/OS 2.3 a new NOTIFY statement provides more flexibility in when and how job completion messages are sent. When specified, it must be placed after the JOB statement and before the first EXEC statement.



This statement name follows the standard naming syntax for other statements, as discussed previously. This is followed by at least one blank and then the statement type: NOTIFY. All parameters used in this statement are keyword, so can be specified in any order. The EMAIL parameter is used to define the email address to whom job completion details will be sent. The TYPE parameter indicates that the message is to be sent as an email message and is the default when the EMAIL parameter is used.



A maximum of eight NOTIFY statements can be specified. In this example the USER parameter is used to identify the TSO user to whom a job completion message will be sent. The TYPE parameter indicates that the notification should be sent by using a TSO message, and is the default when the USER parameter is specified.



The WHEN parameter can be used to define conditions under which notification will be performed. In the NOT1 statement, a confirmation email will be sent only if the job completes with a maximum condition code of 4 or 8. In the NOT2 statement, if an abend occurs then a message will be sent to TSO user LAYLA. The NOT3 statement is more specific as it indicates that either of the abend codes specified need to be produced before the message is sent. In the NOT4 statement if the job did not run - for example, it had a JCL syntax error - then a TSO message will be sent to TSO user IBMUSER. The exclamation mark (!) character indicates a not operation.

Additional JOB Statement Parameters

Popular Statement Parameters

TYPRUN Parameter

Normally when you submit a job, it will run immediately if the required resources are available. There may be situations though where you need the system to handle your job differently when it is submitted.

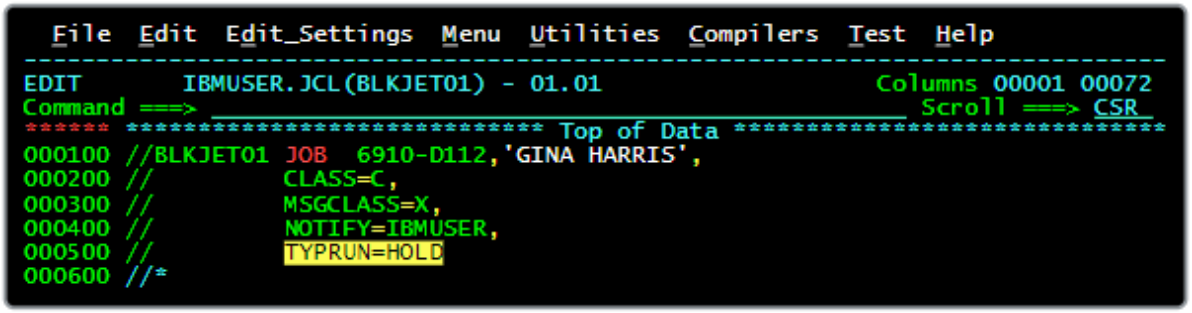
For example, the TYPRUN parameter can be used for the following purposes:

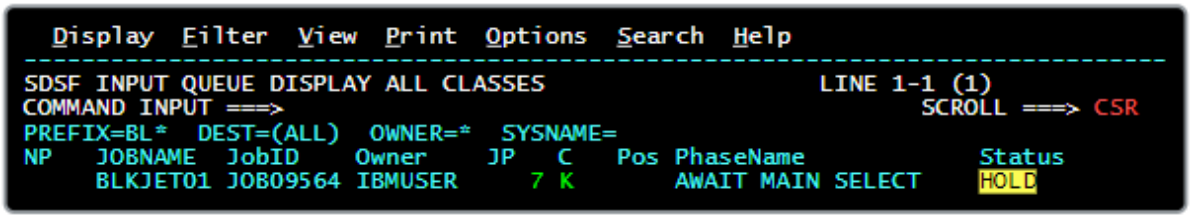
- Holding your job, pending a manual release, either by yourself or an operator.
- Scanning the syntax of your JCL. In this scenario, no actual job processing is performed.
- Sending a copy of your JCL directly to sysout, without performing any processing, in a JES2 environment only.

Scenario 1 - TYPRUN HOLD:

You are going to be detained all afternoon with meetings but need to submit a job that must be run when a file becomes available after 1:00pm. Rather than wait until you are back in the office later in the day to submit your job, you can submit it now and instruct an operator to release it.

The TYPRUN=HOLD parameter can be used for this purpose. You can see in the SDSF screen at the bottom of the page that the job has a HOLD status. Note that if your job contains syntax errors when it is submitted, it will fail immediately and not be held.

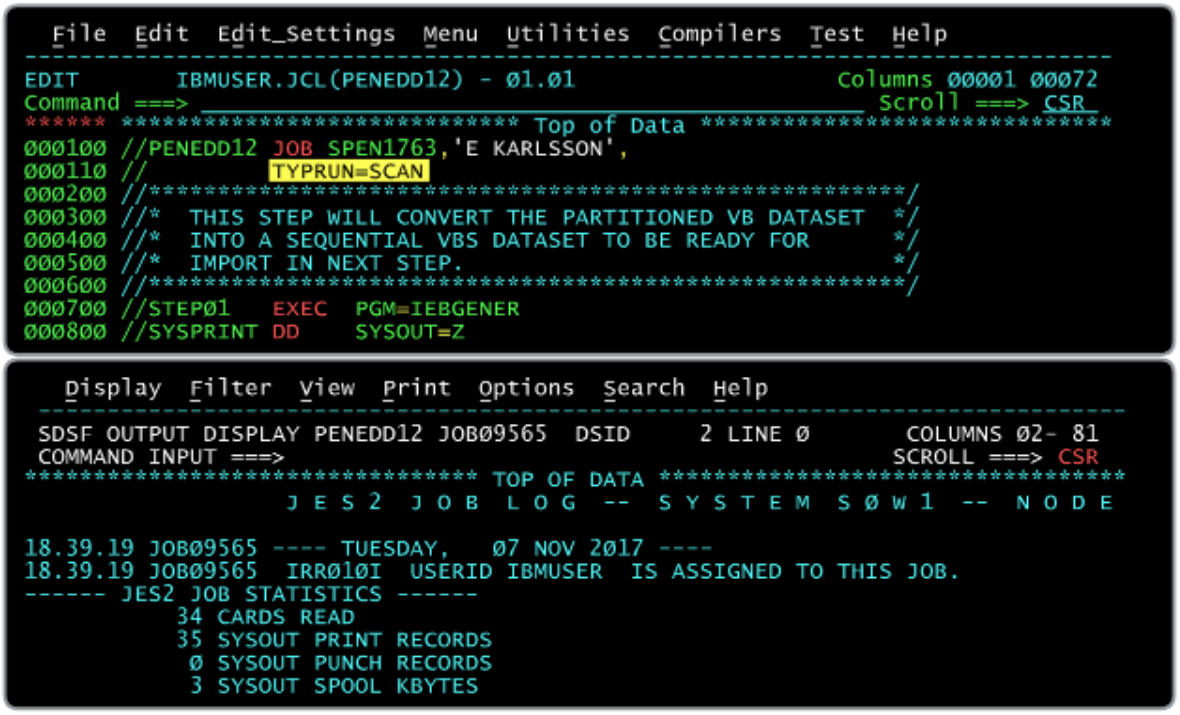




Scenario 2 - TYPRUN SCAN:

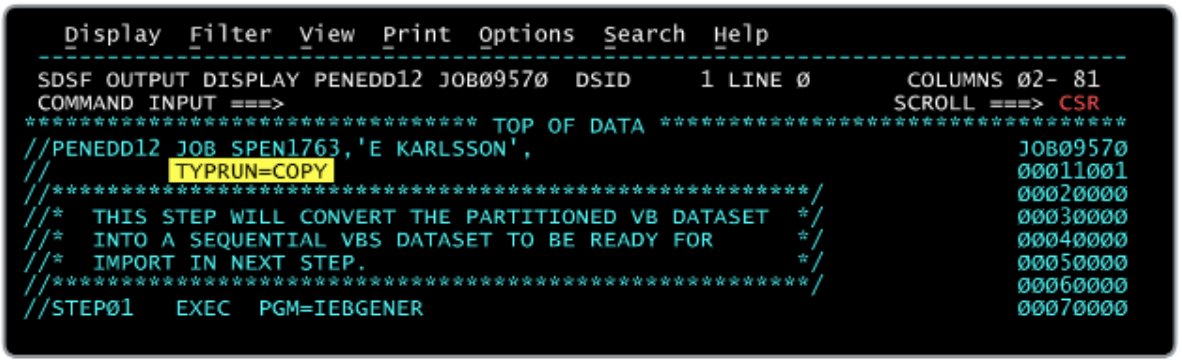
You have been copying components of JCL from other sources to build your JCL. You still need to work on the programs that will be invoked but want to check your JCL to see whether it is syntactically correct.

The TYPRUN=SCAN parameter can be used for this purpose. You can see in the SDSF screen at the bottom that if the syntax is acceptable, there are no error-related messages. Note though that the job is not run, it is only scanned for syntax issues. Although you cannot see it here, at the bottom of the SDSF output, a copy of the JCL is also produced.



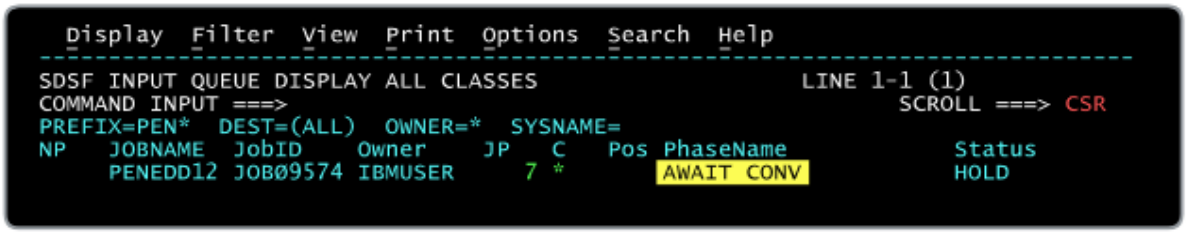
Scenario 3 - TYPRUN COPY:

TYPRUN=COPY can be used if you just need to copy your JCL directly to sysout - output that can be displayed through SDSF. If there are any syntax errors, this option will not pick them up.



Scenario 4 - TYPRUN JCLHOLD:

TYPRUN=JCLHOLD is similar to TYPRUN=HOLD except that the job is held prior to the JES2 conversion stage. You will see in a later course what happens during this stage and why this option may be useful. When the job is released by the operator, if there are any JCL syntax errors, similar output to that of the TYPRUN=SCAN will be produced, otherwise the job will begin executing if resources are available.

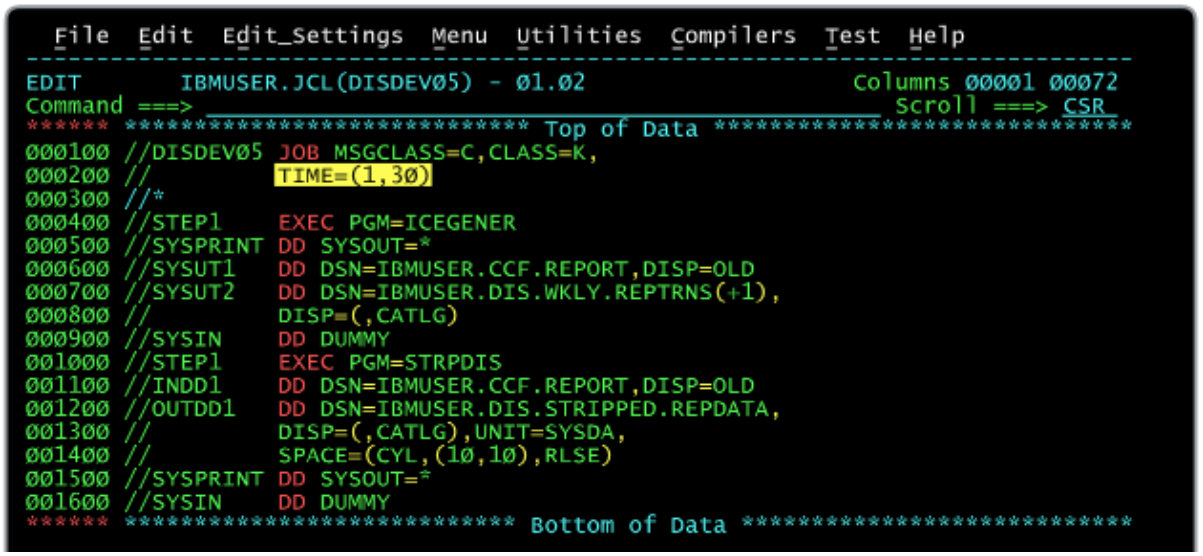


If no TYPRUN parameter is specified, then the job is submitted and will run immediately if all required resources are available.

TIME Parameter

When testing any programs you are working on, you may want to ensure that they do not run longer than you think they should, for example in the case of a program looping. A TIME parameter can be coded on the JOB statement for this purpose. This is used to specify the maximum amount of CPU time, not elapsed time, that your job can execute for.

There are several values that can be specified for this parameter. In the example shown here, two positional sub parameters can be used to specify minutes and seconds.



The first sub parameter in the TIME parameter refers to minutes. This value must be between 0 and 357913. The seconds sub parameter appears after the comma and must be a value between 0 and 59 - inclusive.

- If minutes are not required just put a comma in front of the seconds like with the accounting information and programmer name combo.
- If seconds are not required, just do TIME=2 or whatever value for the CPU minutes.
- It is possible for a job to take 1 hour to run but only consume a minute of CPU time so keep this in mind when using this parameter.

There may be several areas where your systems administrator has defined the maximum amount of time that a job can run. As well as specifying minutes and seconds as shown on the previous page, there are also several other values that can be used.

```
000100 //JETDEV01 JOB ,'B CORNES',
000200 //          CLASS=K,
000300 //          MSGCLASS=X,
000400 //          TIME=1440,
000500 //          NOTIFY=IBMUSER
000600 //S1       EXEC PGM=IEFBR14
```

```
000100 //FAMXTP15 JOB 6910-D112,'GINA HARRIS',
000200 //          CLASS=K,
000300 //          MSGCLASS=X,
000400 //          TIME=NOLIMIT
000500 //*
```

```
000100 //PENEDD12 JOB SPEN1763,'E KARLSSON',
000200 //          TIME=MAXIMUM
000300 //*************************************************************
```

COND Parameter

When your batch job runs, each step or program runs sequentially, and when it completes it produces a condition code. Following this, the next program sequentially in your JCL will be invoked, and so on. If for some reason a program completes unsuccessfully, then what usually occurs is that the job itself will fail at that point and no further processing for the job will occur.

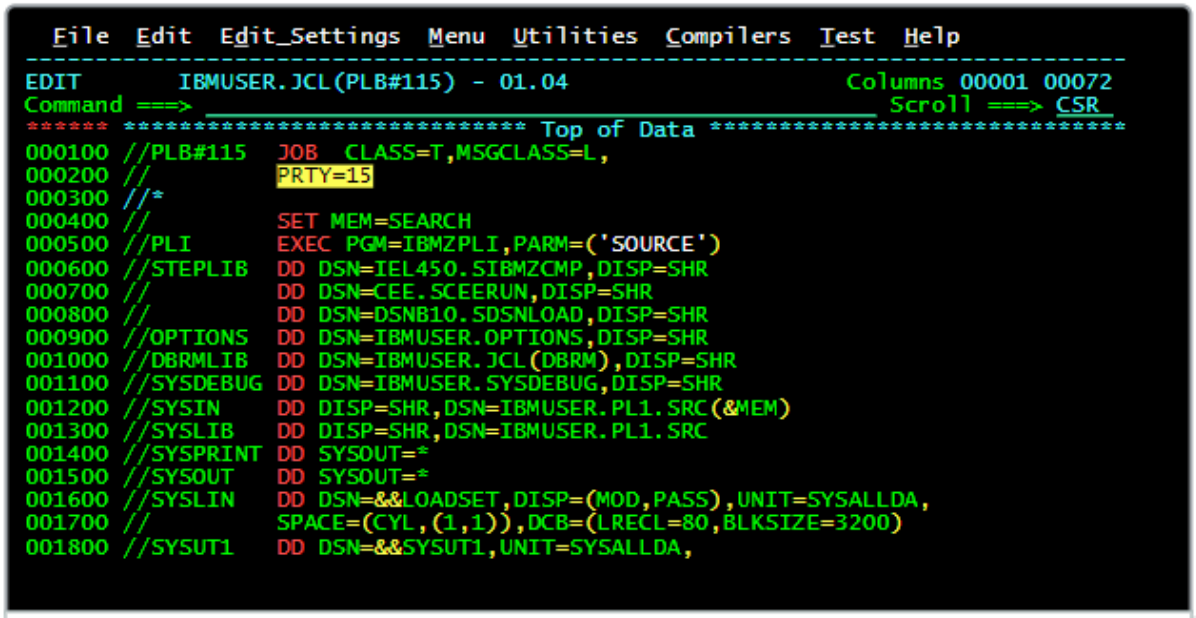
JCL allows you to be more specific about the completion circumstances for each program and whether other programs that appear later in your job should run. One method used to achieve this is the JOB statement COND parameter.

```
  File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
--------------------------------------------------------------------------------
EDIT       IBMUSER.JCL(DISDEV05) - 01.04                Columns 00001 00072
Command ===> ____________________________________________ Scroll ===> CSR
000100 //DISDEV05 JOB MSGCLASS=C,CLASS=K,
000200 //          COND=(4,LT)
000300 //*
000400 //STEP1    EXEC PGM=DISBKUP
000500 //*
000600 //STEP2    EXEC PGM=DISEXT
000700 //*
000800 //STEP3    EXEC PGM=DISSORT
000900 //*
001000 //STEP4    EXEC PGM=DISREP
001100 //*
****** ***************************** Bottom of Data ****************************
```

PRTY Parameter

You saw earlier that when a job is submitted, if an initiator is available then the job will begin executing. Though what happens if there are several jobs waiting to be processed and you need yours to run first once an initiator becomes available?

One of the other tasks performed by JES is to set a priority for the job. In most cases as a general user, you will not have much control of job priority, because if you did then everyone would be promoting their job to the highest level. Having said that, there is a PRTY parameter that can be coded on the JOB statement that allows you, if authorized, to define a priority for your job. In the scenario here your job would appear as a higher order on the input queue than others.



For a JES2 site, the value for this parameter is from 0 through 15, with the highest priority being 15. In a JES3 environment, the range is 0 through 14.

Depending on your organizational standards, if you have coded this parameter the system may just ignore it.

Other JOB Statement Parameters

RESTART Parameter

Normally when you submit a job, it will run from top to bottom, executing programs sequentially. However, what happens when your five-program job processes the first two programs successfully, and after 30 minutes of elapsed time processing them, fails in the third program because a data set you were trying to create already existed. To rerun it from the beginning would waste 30 minutes of previously successful processing. In this situation you may be able to use the JOB statement RESTART parameter.

In the example shown here, the RESTART parameter will begin from a step named STEP3. When the job is submitted, the DISBKP and DISEXT programs will not be run, and processing will begin from STEP3, which invokes the DISSORT program.

```
 File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
--------------------------------------------------------------------------------
EDIT       IBMUSER.JCL(DISDEV05) - 01.05                Columns 00001 00072
Command ===> ____________________________________________ Scroll ===> CSR
****** ***************************** Top of Data ******************************
000100 //DISDEV05 JOB MSGCLASS=C,CLASS=K,
000200 //          RESTART=STEP3
000300 //*
000400 //STEP1     EXEC PGM=DISBKUP
000500 //*
000600 //STEP2     EXEC PGM=DISEXT
000700 //*
000800 //STEP3     EXEC PGM=DISSORT
000900 //*
001000 //STEP4     EXEC PGM=DISREP
001100 //*
****** **************************** Bottom of Data ****************************
```

Memory-Related Parameters:

The REGION parameter is used to specify the maximum amount of memory that can be used by programs within the job. However, there are several other memory-related parameters that can be used to further refine memory requirements.

```
000100 //PENEDD12 JOB SPEN1763,'E KARLSSON',
000200 //          REGIONX=(4M,256M)
000300 //*************************************************************/
```

```
000100 //SUPAPD05 JOB T112-345,'ELIZA BROWN',
000200 //          ADDRSPC=REAL
000300 //STEP1     EXEC PGM=JDEDXX
```

```
000100 //A#PENXT  JOB MSGCLASS=C,
000200 //          MEMLIMIT=10000M
000300 //*
```

- REGIONX can be used if one already knows how much memory above and below the 16MB line is required
- By default, programs will utilize virtual storage which is pageable. This can be altered so all programs in the job use real, non-pageable storage.
- This last one places a limit on the total size of usable virtual storage above the bar in a single address space.

Output-Related Parameters:

Similar to MSGCLASS and MSGLEVEL parameters and how they are used when producing system output there are several other output-related parameters that can be used on the JOB statement to specifically control the maximum amount of output that should be produced. This is useful if your job creates more output than you expected. Note that like other parameters discussed, you will have an installation default for it, so coding this parameter is only useful if it varies from this value.

```
 File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
--------------------------------------------------------------------------------
EDIT       IBMUSER.JCL(AASORT) - 01.03                 Columns 00001 00072
Command ===>                                             Scroll ===> CSR
****** ***************************** Top of Data ******************************
000100 //AASORT   JOB  6910-D112,'GINA HARRIS',
000200 //         CLASS=C,MSGCLASS=X,
000300 //         LINES=(10,WARNING)
000400 //*
000500 //STEP1    EXEC PGM=SORT
000600 //SYSOUT   DD SYSOUT=*
000700 //SORTIN   DD DSN=IBMUSER.PEN.DAILY.TRANS,DISP=OLD
```

```
 Display  Filter  View  Print  Options  Search  Help
--------------------------------------------------------------------------------
SDSF OUTPUT DISPLAY AASORT   JOB0961Ø  DSID     2 LINE Ø       COLUMNS 2Ø- 99
COMMAND INPUT ===>                                           SCROLL ===> CSR
********************************* TOP OF DATA ************************************
J E S 2  J O B  L O G  --  S Y S T E M  S Ø W 1  --  N O D E  S V S C J E S 2

---- WEDNESDAY, Ø8 NOV 2Ø17 ----
IRRØ1ØI  USERID IBMUSER  IS ASSIGNED TO THIS JOB.
ICH7ØØØ1I IBMUSER  LAST ACCESS AT 16:14:57 ON WEDNESDAY, NOVEMBER 8, 2Ø17
$HASP373 AASORT   STARTED - INIT 1    - CLASS K        - SYS SØW1
$HASP375 AASORT   ESTIMATED  LINES EXCEEDED
$HASP375 AASORT   ESTIMATE EXCEEDED BY              1Ø,ØØØ  LINES
$HASP375 AASORT   ESTIMATE EXCEEDED BY              2Ø,ØØØ  LINES
$HASP375 AASORT   ESTIMATE EXCEEDED BY              3Ø,ØØØ  LINES
$HASP375 AASORT   ESTIMATE EXCEEDED BY              4Ø,ØØØ  LINES
```

The JCL at the top of this screen contains a LINES parameter, to indicate if more than ten (thousand) lines of output are produced, then a warning message will be sent to the operator. The job will continue to process the output for the job and an operator message will continue to be sent at regular intervals. The SDSF output for the job at the bottom of the screen shows these messages being produced, which are also sent to the operator.

- The warning parameter can be replaced with DUMP or CANCEL
    - DUMP will cancel the job once the maximum number of lines have been produced and request a system dump.
    - CANCEL will cancel the job without a dump.

```
000100 //BLKJET01 JOB  6910-D112,'GINA HARRIS',
000200 //         CLASS=C,
000300 //         MSGCLASS=X,
000400 //         NOTIFY=IBMUSER,
000500 //         PAGES=(20,WARNING)
```

```
000001 //CSFSMFJ  JOB  (D-117),'GMH RUN',
000002 //         CLASS=A,
000003 //         MSGCLASS=X,
000004 //         BYTES=(5,DUMP)
```

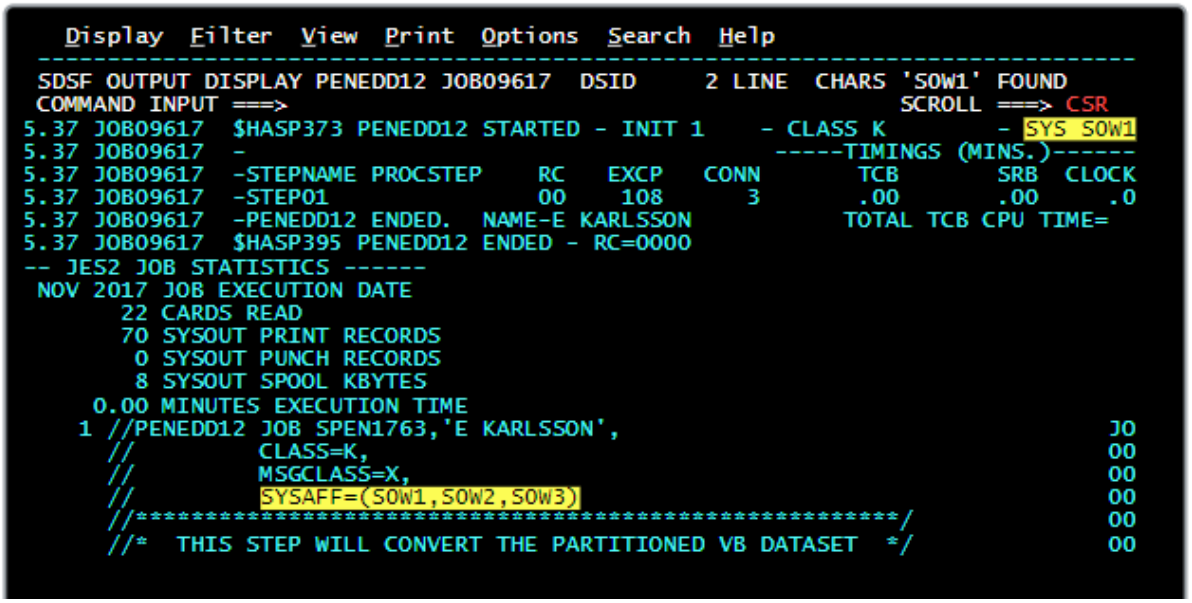
```
000100 //DISDEV05 JOB MSGCLASS=C,
000200 //         CLASS=K,
000300 //         CARDS=(500,CANCEL)
```

The LINES parameter can be substituted with the following, using a different measurement to determine the maximum amount of output to be produced:
- PAGES is the maximum number of pages of output that can be produced before action is taken.
- BYTES is the maximum number of bytes, in thousands, that can be produced before action is taken.
- CARDS is the maximum number of cards that can be punched before action is taken.

SYSAFF Job Processing Environment Parameter

When the submit command is issued for a job, the system assumes that the job is to be run on the system where it is being submitted from. If resources are not available on that system, then the job may sit and wait a considerable time before it executes.

There are several JOB statement parameters that can be used to specify that the job is run on a more appropriate system within the organization. For example, the SYSAFF parameter shown here can be used to list the JES2 members, or JES3 systems, that are allowed to process this job.



As seen in the image above, this job output was eligible on JES2 members S0W1, S0W2 or S0W3. At the top of this output we can see that it was run on S0W1.

Several other types of values can be coded in the SYSAFF parameter such as:

- ~S0W3 which means the job can be processed on any member except S0W3
- All which means that any system can process the job

A maximum of 33 member names can be specified with this parameter.

SYSTEM Job Processing Environment Parameter

Similar to the SYSAFF parameter, the SYSTEM parameter defines the systems where the job can be processed. Up to 32 eight-character system names can be coded. A preceding hyphen (-) character can be used to identify that the job is not allowed to be processed on the specified system.

The ANY value indicates that it can run on any eligible system, while values of JGLOBAL - global processor only, and JLOCAL - local processor only, are used for JES3 environments.

```
 File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
--------------------------------------------------------------------------------
EDIT       IBMUSER.JCL(PLB#115) - 01.06                Columns 00001 00072
Command ===> ____________________________________________ Scroll ===> CSR
****** ***************************** Top of Data ********************************
000100 //PLB#115  JOB  CLASS=T,
000200 //         MSGCLASS=L,
000210 //         SYSTEM=(CHIGTEST,NEWYTEST,SEATTEST)
000300 //*
000400 //         SET MEM=SEARCH
000500 //PLI      EXEC PGM=IBMZPLI,PARM=('SOURCE')
000600 //STEPLIB  DD DSN=IEL45O.SIBMZCMP,DISP=SHR
000700 //         DD DSN=CEE.SCEERUN,DISP=SHR
000800 //         DD DSN=DSNB1O.SDSNLOAD,DISP=SHR
000900 //OPTIONS  DD DSN=IBMUSER.OPTIONS,DISP=SHR
001000 //DBRMLIB  DD DSN=IBMUSER.JCL(DBRM),DISP=SHR
001100 //SYSDEBUG DD DSN=IBMUSER.SYSDEBUG,DISP=SHR
001200 //SYSIN    DD DISP=SHR,DSN=IBMUSER.PL1.SRC(&MEM)
001300 //SYSLIB   DD DISP=SHR,DSN=IBMUSER.PL1.SRC
001400 //SYSPRINT DD SYSOUT=*
001500 //SYSOUT   DD SYSOUT=*
001600 //SYSLIN   DD DSN=&&LOADSET,DISP=(MOD,PASS),UNIT=SYSALLDA,
001700 //         SPACE=(CYL,(1,1)),DCB=(LRECL=80,BLKSIZE=3200)
```

SCHENV Job Processing Environment Parameter

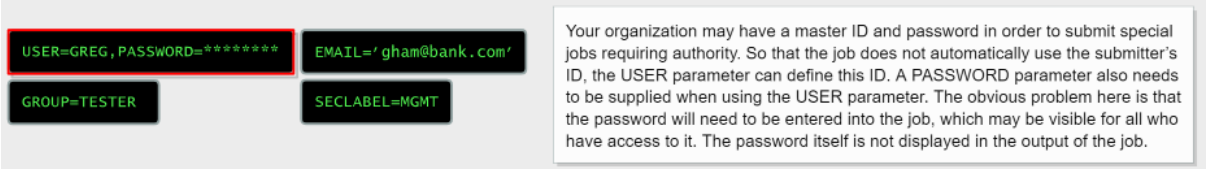
In your Workload Management (WLM) policy you may have defined a number of scheduling environments. A scheduling environment consists of a set of resources whose state can be set depending on the system they reside on. For example, you could have a scheduling environment named DBBAT that requires a resource representing the Db2 database being available to be set to OFF, and that a batch processing window resource be set to ON. When these resources meet this requirement, the DBBAT scheduling environment is available.

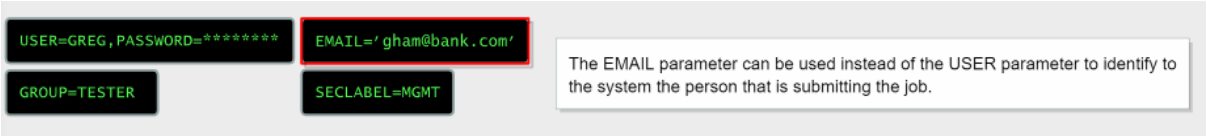
```
 File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
--------------------------------------------------------------------------------
EDIT       IBMUSER.JCL(DBBAT20) - 01.02                Columns 00001 00072
Command ===> ____________________________________________ Scroll ===> CSR
****** ***************************** Top of Data ********************************
000001 //DBBAT20  JOB CLASS=K,
000002 //         MSGCLASS=A,
000003 //         NOTIFY=IBMUSER,
000004 //         SCHENV=DBBAT
000005 //*
000006 //ALTERTAB EXEC PGM=IKJEFT01,DYNAMNBR=20
000007 //SYSTSPRT DD SYSOUT=*
000008 //SYSTSIN  DD *
000009  DSN SYSTEM(DB2)
000010  RUN  PROGRAM(DSNTIAD) -
000011    PLAN(DSNTIA31) -
000012    LIB('DB310.RUNLIB.LOAD')
000013 /*
000014 //SYSPRINT DD SYSOUT=*
000015 //SYSUDUMP DD SYSOUT=*
000016 //SYSIN    DD *
000017
000018  ALTER TABLE OLIBRARY ADD MEMBER CHAR(16) NOT NULL WITH DEFAULT;
```

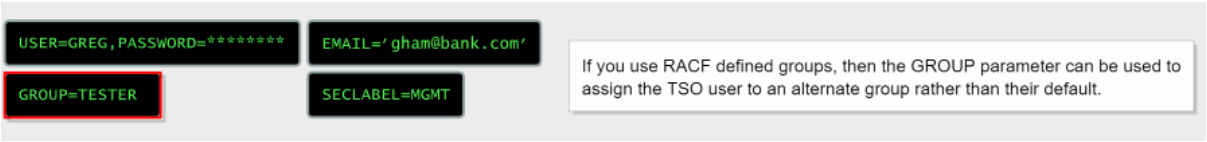
When this batch job is submitted, JES will check the scheduling environment specified and then assign the work to a system that matches that scheduling environment.

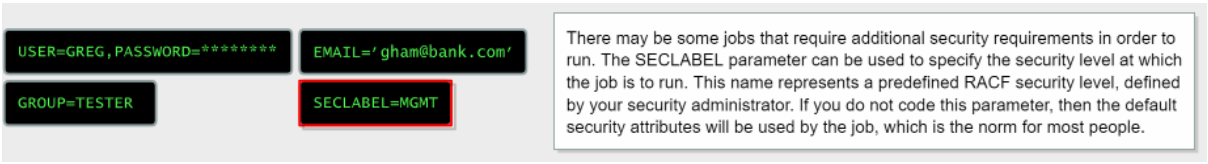
Identification-Related Parameters

Normally, the system can identify the authorization requirements for a batch job as they are taken from the user submitting the job. However, there are several other lesser used JOB statement parameters that can be used for identification and authorization purposes.









Running a Program using EXEC Statements

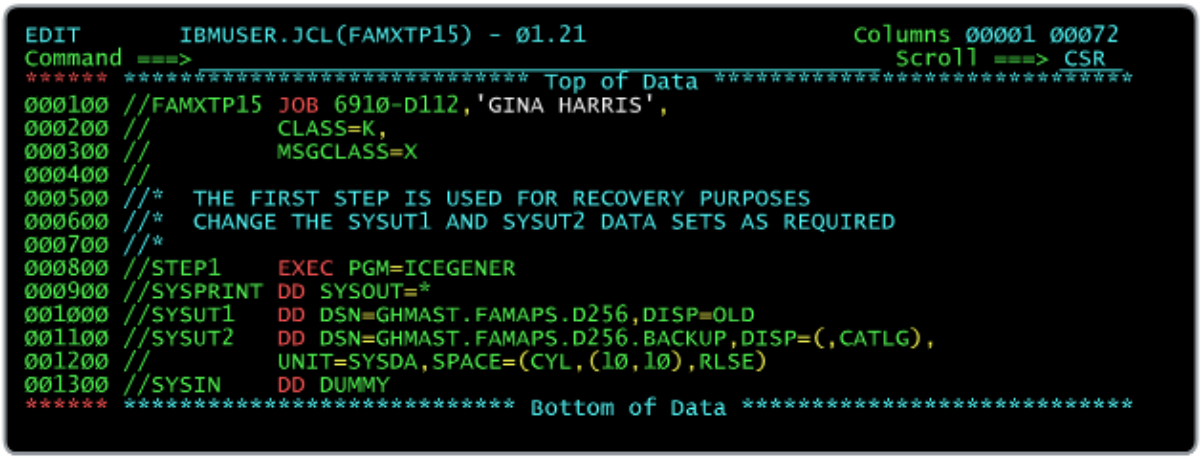
EXEC Statement Basics

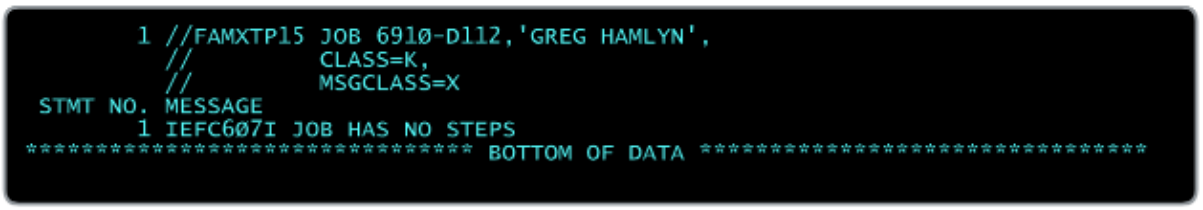
Each EXEC statement marks the beginning of that step, which ends when another EXEC statement is encountered, or it is the end of the job. As well as informing the system of the programs to run, various parameters can be passed to the program in this statement.

A job can contain as many as 255 EXEC statements and must contain at least one, but it is rare today to see a large number of steps in a job. This is because it is more difficult to manage a job like this if a restart is required.

No JOB without Steps

A job must contain at least one step, otherwise it will fail. The SDSF output shown on the bottom screen indicates this exact problem, but it looks like there are steps in the JCL, which is shown at the top (it's the empty // curse).





Statement Name - EXEC

In the example below, the EXEC statements do not have a name. This is because it is optional, as this statement's only purpose is to invoke a program.

```
000300 //          EXEC PGM=TRANAPL
000400 //INPUT     DD DSN=PROD.D112Y17.PENTRANS,DISP=OLD
000500 //OUTPUT    DD DSN=IBMUSER.TOTAL.PENTRANS,DISP=(,CATLG),
000600 //          UNIT=SYSDA,SPACE=(CYL,(10,10,RLSE))
000700 //SYSPRINT DD SYSOUT=*
000800 //SYSOUT    DD SYSOUT=*
000900 //*
001000 //          EXEC PGM=TRANXCT
001100 //SYSPRINT DD SYSOUT=*
001200 //INDD1     DD DSN=IBMUSER.TOTAL.PENTRANS,DISP=SHR
001300 //SYSIN     DD *
001400  RECORDS=10000
****** ***************************** Bottom of Data ******************************
```

However, it is good practice to use them because:

- You need them for restart statements.
- It makes debugging easier.
- May have duplication issues in the long run.
- We are also able to refer back to certain statements in previous steps. To do this, we also need a statement name.

```
 File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
--------------------------------------------------------------------------------
EDIT       IBMUSER.JCL(APENCHK) - 01.01                Columns 00001 00072
Command ===> ____________________________________________ Scroll ===> CSR
002000 //STEP3    EXEC PGM=IEBGENER
002100 //SYSUT1   DD DSN=IBMUSER.SALESIN,DISP=OLD
002200 //SYSUT2   DD DSN=IBMUSER.PEN.TRANS.PARMPEN,
002300 //          DCB=(RECFM=FB,LRECL=80,BLKSIZE=32720),
002400 //          SPACE=(CYL,(1,1),RLSE),UNIT=SYSDA,DISP=(,CATLG)
002500 //SYSPRINT DD SYSOUT=*
002600 //SYSIN    DD DUMMY
002700 //*
002800 //STEP4    EXEC PGM=IEBGENER
002900 //SYSUT1   DD DSN=*.STEP3.SYSUT2,DISP=SHR
003000 //SYSUT2   DD SYSOUT=*
003100 //SYSPRINT DD SYSOUT=*
003200 //SYSIN    DD DUMMY
****** ***************************** Bottom of Data ******************************
```

Although this has not yet been covered, you are also able to refer back to certain statements in previous steps. For example, to use the same data set created earlier in the job. In order to do this, you need to specify a step name. You can see in this example that on line 002900 the DSN parameter uses an asterisk (*) to indicate a referback, and what follows is the step name and DD statement name. It says to use the same data set used in STEP3 by DD statement SYSUT2. This is identified on line 002200. Do not worry too much about this parameter at this stage, it is just included here to highlight referencing of a step name.

EXEC Operation

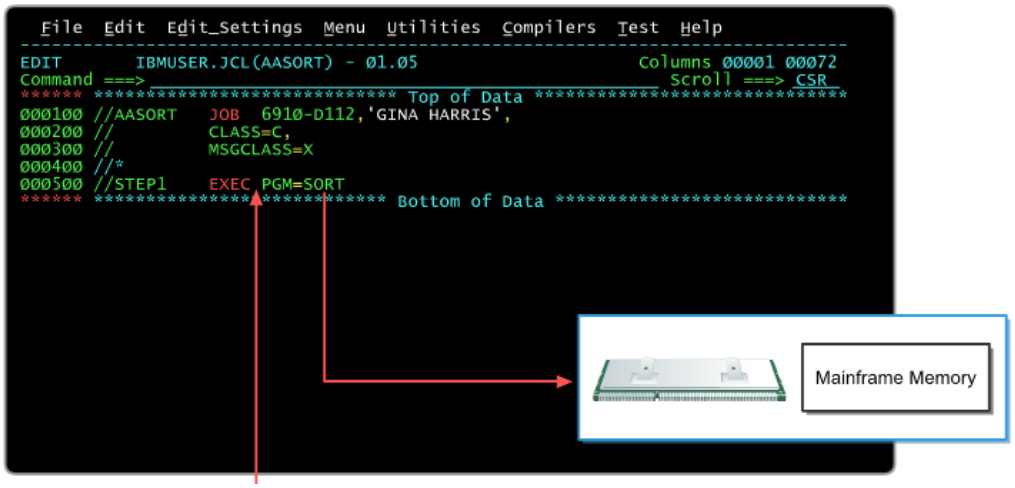
Following the statement name, at least one space needs to be coded before entering the statement type EXEC. If the name is not specified, EXEC must appear at least one blank character after the // characters.

- If there's no EXEC for a statement, an error will appear.

Programs

The next step is to tell the system the program, or programs that it needs to run.This can be as easy as coding PGM=programname. This will instruct the system to look in the system program libraries for the program specified. These programs are stored as members in partitioned data sets. Once located, the program is then loaded into memory, and begins running.

- There must be a space between EXEC and PGM.



JOBLIB and STEPLIB Statements

If you are developing or testing programs there is a good chance that they will not reside in the default program libraries that the submitted JCL uses. In these situations, you can specify a JOBLIB or STEPLIB statement to instruct the system to begin the search for programs in these libraries.

- JOBLIB - When this job is run, the data specified in it - IBMUSER.LOAD will be searched first when there is any program request. If it is not located here, the default system libraries will then be searched.
- STEPLIB - this performs the same function as JOBLIB but only for a single step. In this scenario below, IGYCRCTL program will be searched for in the data set IBMUSER.MOD first. Other programs in this job will only use the default system libraries.

```
000100 //BLKJET01 JOB  6910-D112,'GINA HARRIS',
000200 //         CLASS=C,
000300 //         MSGCLASS=X
000400 //JOBLIB   DD DSN=IBMUSER.LOAD,DISP=SHR
000500 //*
000600 //STEP1    EXEC PGM=BLKSRTIT
000700 //SRTIN    DD DSN=IBMUSER.JET.DAILY.PAYMENTS,DISP=OLD
000800 //SRTOUT   DD DSN=IBMUSER.SORTED.JET.DAILY.PAYMENTS,DISP=(,CATLG),
000900 //         SPACE=(CYL,(1,1),RLSE),UNIT=SYSDA
001000 //REPORT   DD SYSOUT=R
001100 //*
001200 //ALTERTAB EXEC PGM=GHCHKDB,DYNAMNBR=20
```

```
000100 //OOCCOMP  JOB  D22789B,'COBOL COMPILE',CLASS=A,
000200 //         MSGCLASS=X
000300 //*
000400 //         SET MEM=OOCBOOK
000500 //COMP1    EXEC  PGM=IGYCRCTL,REGION=0M,
000600 //      PARM='LIST,MAP,SOURCE,XREF(SHORT),DLL,COMPILE,RENT,THREAD'
000700 //SYSIN    DD DSN=IBMUSER.SRC(OOCBOOK),DISP=SHR
000800 //SYSLIB   DD DISP=SHR,DSN=IBMUSER.SRC
000900 //         DD DISP=SHR,DSN=CEE.ACEESRC1
001000 //STEPLIB  DD DISP=SHR,DSN=IBMUSER.MOD
```

Program Referback

The method shown previously is the most common way that you will see a single program being invoked. There is a possibility though that your organization may use the PGM parameter for referring back to a JCL statement earlier in your job. A scenario might be that your first step creates a temporary program which then needs to be referred to in a later step.

- The '*' - in JCL the asterisk is used in several different ways; in this case it indicates a referback
- COMP - A period (.) is used to separate referback data. COMP indicates the name of the previous step that will be used to obtain the program name. If SYSLMOD had not been coded, the system would have used the program from the COMP step - IEWL.
- SYSLMOD - this is used to indicate the DD statement that contains the name of the program to be used - PROG.

```
001500 //COMP     EXEC PGM=IEWL,REGION=0M,COND=(5,LT,COMP1),
001600 //         PARM='LIST,XREF'
001700 //STEPLIB  DD DSN=IGY520.SIGYCOMP,DISP=SHR
001800 //SYSLIB   DD DISP=SHR,DSN=CEE.SCEELKED
001900 //SYSLIN   DD DISP=(OLD,DELETE),DSN=&&LOADSET
002000 //SYSLMOD  DD DISP=OLD,DSN=IBMUSER.MOD(&MEM)
002100 //SYSPRINT DD SYSOUT=A
002200 //SYSUT1   DD UNIT=SYSDA,SPACE=(1024,(200,20))
002300 //GO       EXEC PGM=*.COMP.SYSLMOD
```

Procedures

Another way of providing programs to be run is through a procedure. A procedure is a set of JCL statements, multiple steps usually, stored in a separate partitioned data set. They are useful if there is some common program processing that needs to be used by a number of users. In this way, the user only needs to code the name of the procedure on a single statement, rather than enter all of this code into their JCL.

```
 File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
--------------------------------------------------------------------------------
EDIT       IBMUSER.JCL(ARUNREP) - 01.05                Columns 00001 00072
Command ===> ____________________________________________ Scroll ===> CSR
****** ***************************** Top of Data ******************************
000100 //ARUNREP  JOB MSGCLASS=C,NOTIFY=IBMUSER,CLASS=K
000200 //*
000300 //RUNGH    EXEC GHPROC
****** **************************** Bottom of Data ****************************
```

```
 File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
--------------------------------------------------------------------------------
EDIT       VENDOR.PROCLIB(GHPROC) - 01.05              Columns 00001 00072
Command ===> ____________________________________________ Scroll ===> CSR
****** ***************************** Top of Data ******************************
000100 //STEP1    EXEC PGM=ALBKALL,REGION=4M
000200 //SYSPRINT DD SYSOUT=*
000300 //SYSIN    DD DISP=SHR,DSN=VENDOR.PARMLIB(GHPCKS1)
000400 //OUTDD    DD DSN=&&VOLEUR,DISP=(,PASS),
000500 //         UNIT=SYSDA,
000600 //         SPACE=(TRK,(5,5),RLSE),
000700 //         DCB=(LRECL=80,RECFM=FB)
000800 //*
000900 //STEP2    EXEC PGM=IEBGENER
001000 //SYSIN    DD DUMMY
001100 //SYSPRINT DD SYSOUT=*
```

Calling a Procedure

There are two ways of calling a procedure, using the PROC parameter and specifying the procedure name. Both of these methods perform the same task.

```
 File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
--------------------------------------------------------------------------------
EDIT       IBMUSER.JCL(ARUNREP) - 01.05                Columns 00001 00072
Command ===> _____________________________________________ Scroll ===> CSR
****** ***************************** Top of Data ******************************
000100 //ARUNREP  JOB MSGCLASS=C,NOTIFY=IBMUSER,CLASS=K
000200 //*
000300 //RUNGH    EXEC GHPROC
****** **************************** Bottom of Data ****************************
```

```
 File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
--------------------------------------------------------------------------------
EDIT       IBMUSER.JCL(ARUNREP) - 01.06                Columns 00001 00072
Command ===> _____________________________________________ Scroll ===> CSR
****** ***************************** Top of Data ******************************
000100 //ARUNREP  JOB MSGCLASS=C,NOTIFY=IBMUSER,CLASS=K
000200 //*
000300 //RUNGH    EXEC PROC=GHPROC
****** **************************** Bottom of Data ****************************
```

Locating a Procedure

Just as programs are searched for in default system libraries when they are referenced, so too are procedures. But with these items, system procedure libraries are searched and if the procedure you are looking for is not found, then the messages highlighted here will be produced.

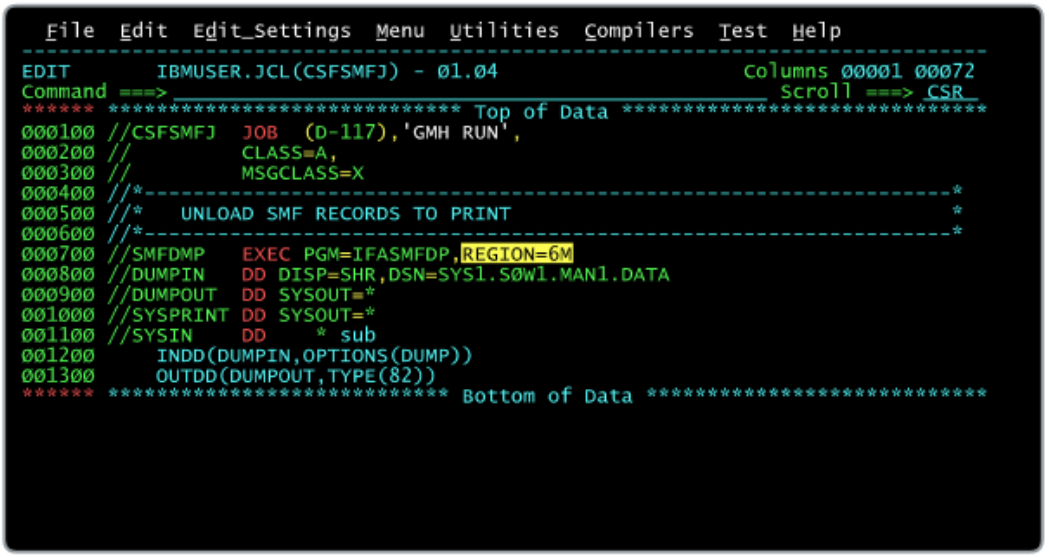
```
  Display  Filter  View  Print  Options  Search  Help
--------------------------------------------------------------------------------
SDSF OUTPUT DISPLAY JNAPAN30 JOB09688  DSID     2 LINE 0       COLUMNS 02- 81
COMMAND INPUT ===>                                            SCROLL ===> CSR
******************************** TOP OF DATA *********************************
                 J E S 2  J O B  L O G  --  S Y S T E M  S 0 W 1  --  N O D E

00.06.21 JOB09688 ---- MONDAY,    13 NOV 2017 ----
00.06.21 JOB09688  IRR010I  USERID IBMUSER  IS ASSIGNED TO THIS JOB.
00.06.21 JOB09688  IEFC452I JNAPAN30 - JOB NOT RUN - JCL ERROR  770
------ JES2 JOB STATISTICS ------
           3 CARDS READ
          17 SYSOUT PRINT RECORDS
           0 SYSOUT PUNCH RECORDS
           1 SYSOUT SPOOL KBYTES
        0.00 MINUTES EXECUTION TIME
       1 //JNAPAN30 JOB MSGCLASS=C,NOTIFY=IBMUSER,CLASS=K
         //*
       2 //JNSTEP1  EXEC PROC=JNENG05
 STMT NO. MESSAGE
       2 IEFC612I PROCEDURE JNENG05 WAS NOT FOUND
****************************** BOTTOM OF DATA ********************************
```

EXEC Statement Parameters

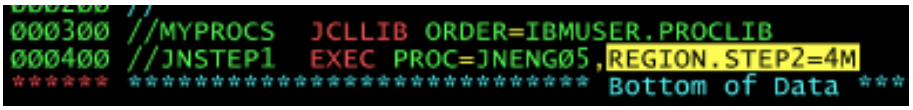
REGION Parameter

As seen previously, the REGION parameter can be specified on the JOB statement, and how it defines the maximum amount of memory to be used by any program in your job. In some situations though, you may only need to specify a memory requirement for one of the programs in your job.

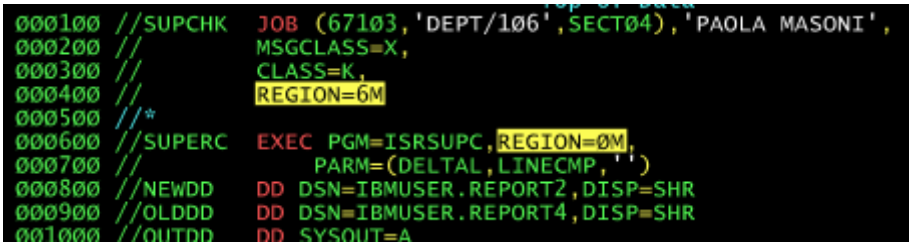
The REGION parameter can be used on the EXEC statement as shown here, and uses the same syntax as discussed previously



- When using this parameter when specifying a procedure, then all programs within the procedure will use the REGION specification.
- If you only need to provide a memory requirement for a specific step within a procedure then this can be coded as shown below, it is rare to do this. In this example, the procedure step name is provided after coding a period (.) after REGION.

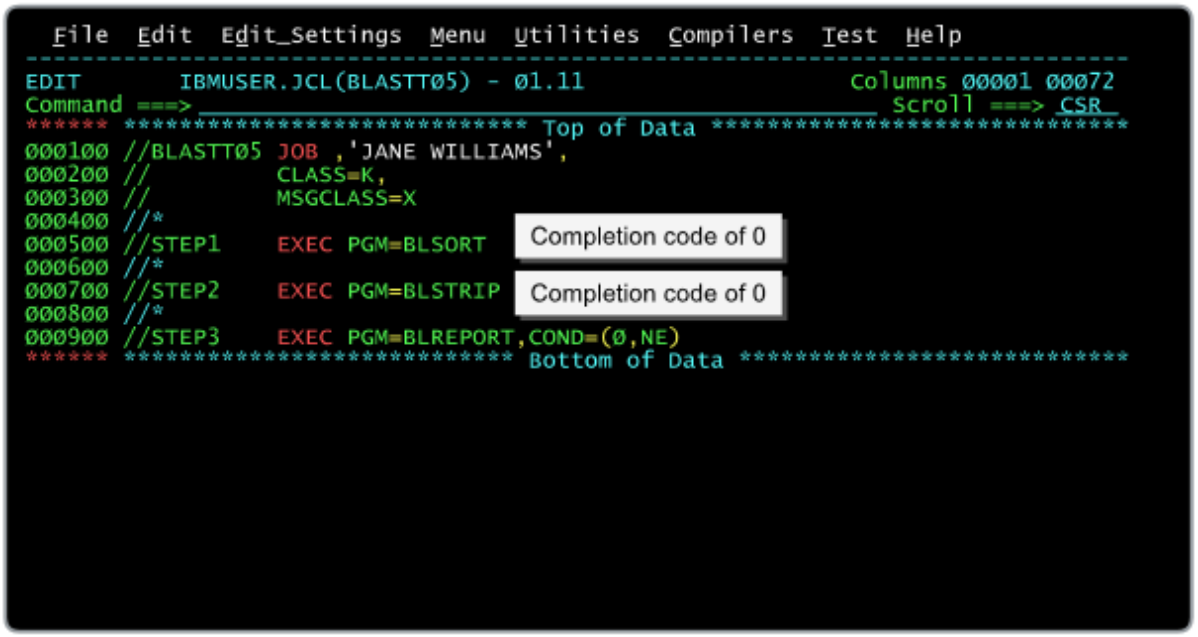


- In the example below, a REGION parameter is coded on both JOB and EXEC statements. This could arise if parts of the JCL have been copied from different sources. If this scenario occurs, the JOB statement REGION parameter will override any EXEC statement equivalent.



COND Parameter

The COND parameter can also be used on the EXEC statement, but is used more specifically to identify whether the step should be run.



The example displayed here shows how a simple COND parameter is used. There are three steps defined, with the last one containing a COND parameter. After the first two steps are run - assuming they both complete with a return code of zero, successful - your last step is a management report and should only run if all preceding steps have a zero-condition code.

The COND parameter can be quite confusing to determine sometimes how it is processing step condition codes. The example used on the previous page was a relatively simple one.

```
000700 //STEP2    EXEC PGM=BLSTRIP
000800 //*
000900 //STEP3    EXEC PGM=BLREPORT,COND=(0,NE)
```

```
000700 //STEP2    EXEC PGM=BLSTRIP
000800 //*
000900 // IF (RC=0) THEN
001000 //STEP3    EXEC PGM=BLREPORT
001100 // ENDIF
```

IBM have recognized the difficulty users have with interpreting this parameter, and recommend using an IF, THEN, ELSE, ENDIF structure, which will be more familiar to programmers.

PARM Parameter

There will be times when you need to pass additional information to the program in your EXEC statement. For example, a program you have written may perform several tasks with data that is input to it, but not necessarily every time the program is run. In this scenario some parameter information can be passed to the program, which the program will use in its processing.

The EXEC statement PARM parameter can be used for this purpose.

```
  File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
--------------------------------------------------------------------------------
EDIT       IBMUSER.JCL(COBCOMP) - 01.99                Columns 00001 00072
Command ===> ____________________________________________ Scroll ===> CSR
****** ***************************** Top of Data ******************************
000100 //IBMUCOB  JOB  123,'COBOL COMPILE',CLASS=A,MSGCLASS=X,
000200 //          NOTIFY=IBMUSER
000300 //*
000400 //          SET MEM=COMPUTE2
000500 //COMP1     EXEC  PGM=IGYCRCTL,
000600 //          PARM='LIST,XREF'
000700 //STEPLIB   DD DSN=IGY520.SIGYCOMP,DISP=SHR
000800 //SYSIN     DD DISP=SHR.DSN=IBMUSER.COBOL.SRC(&MEM)
000900 //SYSLIB    DD DISP=SHR.DSN=IBMUSER.COBOL.SRC
001000 //          DD DISP=SHR.DSN=CEE.ACEESRC1
001100 //SYSPRINT DD SYSOUT=B
001200 //SYSLIN    DD DISP=(MOD,PASS),DSN=&&LOADSET,SPACE=(80,(10,10)),
001300 //          UNIT=SYSDA
001400 //SYSUT1    DD SPACE=(CYL,(1,1)),UNIT=SYSDA
001500 //SYSUT2    DD SPACE=(CYL,(1,1)),UNIT=SYSDA
001600 //SYSUT3    DD SPACE=(CYL,(1,1)),UNIT=SYSDA
001700 //SYSUT4    DD SPACE=(CYL,(1,1)),UNIT=SYSDA
001800 //SYSUT5    DD SPACE=(CYL,(1,1)),UNIT=SYSDA
```

Note: The program must explicitly be written to accept and process the PARM information passed to it. As a result, the format and types of PARM data that can be passed must be documented.

Syntax Rules:

- What is the maximum length of PARM data?
    - Cannot exceed 100 characters
- What if PARM data will not fit entirely on one line?
    - You can extend the PARM data by ensuring that the sub parameters are enclosed in parentheses. At the end of a sub parameter definition, code a comma, and begin the continuation on the following line between columns 4 and 6, inclusive.

```
000400 //PLI      EXEC PGM=IBMZPLI,PARM=(SOURCE,ZZED,'JK;22',GLOBAL,
000500 //         'CULLDATE - 2018/02/11',KD30)
```

- How do you separate sub parameters that are required in PARM data?
    - Separate them using commas and they must be enclosed in parentheses or single quotes

```
000500 //COMP1    EXEC  PGM=IGYCRCTL,
000600 //         PARM='LIST,XREF'
```

- What if you need to pass PARM data to one program within a procedure
    - The PARM.procedurestep can be used to specify that the PARM data is only passed to the specified step in that procedure. If the procedure step name is not used, then the PARM data will only be passed to the first step in the procedure.

```
000300 //INSTP01  EXEC PROC=STPOOLBK,
000400 //         PARM.STEP2=(LIST,XREF)
```

- What if PARM data contains special characters of blanks
    - It needs to be enclosed in single quotes
- What if you enter incorrect PARM data?
    - The program must be defined to interpret and act on the PARM data if it is specified. If you enter incorrect data, then often the program will be designed to just ignore it.

PARMDD Parameter

Rather than having to code your parameter information directly into the EXEC statement, you can also use a PARMDD parameter for this purpose. This instructs the system to obtain parameter information from another statement within the JCL. This statement may contain a reference to a data set that contains this data.

One benefit of using this in preference to the PARM parameter is that more than 100 characters can be specified. Note that the PARMDD parameter cannot be used with the PARM parameter on the same statement.

```
 File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
--------------------------------------------------------------------------------
EDIT       IBMUSER.JCL(OOCBKUP) - 01.05                Columns 00001 00072
Command ===>                                             Scroll ===> CSR
****** ***************************** Top of Data ******************************
000100 //OOCCOMP   JOB  D22789B,'COBOL COMPILE',CLASS=A,
000200 //          MSGCLASS=X,TYPRUN=SCAN
000300 //*
000400 //          SET MEM=OOCBOOK
000500 //COMP1     EXEC  PGM=IGYCRCTL,PARMDD=OOCPARMS
000600 //STEPLIB   DD DISP=SHR,DSN=IBMUSER.MOD
000700 //          DD DISP=SHR,DSN=CEE.SCEERUN2
000800 //SYSIN     DD DSN=IBMUSER.COBOL.SRC(OOCBOOK),DISP=SHR
000900 //SYSLIB    DD DISP=SHR,DSN=IBMUSER.COBOL.SRC
001000 //          DD DISP=SHR,DSN=CEE.ACEESRC1
001100 //OOCPARMS  DD DSN=VENDOR.PARMLIB(OOCPARMS),DISP=SHR
001200 //*
001300 //SYSJAVA   DD PATH='/u/ibmuser/java/elpsorttbl.java',
001400 //          PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
001500 //          PATHMODE=SIRWXU,
001600 //          FILEDATA=TEXT
001700 //SYSPRINT DD    SYSOUT=*
001800 //SYSLIN DD DSNAME=&&OBJECT(TSTOOC),UNIT=SYSALLDA,DISP=(MOD,PASS),
```

Defining Symbol Values

Another item you may encounter on an EXEC statement is a symbol. These are character strings that represent variable information within the JCL. The screen at the top shows a WORK variable being passed to a procedure. In the SDSF output for the job, you can see where this value has been substituted.

```
000100 //ARUNREP  JOB MSGCLASS=C,NOTIFY=IBMUSER,CLASS=K
000200 //*
000300 //INSTP01  EXEC PROC=GHPENBK,WORK='5,5'
```

```
1 //ARUNREP  JOB MSGCLASS=C,NOTIFY=IBMUSER,CLASS=K
  //*
2 //INSTP01  EXEC PROC=GHPENBK,WORK='5,5'
3 XXGHPENBK  PROC
4 XXSTEP1    EXEC PGM=SORT
5 XXSYSOUT   DD SYSOUT=*
6 XXSORTIN   DD DSN=IBMUSER.PEN.TRANS.DAILY,DISP=OLD
7 XXSORTOUT  DD DSN=IBMUSER.SORTED.PEN.TRANZ,DISP=(,CATLG),
  XX         SPACE=(CYL,(10,10),RLSE),UNIT=SYSDA
8 XXSORTWK01 DD UNIT=SYSALLDA,SPACE=(CYL,(&WORK))
  IEFC653I SUBSTITUTION JCL - UNIT=SYSALLDA,SPACE=(CYL,(5,5))
9 XXSORTWK02 DD UNIT=SYSALLDA,SPACE=(CYL,(&WORK))
  IEFC653I SUBSTITUTION JCL - UNIT=SYSALLDA,SPACE=(CYL,(5,5))
```

TIME Parameter

This parameter can also be used on an EXEC statement, but only to specify the maximum amount of CPU time for that specific step.

```
 File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
--------------------------------------------------------------------------------
EDIT       IBMUSER.JCL(PLB#115) - 01.08                Columns 00001 00072
Command ===> _____________________________________________ Scroll ===> CSR
****** ***************************** Top of Data ******************************
000100 //PLB#115  JOB  CLASS=K,
000200 //          MSGCLASS=L
000300 //*
000400 //          SET MEM=SEARCH
000500 //PLI       EXEC PGM=IBMZPLI,PARM=(SOURCE,LIST),TIME=(,20)
000600 //STEPLIB   DD DSN=IEL450.SIBMZCMP,DISP=SHR
000700 //OPTIONS   DD DSN=IBMUSER.OPTIONS,DISP=SHR
000800 //DBRMLIB   DD DSN=IBMUSER.JCL(DBRM),DISP=SHR
000900 //SYSDEBUG  DD DSN=IBMUSER.SYSDEBUG,DISP=SHR
001000 //SYSIN     DD DISP=SHR,DSN=IBMUSER.PL1.SRC(&MEM)
001100 //SYSLIB    DD DISP=SHR,DSN=IBMUSER.PL1.SRC
001200 //SYSPRINT  DD SYSOUT=*
001300 //SYSOUT    DD SYSOUT=*
001400 //SYSLIN    DD DSN=&&LOADSET,DISP=(MOD,PASS),UNIT=SYSALLDA,
001500 //          SPACE=(CYL,(1,1)),DCB=(LRECL=80,BLKSIZE=3200)
001600 //SYSUT1    DD DSN=&&SYSUT1,UNIT=SYSALLDA,
001700 //          SPACE=(1024,(200,50),,CONTIG,ROUND),DCB=BLKSIZE=1024
001800 //*********************************************************************
```

DD Statements

DD Statement Basics

With the majority of programs, you will need to apply its processing logic against some raw data, in order to produce meaningful output.

Using DD Statements

In the COBOL program code shown, you can see where it references SALESIN and SALESOUT. The program instructs the system to look for its input data, SALESIN, and output requirements, SALESOUT, which are required by the program in order to perform its processing.

```
000500        Environment Division.
000600        Input-Output Section.
000700        File-Control.
000800             Select Unsorted-Sales-File Assign to SALESIN
000900                                    Organization is Sequential.
001000             Select Sorted-Sales-File Assign to SALESOUT
001100                                    Organization is Sequential.
001200             Select Sort-Work-File Assign to DISK.
001300
001400        Data Division.
001500        File Section.
001600        FD Unsorted-Sales-File
```

```
000300 //RUN1      EXEC PGM=SRTSAL,REGION=0M
000310 //*
000400 //SALESIN   DD DSN=IBMUSER.JUNE.SALESIN,DISP=SHR
000500 //SALESOUT  DD DSN=IBMUSER.JUNE.SALESOUT,DISP=OLD
```

In the JCL example, the COBOL program - now compiled as executable program SRTSAL - is invoked through the EXEC statement, and the SALESIN and SALESOUT JCL DD statements are used to link the input and output data sets required by the program.

This means that the program's logic can be applied to different data, for example, a different month, by just changing the JCL DD statement's data reference.

DD Statement Name:

Syntax requirements include the following:

- Name must be a maximum of 8 characters and must begin with an alphabetic or national character
- Unlike the JOB and EXEC statement names, a DD statement will, in most cases, need to have a specific name. This is because the program that is invoked contains code that says to look for a specific DD name for its input and output.

Referencing the DD Statement

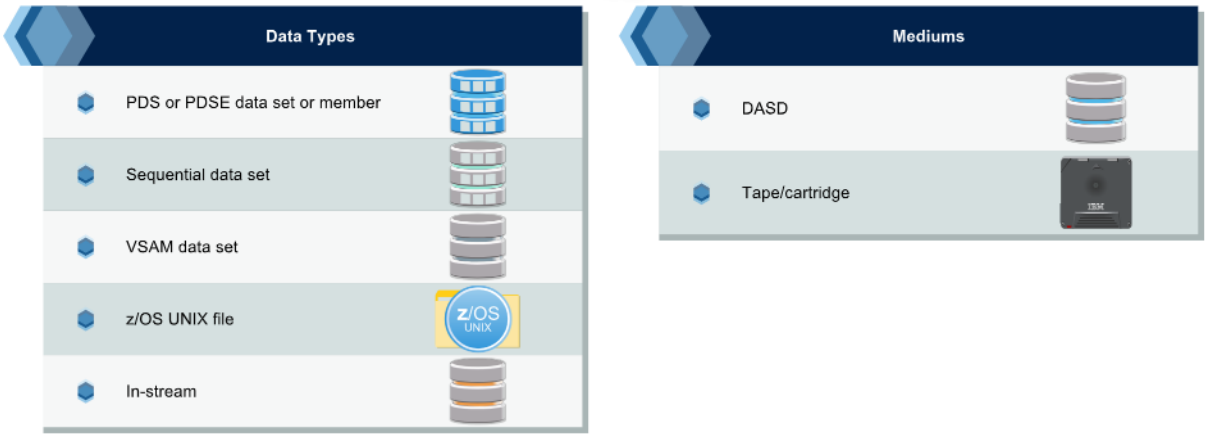
- Do you need to specify the DD statement in the same order that it appears in the program?
    - No, the program is designed to look for a specific DD name. As long as it appears in the step, then it will be able to locate it.
- What if one accidentally codes the same DD statement twice in the same step?
    - The system will use the first one that is encountered and ignore the other one.
- What happens if a required DD statement name does not appear at all in the step?
    - The job step will fail and produce a message indicating that there was a DD statement missing.
- What if you supply a DD statement that the program does not refer to?
    - The DD statement in this case will be ignored by the program, but the system will attempt to locate it, even though the program will not use it.

DD Statement Type

As seen with other statements, you need to specify the type of statement you are working with, which in this case is DD. This needs to be preceded and followed by at least one space.

Input Data Type

The data that needs to be accessed by the program may reside in a number of different types of mainframe files, also referred to as data sets. In turn, this data may reside in several different mediums such as tape/cartridge, or disk - also referred to as Direct Access Storage Devices (DASD).



These items are important to know because a DD statement will often need to contain details about the data.

DSN Parameter

Focusing on input, one of the most common DD statement parameters you will use is DSN or DSNAME. This keyword parameter is used to specify the name of the data that your program will process and can be placed anywhere in the statement. The syntax is straightforward, where the parameter is followed by an equals (=) character and then the name of the data source.

```
 File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
--------------------------------------------------------------------------------
EDIT       IBMUSER.JCL(AGENER) - 01.08                  Columns 00001 00072
Command ===> _____________________________________________ Scroll ===> CSR
****** ***************************** Top of Data ******************************
000100 //AGENER    JOB MSGCLASS=C,CLASS=K
000200 //*  STEP 1 - PEN TRANSACTION BACKUP
000300 //STEP1     EXEC PGM=IEBGENER
000400 //SYSUT1    DD DSN=IBMUSER.PEN.TRANS.DAILY
****** **************************** Bottom of Data ****************************
```

Data Name

Since you are looking at identifying the input data to your program, you can normally assume that the data already exists and is cataloged by the system. If you were to add a name, such as the one shown here, and submit your job, the system would know where that data is stored and prepare it for processing.

This example is referencing a simple sequential data set containing transaction records.

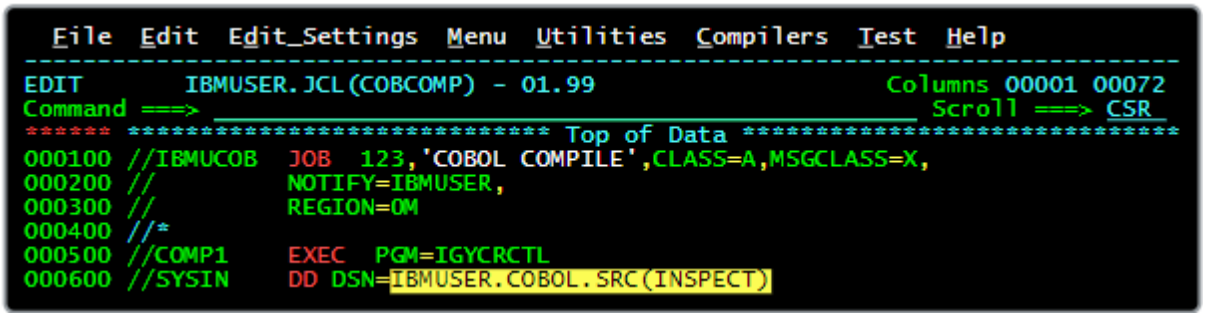
```
 File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
--------------------------------------------------------------------------------
EDIT       IBMUSER.JCL(AGENER) - 01.08                  Columns 00001 00072
Command ===> _____________________________________________ Scroll ===> CSR
****** ***************************** Top of Data ******************************
000100 //AGENER    JOB MSGCLASS=C,CLASS=K
000200 //*  STEP 1 - PEN TRANSACTION BACKUP
000300 //STEP1     EXEC PGM=IEBGENER
000400 //SYSUT1    DD DSN=IBMUSER.PEN.TRANS.DAILY
****** **************************** Bottom of Data ****************************
```

```
 File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
--------------------------------------------------------------------------------
EDIT       IBMUSER.PEN.TRANS.DAILY                      Columns 00001 00072
Command ===> _____________________________________________ Scroll ===> CSR
****** ***************************** Top of Data ******************************
000100 91163DUBE          JASON          CHICAGO       IL
000200 61535KING          ADRIANE        CINCINATTI    OH
000300 10216HOLTZ         RACHEL         NEWPORT       KY
000400 99034WAYMEL        WAYNE          KALAMAZOO     MI
```

Data - Partitioned Data Sets

As mentioned previously, the data may reside in various containers. For example, if your program required data from a member of a partitioned data set (PDS), it would be coded as shown here.

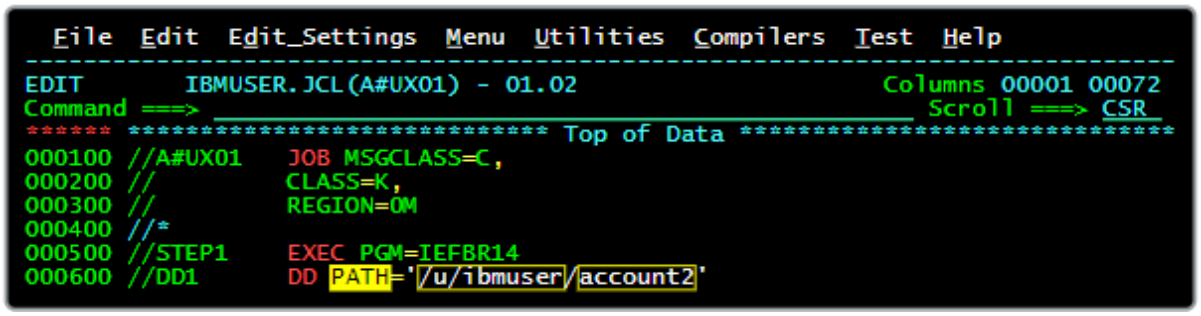
In this example, COBOL source code from PDS member INSPECT is going to be compiled.



Data - z/OS UNIX FIles

JCL DD statements can be used to specify z/OS UNIX data to be used by a program. In this example, the PATH parameter is used to identify the location and file name to be used.

Notice that these statements use lower case text, so if coding something like this make sure the ISPF profile CAPS attribute is turned off, otherwise it will be converted to uppercase when you press the Enter key. Note that apart from this z/OS UNIX reference, all JCL should be coded in uppercase.



- The PATH parameter is only required when referencing a z/OS UNIX file.
- This is the name of the directories within the z/OS UNIX file system, where the z/OS UNIX file resides.
- This is the name of the file to be accessed; because the path name contains special characters (/), as well as lower case characters it is enclosed in single quotes (‘).

DISP Parameter

The DISP parameter describes the disposition, or status, of the data set. It normally consists of three sub parameters that describe the following:

- The status of the data, for example, whether it exists and whether it is required exclusively.
- How the data set is to be handled if the job step completes successfully.
- How the data set is to be handled should the step fail.

If you are dealing with an existing data set being used for input purposes, then it makes sense that regardless of the success of the program using it, that in most cases you will want the data set kept when the program completes. In this scenario, you will often see just the first subparameter used, leaving the other sub parameters to default to a status of KEEP.

```
 File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
--------------------------------------------------------------------------------
EDIT       IBMUSER.JCL(AGENER) - 01.09                 Columns 00001 00072
Command ===> _____________________________________________ Scroll ===> CSR
****** ***************************** Top of Data ******************************
000100 //AGENER    JOB MSGCLASS=C,
000110 //          CLASS=K
000120 //*
000200 //*  STEP 1 - PEN TRANSACTION BACKUP
000210 //*
000300 //STEP1     EXEC PGM=IEBGENER
000400 //SYSUT1    DD DSN=IBMUSER.PEN.TRANS.DAILY,DISP=SHR
000500 //*
```

DISP Sub Parameters

What you will often see when referencing an existing data set for input, is a status of SHR (share), or OLD (exclusive use).

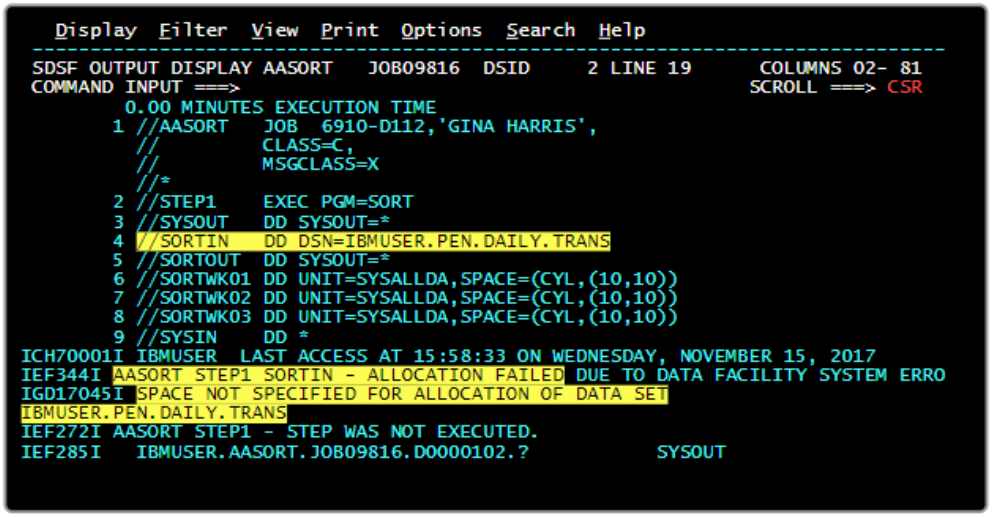
- As the names suggest, SHR allows other users or programs to access the data at the same time, as long as they do not require it exclusively. It is effectively a read-only request of that data.
- A DISP of OLD indicates that the use of the data is required exclusively by this step, making it unavailable to other programs even if they require only read access.

```
DD DSN=IBMUSER.JSON.RECS.MSTR,DISP=OLD
DD SYSOUT=*

DD DSN=IBMUSER.COPY.STREAM2,DISP=SHR
```

Forgetting to use DISP

It can be easy to forget to code one of the DD statement parameters, so what will happen if you forget to code a DISP parameter?

In this scenario you want to provide some input to a sort program through the SORTIN DD statement and although the data set name is coded, there is no DISP parameter. The default when no DISP parameter is coded is (NEW,DELETE,DELETE). In this case, the job's output messages indicate that when it attempted to create the data set, which is not what you want anyway, there was no space specified for the data set, creating an error.

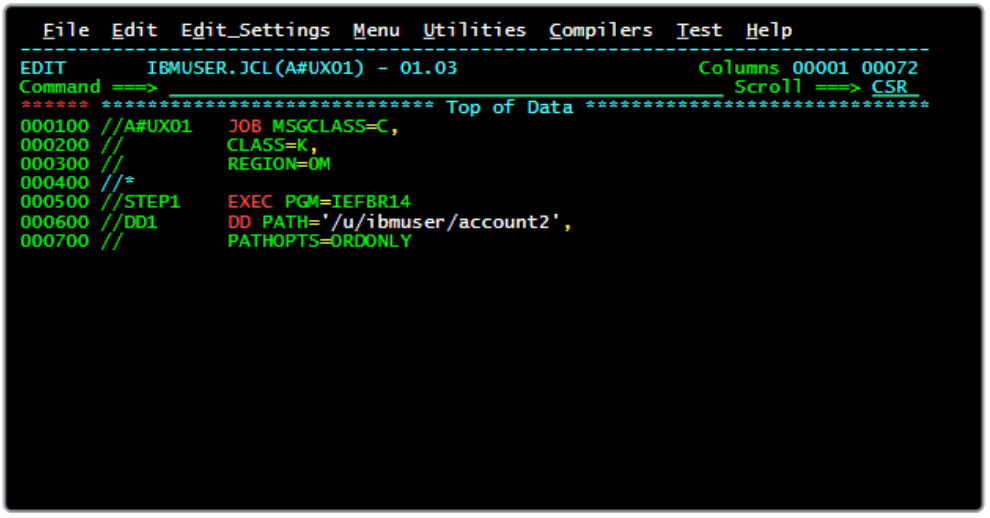


PATHOPTS Parameter

When referencing an existing z/OS UNIX file, a PATHOPTS parameter can be used to specify the access type and status of the file, so it loosely correlates to the first DISP subparameter discussed earlier.

For a file being used for input purposes only, a value of ORDONLY - open the file for reading only can be used.

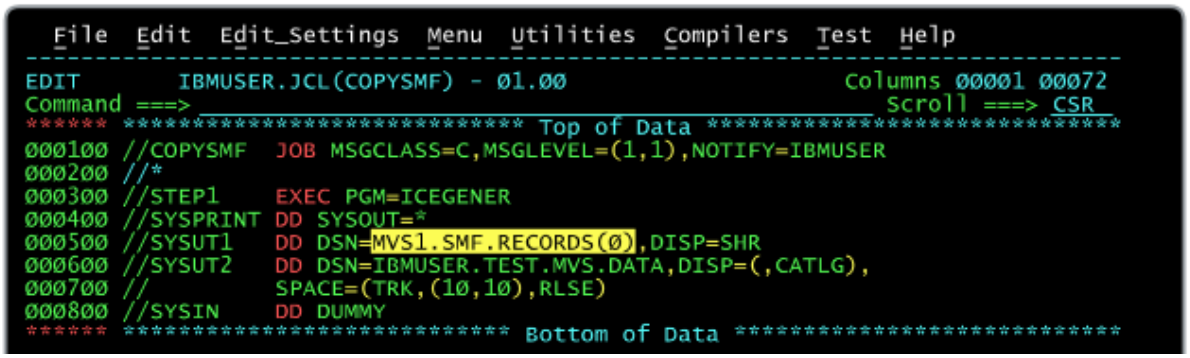
- If this PATHOPTS parameter is not specified, then the system assumes that it is an existing file and will search for it. If not located, the step will fail.



Referencing Existing Data

If you have been working with JCL, you are likely to have seen a data reference similar to the one shown here. These are called generation data sets.

Suppose you have the same job that needs to run every day, and it creates a sequential data set that contains daily transactions. This data set also needs to be kept for a minimum of 20 working days. You would not normally be able to run the exact same JCL everyday day because it would be trying to create a data set of that name when one already exists



It is now the following day and the same job is run using the same JCL but referencing different daily transactions. It has attempted to create a data set called PEN.XTRCTED.DAILY.TRANS but because the previous day's job already did this, the job has failed indicating a data set of that name already exists.

One way around this is to define a data set name that accepts different versions, or generations, under that name. This is possible by using generation data sets. As mentioned, this topic is discussed in detail in a later course. For the time being.

Temporary Data Sets

The types of data sets discussed so far are permanent data sets and files. They remain on the system until they are removed, either manually - by you or an operator, or automatically, for example, if they have met an expiry date.

When creating your job, you may find that you only need a temporary data set. This type of data set is useful when steps require data to be sorted or manipulated, possibly before being printed, or you need to copy some production data to be used for testing, and after your job completes the data is deleted.

- When temporary data is created it will contain && followed by a one to eight character name of your choice.
- When this data is referenced as input in a later step the same disposition indicates it is existing data.

```
  File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
------------------------------------------------------------------------------
EDIT       IBMUSER.JCL(DFPSSIMP) - 01.06                Columns 00001 00072
Command ===> ____________________________________________ Scroll ===> CSR
000900 //STEP01   EXEC PGM=IEBGENER
001000 //SYSPRINT DD SYSOUT=*
001100 //SYSUT3   DD UNIT=SYSDA,SPACE=(80,(261,261))
001200 //SYSUT4   DD UNIT=SYSDA,SPACE=(256,(220,220))
001300 //SYSUT1   DD DSN=SYS1.IGDVBS1(DFPSSCDS),DISP=SHR
001400 //SYSUT2   DD DSN=&&XASCDS,DISP=(NEW,PASS),
001500 //            UNIT=SYSDA,SPACE=(CYL,(1,1)),
001600 //            DCB=(BLKSIZE=4104,LRECL=4100,RECFM=VBS)
001700 //SYSIN    DD DUMMY
001800 //*************************************************/
001900 //*  THIS STEP WILL IMPORT THE STARTER SET SCDS     */
002000 //*  REPLACE XXXXXX TO YOUR TARGET  VOLUME SERIAL   */
002100 //*************************************************/
002200 //STEP02   EXEC PGM=IDCAMS,COND=(4,LT,STEP01)
002300 //SCDSIN   DD DSN=&&XASCDS,DISP=OLD
002400 //SYSPRINT DD SYSOUT=Z
002500 //SYSIN    DD *
002600   IMPORT -
002700       OUTDATASET(SCDS.PRIMARY.LINEAR) -
```

Referbacks

If your job uses a data set across several steps then you might see the DSN parameter used as coded here. This is used to referback and use the same data set that was referenced earlier. This means that if the name of the data set needs to be changed for any reason, you only need to update this name where it is first referenced.

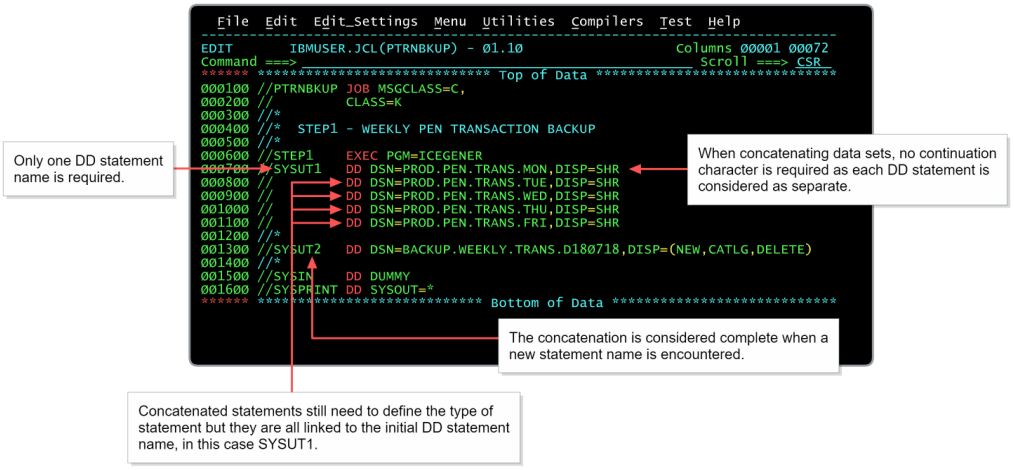
- The asterisk indicates a referback reference
- The second part is the name of the step to which the reference is being made
- The final section is the DD statement name within the specified step, in this case STEP3 that is to be referenced.
    - Data set IBMUSER.PEN.TRANS.PARMPEN will be used by the INDD DD statement in STEP4.

```
  File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
------------------------------------------------------------------------------
EDIT       IBMUSER.JCL(APENCHK) - 01.02                 Columns 00001 00072
Command ===> ____________________________________________ Scroll ===> CSR
001700 //*
001800 //STEP3    EXEC PGM=ICEGENER
001900 //SYSUT1   DD DSN=IBMUSER.SALESIN,DISP=SHR
002000 //SYSUT2   DD DSN=IBMUSER.PEN.TRANS.PARMPEN,
002100 //          DCB=(RECFM=FB,LRECL=80,BLKSIZE=32720),
002200 //          SPACE=(CYL,(1,1),RLSE),UNIT=SYSDA,DISP=(,CATLG)
002300 //SYSPRINT DD SYSOUT=*
002400 //SYSIN    DD DUMMY
002500 //*
002600 //STEP4    EXEC PGM=REPPEN
002700 //INDD     DD DSN=*.STEP3.SYSUT2,DISP=SHR
002800 //OUTDD    DD SYSOUT=*
002900 //SYSPRINT DD SYSOUT=*
003000 //SYSIN    DD DUMMY
****** ***************************** Bottom of Data ******************************
```

Concatenating Data Sets

There will be times when you need to include more than one set of data as part of the input to a program. For example, a management report run every Friday requires a data set that is created on a daily basis throughout the week.

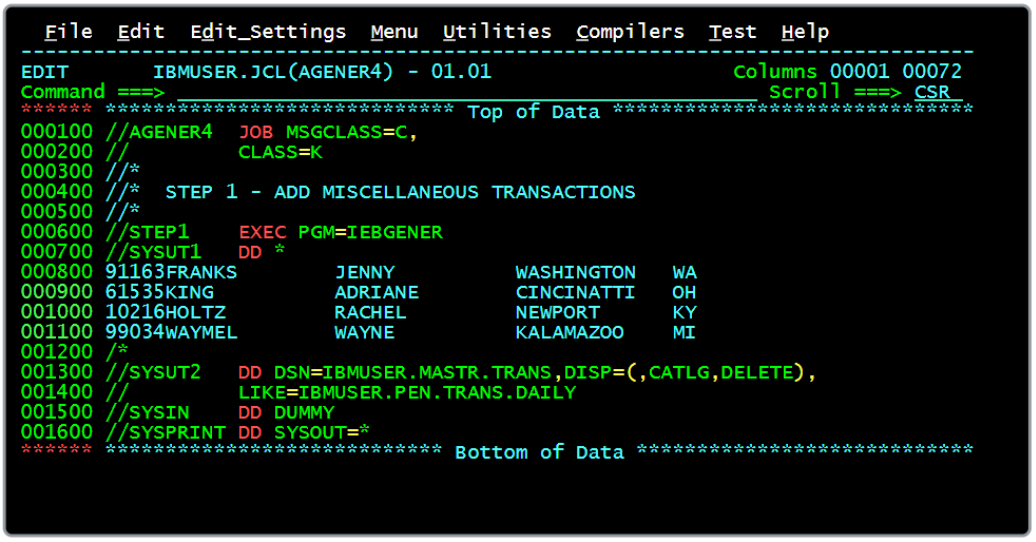
Joining these data sets together so that they form one set of data is called concatenation.



In-Stream Data

Finally, data that your program references does not need to be stored in a data set or file. It can be supplied as in-stream data. This is raw data that appears within your job.

You can see with the SYSUT1 DD statement that what immediately follows this is an asterisk (*). This signifies that what follows is in-stream, or raw, data. You will also notice that the lines that follow do not contain // characters. In-stream data is considered complete when a delimiter (/*) character, or a JCL statement (//), is encountered, or there is no other data following the last line of in-stream data.



In-Stream Data Parameter

What happens if the in-stream data you need to pass contains the // characters in columns one and two? This might be required to copy JCL to a PDS member that will be used for later processing.

Well, normally as soon as the // characters are encountered the system assumes it is the end of your in-stream data. To resolve this, the DD DATA statement shown here can be used.

```
 File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
--------------------------------------------------------------------------------
EDIT       IBMUSER.JCL(COPCOLL) - 01.01                Columns 00001 00072
Command ===>                                             Scroll ===> CSR
****** ***************************** Top of Data ********************************
000100 //COPCOLL  JOB MSGCLASS=C,CLASS=K
000200 //*
000300 //STEP1    EXEC PGM=IEBGENER
000400 //SYSUT1   DD DATA,DLM='$$'
000500 //OLCOLLSX JOB CLASS=K,NOTIFY=GREG
000600 //*
000700 //STEP1    EXEC PGM=SORT
000800 //SORTIN   DD DSN=IBMUSER.PEN.DAILY.TRANS,DISP=OLD
000900 //SORTOUT  DD SYSOUT=*
001000 //SORTWK01 DD UNIT=SYSALLDA,SPACE=(CYL,(10,10))
001100 //SORTWK02 DD UNIT=SYSALLDA,SPACE=(CYL,(10,10))
001200 //SYSOUT   DD SYSOUT=*
001300 //SYSIN    DD *
001400  SORT FIELDS=(20,10,CH,A)
001500 /*
001600 $$
001700 //SYSUT2   DD DSN=IBMUSER.SAMP.DATA(OLCOLL),
001800 //            DISP=(NEW,CATLG),
```

Normally the /* characters can be used as a delimiter, to terminate the in-stream data, but if these characters need to be used in your in-stream data, as per the example here, then this needs to be changed. The DLM parameter, highlighted here, can be used to define two characters that are to be used as the delimiter.

Accidental In-Stream Data

In earlier sections it mentioned how the // characters needed to be coded for the system to determine that what you were passing was a JCL statement. While the previous page discussed how to pass in-stream data that does not require these characters, have a look at the output from the job shown here, which is one type of error you might receive when you forget to code // for a required JCL statement.

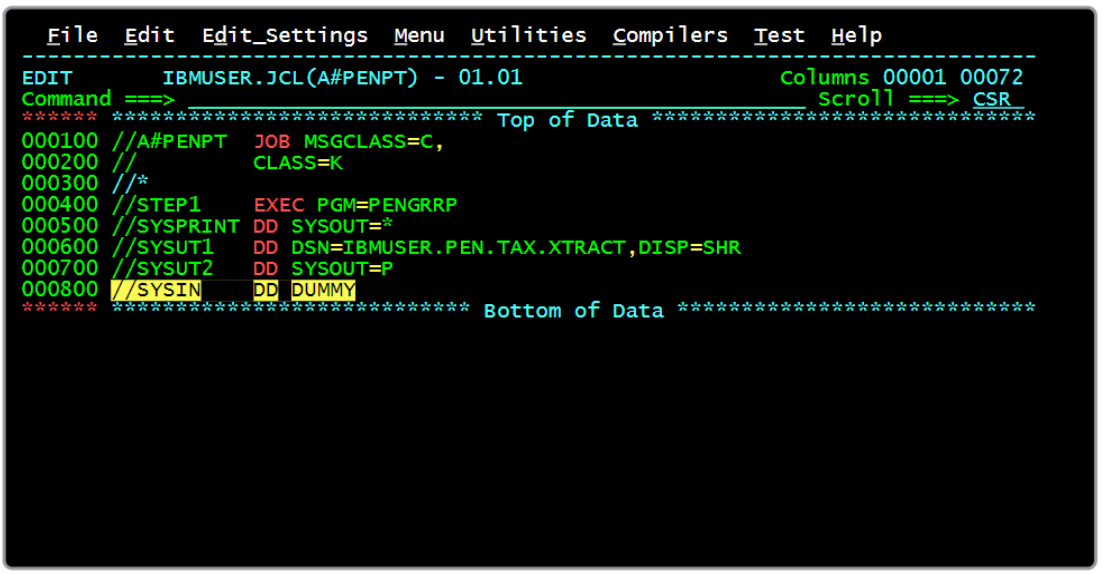
In this example, the system generates a SYSIN DD statement, assuming that what you want to do is to pass this raw data to the program. Depending on the program, this could result in several different types of problems or errors, usually identified in your job output.

Dummying Data

There will be occasions when you require an input data set to be dummied so that no data is passed to the program. An example of this might be if you are involved with testing JCL but do not need to create any of the data associated with a program's processing.

Another example shown here involves the IEBGENER program. With this program the SYSIN DD statement is normally associated with supplying in-stream control statements, which this utility might use for specialized copying. You will see that in most cases these control statements are not required but because the DD statement is required by the program, it needs to appear but be ignored.

This is achieved by using a DSN=NULLFILE or by specifying DUMMY as shown below.



Output Data Set Naming and Placement

Output Data Set Naming and Disposition Requirements

Sending Output to an Existing Data Set

There are two options available when producing data from your program. You can either create a new data set or z/OS UNIX file and send the data to it, or you can add data to an existing data set or file.

Overwriting a Data Set:

```
  File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
--------------------------------------------------------------------------------
EDIT       IBMUSER.JCL(J#ETSCR1) - 01.01                Columns 00001 00072
Command ===> ____________________________________________ Scroll ===> CSR
****** ***************************** Top of Data ******************************
000100 //J#ETSCR1 JOB MSGCLASS=C,
000200 //          CLASS=K
000300 //*
000400 //STEP1     EXEC PGM=IEBGENER
000500 //SYSPRINT DD SYSOUT=*
000600 //SYSUT1   DD DSN=IBMUSER.PEN.DAILY.TRANS(+0),DISP=SHR
000700 //SYSUT2   DD DSN=IBMUSER.TEST.PEN.TRANS,DISP=OLD
000800 //SYSIN    DD DUMMY
****** **************************** Bottom of Data ****************************
```

- When a DISP=OLD is specified for an output data set, the data set is held for exclusive use, as you are going to be writing to it.
- What then occurs is that any output data will overwrite or replace the data set's content entirely

```
  File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
--------------------------------------------------------------------------------
EDIT       IBMUSER.JCL(J#ETSCR1) - 01.01                Columns 00001 00072
Command ===> ____________________________________________ Scroll ===> CSR
****** ***************************** Top of Data ******************************
000100 //J#ETSCR1 JOB MSGCLASS=C,
000200 //          CLASS=K
000300 //*
000400 //STEP1     EXEC PGM=IEBGENER
000500 //SYSPRINT DD SYSOUT=*
000600 //SYSUT1   DD DSN=IBMUSER.PEN.DAILY.TRANS(+0),DISP=SHR
000700 //SYSUT2   DD DSN=IBMUSER.TEST.PEN.TRANS,DISP=MOD
000800 //SYSIN    DD DUMMY
****** **************************** Bottom of Data ****************************
```

- Another method of sending data to an existing data set is by using the MOD value for DISP parameter. This will append the information to the end of the existing data, rather than overwriting it.
- This is useful when you might need to continuously accumulate data over several days.

Creating a new Data Set:

The other option is to create a new data set, or z/OS UNIX file, to store your data. In the case of mainframe data sets, the Data Source Name (DSN) parameter is used to specify its name.

```
  File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
--------------------------------------------------------------------------------
EDIT       IBMUSER.JCL(AASORT3) - 01.02                Columns 00001 00072
Command ===> _____________________________________________ Scroll ===> CSR
****** ***************************** Top of Data ******************************
000100 //UBSRTEX  JOB  6910-D112,'GINA HARRIS',
000200 //          CLASS=C,
000300 //          MSGCLASS=X
000400 //*
000500 //STEP1     EXEC PGM=SORT
000600 //SORTIN    DD DSN=PROD.PEN.TRANS(0),DISP=OLD
000700
000800 //SORTOUT   DD DSN=PROD.PEN.TRANS.SORTED,DISP=(NEW,CATLG),
000900 //          UNIT=SYSDA,SPACE=(TRK,(1,1),RLSE)
001000 //SORTWK01 DD UNIT=SYSALLDA,SPACE=(CYL,(10,10))
001100 //SORTWK02 DD UNIT=SYSALLDA,SPACE=(CYL,(10,10))
001200 //SORTWK03 DD UNIT=SYSALLDA,SPACE=(CYL,(10,10))
001300 //SYSOUT    DD SYSOUT=*
001400 //SYSIN     DD *
001500  SORT FIELDS=(20,10,CH,A)
****** ***************************** Bottom of Data ***************************
```

There are several rules in relation to the name you supply:

- A maximum of 44 characters can be specified
    - This includes periods (.)
- For generation data sets, the maximum is 35 characters
- You must be authorized to use the name you specify

Data Set Naming Conventions

The data set name you create can be qualified or unqualified.
An unqualified name only consists of one to eight alpha-numeric or national characters. Like some of the previous standards you looked at, the first character in this name must be alphabetic or national.

```
//SYSUT1   DD DSN=GHMAST.FAMAPS.D256,DISP=OLD
//SYSUT2   DD DSN=FAMDATA,DISP=(,CATLG),
//         UNIT=SYSDA,SPACE=(CYL,(10,10),RLSE)
```

```
//SORTIN   DD DSN=PROD.PEN.TRANS(0),DISP=OLD
//SORTOUT  DD DSN=PROD.PEN.TRANS.SORTED,DISP=(NEW,CATLG),
//         UNIT=SYSDA,SPACE=(TRK,(1,1),RLSE)
```

```
//INDD     DD DSN=IBMUSER.REPORT,DISP=OLD
//OUTDD    DD DSN=IBMUSER.PENWEEKLY.TRANS,
//         DCB=(RECFM=FB,LRECL=80,BLKSIZE=32720),
//         SPACE=(CYL,(1,1),RLSE),UNIT=SYSDA,DISP=(,CATLG)
```

You are more likely to use qualified names for your data sets. These are multiple unqualified names that are joined by periods. In the second example, PROD is referred to as the high-level qualifier (HLQ).

Creating a New Generation Data Set

Previously we saw that you can reference a generation data set by specifying its relative generation number. For example, 0 or +0 would call the latest, or current, generation of that data set.

```
 File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
--------------------------------------------------------------------------------
EDIT       IBMUSER.JCL(MASTCOP) - 01.10                Columns 00001 00072
Command ===> _____________________________________________ Scroll ===> CSR
****** ***************************** Top of Data ******************************
000100 //MASTCOP  JOB MSGCLASS=C,MSGLEVEL=(1,1),NOTIFY=IBMUSER
000200 //*
000300 //STEP1    EXEC PGM=ICEGENER
000400 //SYSPRINT DD SYSOUT=*
000500 //SYSUT1   DD DSN=IBMUSER.CCF.REPORT,DISP=OLD
000600 //SYSUT2   DD DSN=IBMUSER.PEN.DAILY.TRANS(+1),
000700 //         DISP=(,CATLG),
000800 //         SPACE=(TRK,(5,5),RLSE),
000900 //         UNIT=SYSDA
001000 //SYSIN    DD DUMMY
****** **************************** Bottom of Data ****************************
```

In the example shown here, +1 is specified to create a new generation of a data set. However, the use of generations can get more complex than this.

Creating a New Temporary Data Set

Creating a temporary data set is a much simpler process. Similar to what was mentioned previously we will be using the two ampersand (&&) characters.

- What follows this is a one to eight-character name that must begin with an alphabetical or national character.
- The remaining characters can be alphabetical, national, or numeric.

```
//STEP1    EXEC PGM=IEBGENER
//SYSUT1   DD DSN=IBMUSER.JET.DAILY.PAYMENTS,DISP=OLD
//SYSUT2   DD DSN=&&JETSRT,DISP=(,PASS),
//         SPACE=(CYL,(1,1),RLSE),UNIT=SYSDA
```

Another method used to create a temporary data set is to not code a DSN parameter at all. In this scenario, the data set will be named by the system. This is useful if you do not need to reference the temporary data set in later steps of your job.

Dummy Data Set

There will be situations where you are testing a program, or recreating part of a job, and you do not need to create a new data set. You may think that you can remove the DD statement that would normally reference this data set, but remember, the program is coded to look for certain DD statements, and if not found the program may fail.

In this scenario, you will need to define the DD statement so that any data set reference is ignored. This is achieved using a DSN=NULLFILE or just specifying DUMMY for the DD statement.

Normally in STEP2 (image below) some sorted records are used as input to produce three types of different output, all printed on various forms. However, in the case where some forms need to be recreated, the step could be rerun and only the required forms need to be specified (UB101).

The other two DD statements that would normally create output are not required so either DUMMY or DSN=NULLFILE can be specified.

```
  File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
--------------------------------------------------------------------------------
EDIT       IBMUSER.JCL(UBSRTPT) - 01.01                Columns 00001 00072
Command ===> ____________________________________________ Scroll ===> CSR
001600 //*
001700 //* THIS STEP TAKES SORTED RECORDS AND PRODUCES VARIOUS FORMS
001800 //*
001900 //STEP2    EXEC PGM=USBPTF
002000 //SORTIN   DD DSN=XAPROD.USB.RECS.SORTED,DISP=OLD
002100 //UB101    DD SYSOUT=(F,,F101)
002200 //UB230    DD DUMMY
002300 //UB770    DD DSN=NULLFILE
002400 //SYSPRINT DD SYSOUT=*
****** ***************************** Bottom of Data ****************************
```

If you are involved in testing JCL, you may see the DUMMY parameter used as shown here. In this example, the REPDATA DD statement would normally contain the SYSOUT parameter, but as this is not required during testing, it is dummied.

The data that appears after DUMMY is not used. Once testing is completed, the DUMMY can be removed, leaving the original SYSOUT requirement.

```
  File  Edit  Edit Settings  Menu  Utilities  Compilers  Test  Help
--------------------------------------------------------------------------------
EDIT       IBMUSER.JCL(TAXPRINT) - 01.02               Columns 00001 00072
Command ===> ____________________________________________ Scroll ===> CSR
002800 //*
002900 //PRINTIT  EXEC PGM=TAXPT
003000 //SYSPRINT DD SYSOUT=*
003100 //INDATA   DD DSN=TAXDEPT.EOM.PAYRECS,DISP=SHR
003200 //REPDATA  DD DUMMY,SYSOUT=(P,,TX209)
003300 //SYSOUT   DD SYSOUT=*
****** ***************************** Bottom of Data ****************************
```