

# Table of Contents:

<b>Table of Contents:</b>	<b>1</b>
<b>Introduction to Batch Processing</b>	<b>2</b>
Enterprise Data Processing	2
Mainframe Data Processing	2
Data Types:	2
Online and Batch Processing	3
Benefits of Batch Processing	3
Working with Batch Processing	4
Examples:	4
Running a Batch Job	4
Other terminology:	4
Batch Job Automation	5
Batch Job Scheduling	5
Batch Processing Prerequisites	6
<b>Coding Requirements</b>	<b>7</b>
JCL Coding Rules	7
Planning Processing Requirements	7
Reality of JCL Coding	7
How to copy JCL:	7
Identifying JCL code	8
The empty // curse	8
Uppercase Characters	9
Record Length	9
Continuing a statement	10
Statement Breakdown	11
Statement Names	11
Statement Types	11
Parameters	12
Positional Parameters	12
Example:	12
Keyword Parameters	13
Example:	13
Comments	13

# 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.

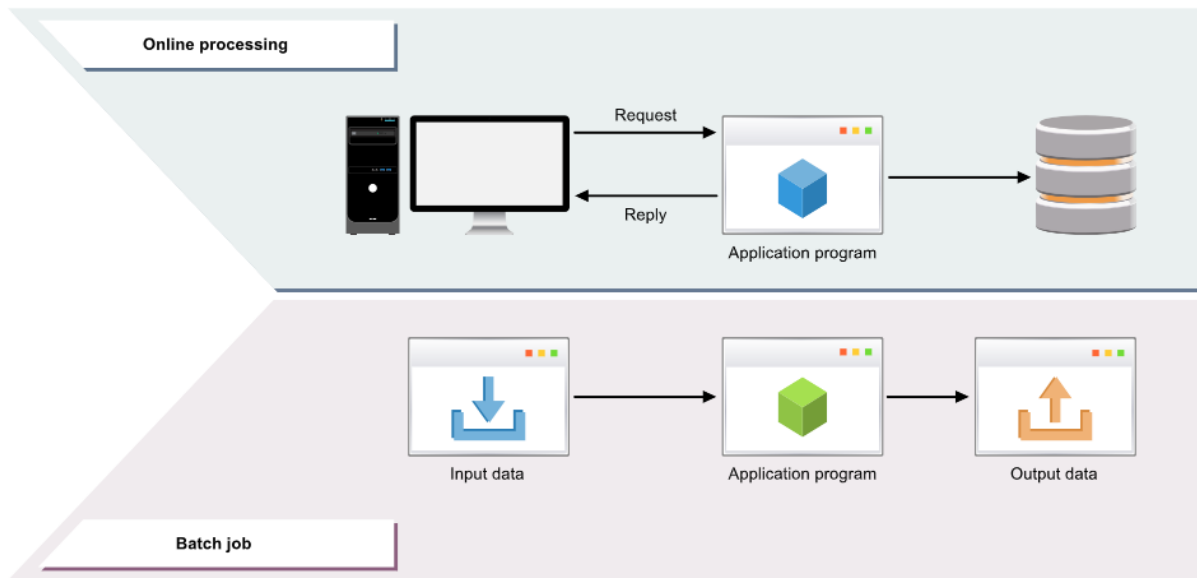
#### **Data Types:**

- 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.

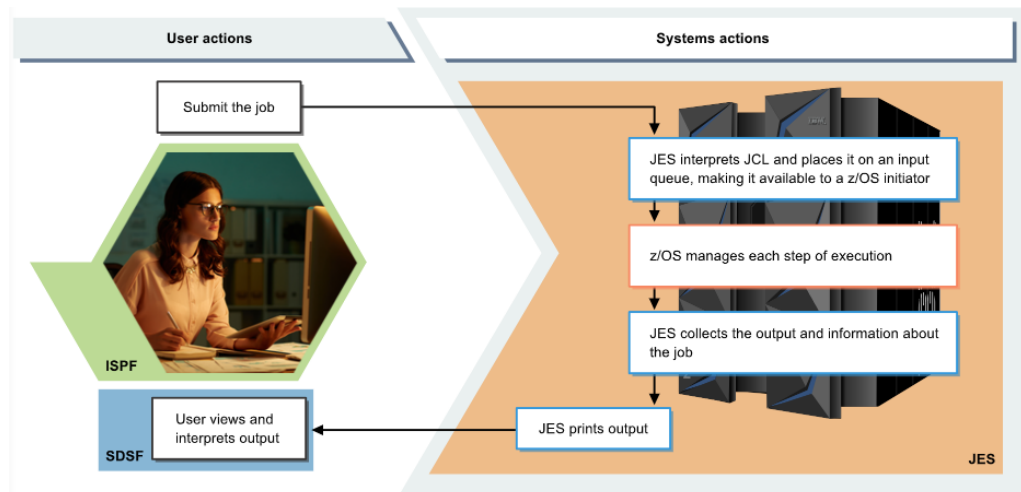
### Examples:

- 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**

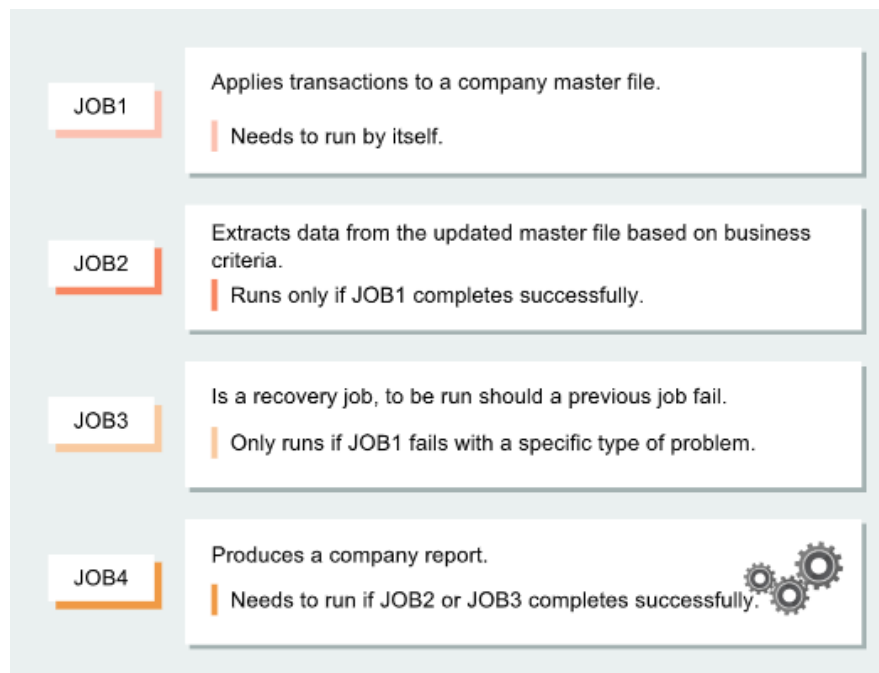
Many larger organizations have implemented automated batch scheduling software that is capable of handling complex batch job scheduling requirements. For example, batch processing may need to run at a specific time, once other data has become available, when a resource becomes free, or simply triggered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

Job used to compile COBOL source code.

Job used to unload Db2 database records.

```
Menu  Functions  Confirm  Utilities  Help
-----
EDIT  IBMUSER.JCL                                     Row 0000035 of 0000066
Command ==>

Name      Prompt      Size  Created      Changed      ID
-----
ARUNREP   ARUNREP   3     2017/09/04   2017/09/07   20:11:09   IBMUSER
ASM1GH    ASM1GH    9     2017/09/04   2017/09/04   20:05:24   IBMUSER
COBCOMP   COBCOMP   38    2016/06/14   2017/09/06   20:34:02   IBMUSER
CSFSMFJ   CSFSMFJ   12    2017/09/03   2017/09/03   23:57:06   IBMUSER
DBRM      DBRM      10    2017/09/19   2017/09/19   20:34:22   IBMUSER
DBUNLOAD  DBUNLOAD  10    2017/09/19   2017/09/19   20:34:22   IBMUSER

Data Set Information
Command ==>
Data Set Name . . . . : IBMUSER.JCL
General Data
Data class . . . . . : **None**
Organization . . . . : PO
Record format . . . . : FB
Record length . . . . : 80
Block size . . . . . : 32720
Current utilization
Used pages . . . . . : 123
% Utilized . . . . . : 68
```

## 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.

```
EDIT      IBMUSER.JCL.LIB(JIEBGNR2) - 01.04          Columns 00001 00072
Command ==>                                         Scroll ==> CSR
***** ***** Top of Data *****
000100 //JIEBGNR2 JOB 'TEST RUN',MSGCLASS=X,CLASS=A,NOTIFY=&SYSUID
000110 //MYLIB   JCLLIB ORDER=IBMUSER.PROCLIB
000111 //STEP1    EXEC  PROCGEN
000128 //PSTEP1.SYSUT1 DD *
000129 THIS WAS LINE 1
000130 /*
000140 //          DD NULLOVRD
000150 // DD *
000160 THIS WAS LINE 3
000170 /*
***** ***** Bottom of Data *****
```



## 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.

```
File Edit Edit_Settings Menu Utilities Compilers Test Help
-----
EDIT      IBMUSER.JCL(COPYSMF) - 01.05          Columns 00001 00072
Command ==> SUBMIT                               Scroll ==> CSR
=COLS> -----1-----2-----3-----4-----5-----6-----7--
***** Top of Data *****
000100 COPYSMF JOB MSGCLASS=C,MSGLEVEL=(1,1),NOTIFY=IBMUSER
000200 //*
000300 //STEP1 EXEC PGM=ICEGENER
000400 //SYSPRINT DD SYSOUT=*
000500 //SYSUT1 DD DSN=MVS1.SMF.RECORDS(0),DISP=SHR
000600 //SYSUT2 DD DSN=IBMUSER.TEST.MVS.DATA,DISP=(,CATLG),
000700 //          SPACE=(TRK,(10,10),RLSE)
000800 //SYSIN DD DUMMY
***** Bottom of Data *****

IKJ56700A ENTER JOBNAME CHARACTER(S) -
```

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.

```
IKJ56700A ENTER JOBNAME CHARACTER(S) -
COPYSMF
IKJ56254I JOBNAME TRUNCATED+
IKJ56250I JOB IBMUSERC(JOB09296) SUBMITTED
***
```

What the initial prompt did 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.

```
000300 //
000400 // SET MEM=COMPUTE2
000500 //
000600 //COMP1 EXEC PGM=IGYCRCTL,REGION=0M,
000700 // PARM='LIST,XREF'
000800 //* SCHEDULE STARTBY='+00:05'
```



## 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.

```
File Edit Edit_Settings Menu Utilities Compilers Test Help
-----
EDIT      IBMUSER.JCL(CSFSMFJ) - 01.00                      Columns 00001 00072
Command ==> PROF                                           Scroll ==> CSR
***** ***** Top of Data *****
=PROF> ....JCL (FIXED - 80)....RECOVERY ON....NUMBER OFF.....
=PROF> ....CAPS OFF....HEX OFF....NULLS ON STD....TABS OFF.....
=PROF> ....AUTOSAVE ON....AUTONUM OFF....AUTOLIST OFF....STATS ON.....
=PROF> ....PROFILE UNLOCK....IMACRO NONE....PACK OFF....NOTE ON.....
=PROF> ....HILITE OFF CURSOR FIND.....
000001 //CSFSMFJ JOB (), 'GMH RUN', CLASS=A, MSGCLASS=X, NOTIFY=gh8ibm
000002 //*****
000003 //*-----*
000004 //* UNLOAD SMF RECORDS TO PRINT *
000005 //*-----*
000006 //SMFDMP EXEC PGM=IFASMFDP
000007 //DUMPIN DD DISP=SHR, DSN=SYS1.S0W1.MAN1.DATA
000008 //DUMPOUT DD SYSOUT=*
000009 //SYSPRINT DD SYSOUT=*
000010 //SYSIN DD *
000011 INDD(DUMPIN, OPTIONS(DUMP))
000012 OUTDD(DUMPOUT, TYPE(82))
***** ***** Bottom of Data *****
```

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.

```
File Edit Edit_Settings Menu Utilities Compilers Test Help
-----
EDIT      IBMUSER.JCL(A#UX01) - 01.01                      Columns 00001 00072
Command ==>                                           Scroll ==> CSR
***** ***** Top of Data *****
000001 //A#UX01 JOB MSGCLASS=C, MSGLEVEL=(1,1), NOTIFY=IBMUSER, REGION=0M
000002 //STEP1 EXEC PGM=IEFBR14
000003 //DD1 DD PATH='/u/ibmuser/account2',
000004 // FILEDATA=BINARY,
000005 // PATHMODE=(SIRUSR, SIWUSR, SIRGRP, SIROTH),
000006 // PATHDISP=(KEEP, DELETE),
000007 // PATHOPTS=(OCREAT, ORDWR)
```

## 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.

```
File Edit Edit_Settings Menu Utilities Compilers Test Help
EDIT IBMUSER.JCL(APDSE2) - 01.01 Columns 00001 00072
Command ==> Scroll ==> CSR
=COLS> -----1-----2-----3-----4-----5-----6-----7--
***** Top of Data *****
000100 //APDSE2 JOB MSGCLASS=X,CLASS=C
000200 //*
000300 //CREATE EXEC PGM=IEFBR14
000400 //PDSE2 DD DSN=IBMUSER.PDSE.GENS,
000500 // DSNTYPE=(LIBRARY,2),MAXGENS=10,
000600 // RECFM=FB,LRECL=80,
000700 // UNIT=SYSALLDA,SPACE=(CYL,(1,1,1)),
000800 // DISP=(,CATLG,DELETE)
***** Bottom of Data *****
```

Commas appear at the end of each line indicating to the system that the line that follows contains continuation information for that statement.

No comma is required here, signifying the end of information for this statement.







In this example, the continuation appears in column 12.

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.

Incorrect statement names		
	1STSTEP	Begins with a number 
	MYPROGSTEP	Too many characters 
	&BACKUP	Ampersand (&) is not a national or alpha character 

### Statement Types

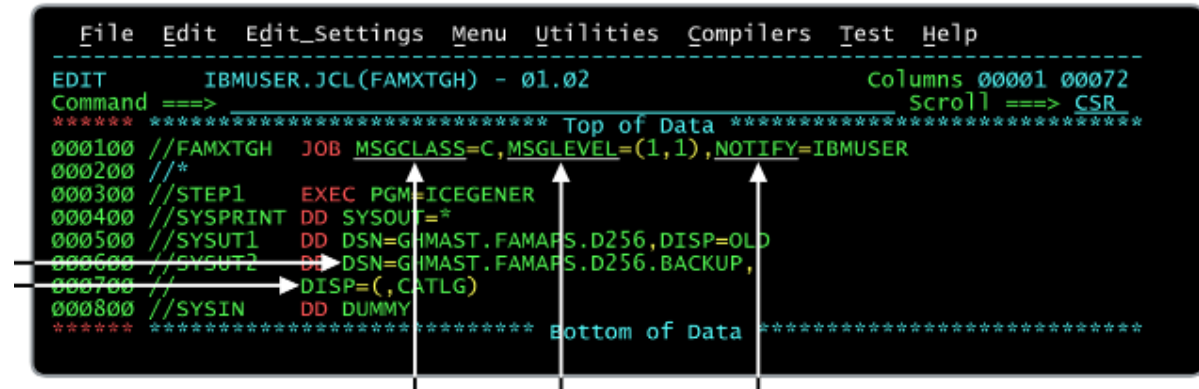
Associated with the name is the statement type. Common statements used include JOB, EXEC, and DD (as seen in the image below). At least one space must be coded following the name before this statement type is entered.

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.

```
File Edit Edit_Settings Menu Utilities Compilers Test Help
-----
EDIT          IBMUSER.JCL(FAMXTGH) - 01.00          Columns 00001 00072
Command ==>                                         Scroll ==> CSR
***** Top of Data *****
000100 //FAMXTGH JOB MSGCLASS=C,MSGLEVEL=(1,1),NOTIFY=IBMUSER
000200 //*
000300 //STEP1 EXEC PGM=ICEGENER
000400 //SYSPRINT DD SYSOUT=*
000500 //SYSUT1 DD DSN=GHMAST.FAMAPS.D256,DISP=OLD
000600 //SYSUT2 DD DSN=GHMAST.FAMAPS.D256.BACKUP,
000700 //          DISP=(,CATLG)
000800 //SYSIN DD DUMMY
***** Bottom of Data *****
```

## 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.

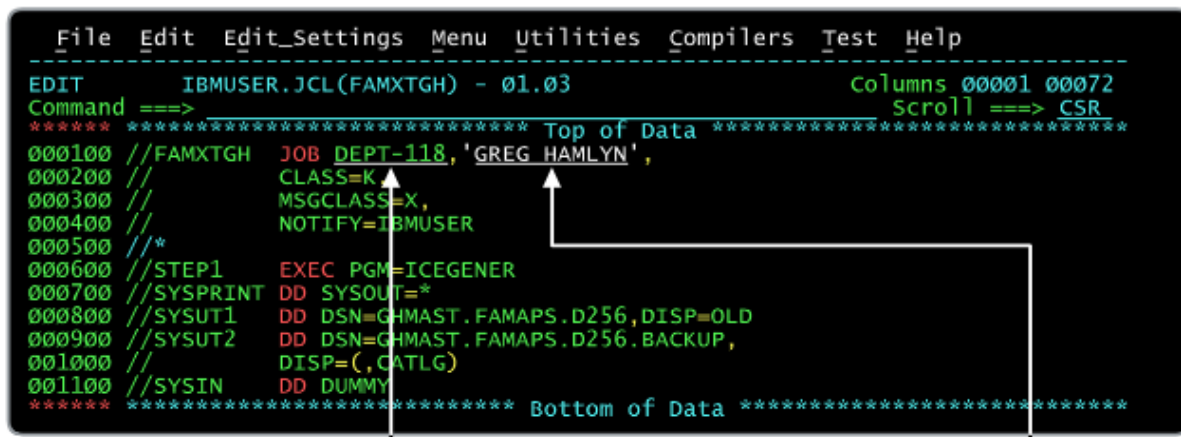


```
File Edit Edit_Settings Menu Utilities Compilers Test Help
-----
EDIT      IBMUSER.JCL(FAMXTGH) - 01.02                      Columns 00001 00072
Command ==>                                         Scroll ==> CSR
*****
***** Top of Data *****
000100 //FAMXTGH  JOB  MSGCLASS=C,MSGLEVEL=(1,1),NOTIFY=IBMUSER
000200 //*
000300 //STEP1   EXEC PGM=ICEGENER
000400 //SYSPRINT DD SYSOUT=*
000500 //SYSUT1    DD DSN=GHMAST.FAMAPS.D256,DISP=OLD
000600 //SYSUT2    DD DSN=GHMAST.FAMAPS.D256.BACKUP,
000700 //          DISP=(,CATLG)
000800 //SYSIN     DD DUMMY
*****
***** Bottom of Data *****
```

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.



```
File Edit Edit_Settings Menu Utilities Compilers Test Help
-----
EDIT      IBMUSER.JCL(FAMXTGH) - 01.03                      Columns 00001 00072
Command ==>                                         Scroll ==> CSR
*****
***** Top of Data *****
000100 //FAMXTGH  JOB  DEPT-118, 'GREG HAMLYN',
000200 //          CLASS=K
000300 //          MSGCLASS=X,
000400 //          NOTIFY=IBMUSER
000500 //*
000600 //STEP1   EXEC PGM=ICEGENER
000700 //SYSPRINT DD SYSOUT=*
000800 //SYSUT1    DD DSN=GHMAST.FAMAPS.D256,DISP=OLD
000900 //SYSUT2    DD DSN=GHMAST.FAMAPS.D256.BACKUP,
001000 //          DISP=(,CATLG)
001100 //SYSIN     DD DUMMY
*****
***** Bottom of Data *****
```

### 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.

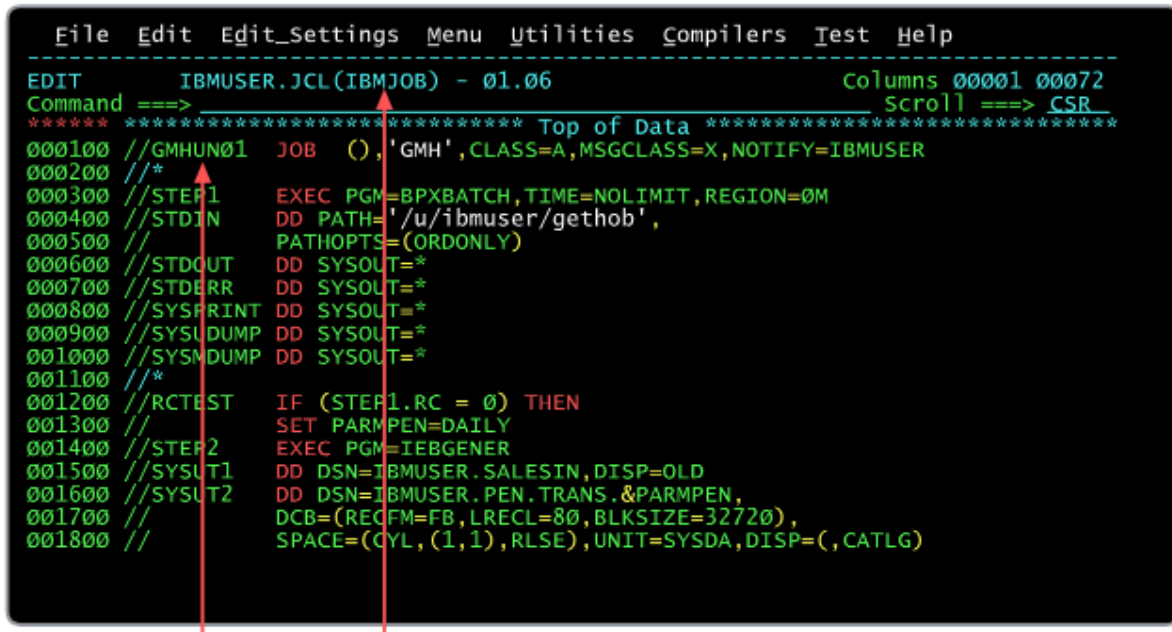
```
File Edit Edit_Settings Menu Utilities Compilers Test Help
-----
EDIT          IBMUSER.JCL(FAMXTGH) - 01.04          Columns 00001 00072
Command ==>                                         Scroll ==> CSR
***** ***** Top of Data *****
000100 //FAMXTGH  JOB 'GREG HAMLYN',
000200 //          CLASS=K,
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.



```
File Edit Edit_Settings Menu Utilities Compilers Test Help
-----
EDIT      IBMUSER.JCL(IBMJOB) - 01.06                      Columns 00001 00072
Command ==>                                         Scroll ==> CSR
***** Top of Data *****
000100 //GMHUN01 JOB (),'GMH',CLASS=A,MSGCLASS=X,NOTIFY=IBMUSER
000200 //*
000300 //STEP1 EXEC PGM=BPXBATCH,TIME=NOLIMIT,REGION=0M
000400 //STDIN DD PATH='/u/ibmuser/gethob',
000500 //      PATHOPTS=(ORDONLY)
000600 //STDOUT DD SYSOUT=*
000700 //STDERR DD SYSOUT=*
000800 //SYSPRINT DD SYSOUT=*
000900 //SYSUDUMP DD SYSOUT=*
001000 //SYSNDUMP DD SYSOUT=*
001100 //*
001200 //RCTEST IF (STEP1.RC = 0) THEN
001300 //      SET PARMEN=DAILY
001400 //STEP2 EXEC PGM=IEBGENER
001500 //SYSUT1 DD DSN=IBMUSER.SALESIN,DISP=OLD
001600 //SYSUT2 DD DSN=IBMUSER.PEN.TRANS.&PARMPEN,
001700 //      DCB=(RECFM=FB,LRECL=80,BLKSIZE=32720),
001800 //      SPACE=(CYL,(1,1),RLSE),UNIT=SYSDA,DISP=(,CATLG)
```

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.

```
File Edit Edit_Settings Menu Utilities Compilers Test Help
-----
EDIT      IBMUSER.JCL(IDCAMS6) - 01.04      Columns 00001 00072
Command ==> SUBMIT      Scroll ==> CSR
***** ***** Top of Data *****
000100 //IBMUSER  JOB MSGCLASS=X,CLASS=C
000200 //*
000300 //STEP1   EXEC PGM=IDCAMS
000400 //SYSPRINT DD SYSOUT=*
000500 //SYSIN    DD *
000600     DEFINE GENERATIONDATAGROUP -
000700         (NAME(IBMUSER.SMF.S0A1.DATA) -
000800         EMPTY -
000900         EXTENDED -
001000         LIMIT(200) )
001100 /*
***** ***** Bottom of Data *****

IKJ56700A ENTER JOBNAME CHARACTER(S) -
A
IKJ56250I JOB IBMUSERA(JOB09357) 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.

```
File Edit Edit_Settings Menu Utilities Compilers Test Help
-----
EDIT      IBMUSER.JCL(SUPJOB5) - 01.01      Columns 00001 00072
Command ==> SUBMIT      Scroll ==> CSR
***** ***** Top of Data *****
000100 //SUPAPD05 JOB MSGCLASS=X,CLASS=C
000200 //APPLY    EXEC PGM=TRANAPL
000300 //INPUT    DD DSN=PRODD.D112Y17.PENTRANS,DISP=OLD
000400 //OUTPUT   DD DSN=IBMUSER.TOTAL.PENTRANS,DISP=(,CATLG),
000500 //          UNIT=SYSDA,SPACE=(CYL,(10,10,RLSE))
000600 //SYSPRINT DD SYSOUT=*
000700 //SYSOUT   DD SYSOUT=*
000800 //SUPXTD10 JOB MSGCLASS=X,CLASS=C
000900 //EXTRACT  EXEC PGM=TRANXCT
001000 //SYSPRINT DD SYSOUT=*
001100 //INDD1    DD DSN=IBMUSER.TOTAL.PENTRANS,DISP=SHR
001200 //SYSIN    DD *
001300 RECORDS=10000
***** ***** Bottom of Data *****

IKJ56250I JOB SUPAPD05(JOB09358) SUBMITTED
IKJ56250I JOB SUPXTD10(JOB09359) SUBMITTED
***
```

## 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.JSODATA, 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 IRRO10I USERID IBMUSER IS ASSIGNED TO THIS JOB.
3.14.27 JOB09360 IEF452I 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 IEF452I 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=GHEMAST.FAMAPS.D256, DISP=OLD
001200 //SYSUT2 DD DSN=GHEMAST.FAMAPS.D256, BACKUP, DISP=(,CATLG)
001300 //SYSIN DD DUMMY
***** ***** Bottom of Data *****
```



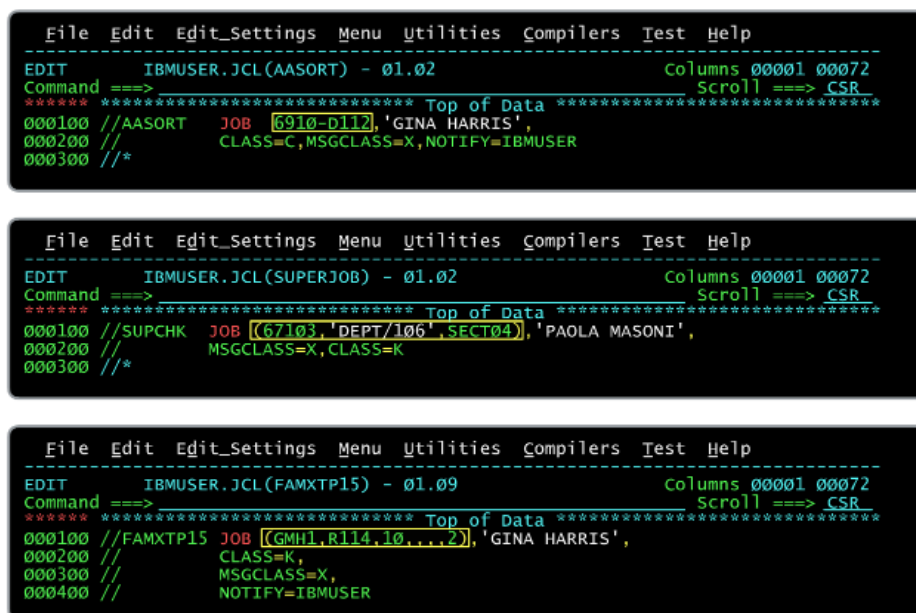
## Accounting Information:

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.



The three screenshots show the JCL editor interface with the following JCL code:

```
File Edit Edit_Settings Menu Utilities Compilers Test Help
EDIT IBMUSER.JCL(AASORT) - 01.02 Columns 00001 00072
Command ==> Scroll ==> CSR
***** Top of Data *****
000100 //AASORT JOB (6910-D112,'GINA HARRIS',
000200 // CLASS=C,MSGCLASS=X,NOTIFY=IBMUSER
000300 //
```

```
File Edit Edit_Settings Menu Utilities Compilers Test Help
EDIT IBMUSER.JCL(SUPERJOB) - 01.02 Columns 00001 00072
Command ==> Scroll ==> CSR
***** Top of Data *****
000100 //SUPCHK JOB (67101,'DEPT/106',SECT04),'PAOLA MASONI',
000200 // MSGCLASS=X,CLASS=K
000300 //
```

```
File Edit Edit_Settings Menu Utilities Compilers Test Help
EDIT IBMUSER.JCL(FAMXTP15) - 01.09 Columns 00001 00072
Command ==> Scroll ==> CSR
***** Top of Data *****
000100 //FAMXTP15 JOB (GMH1,R114,10,,,2),'GINA HARRIS',
000200 // CLASS=K,
000300 // MSGCLASS=X,
000400 // NOTIFY=IBMUSER
```

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:

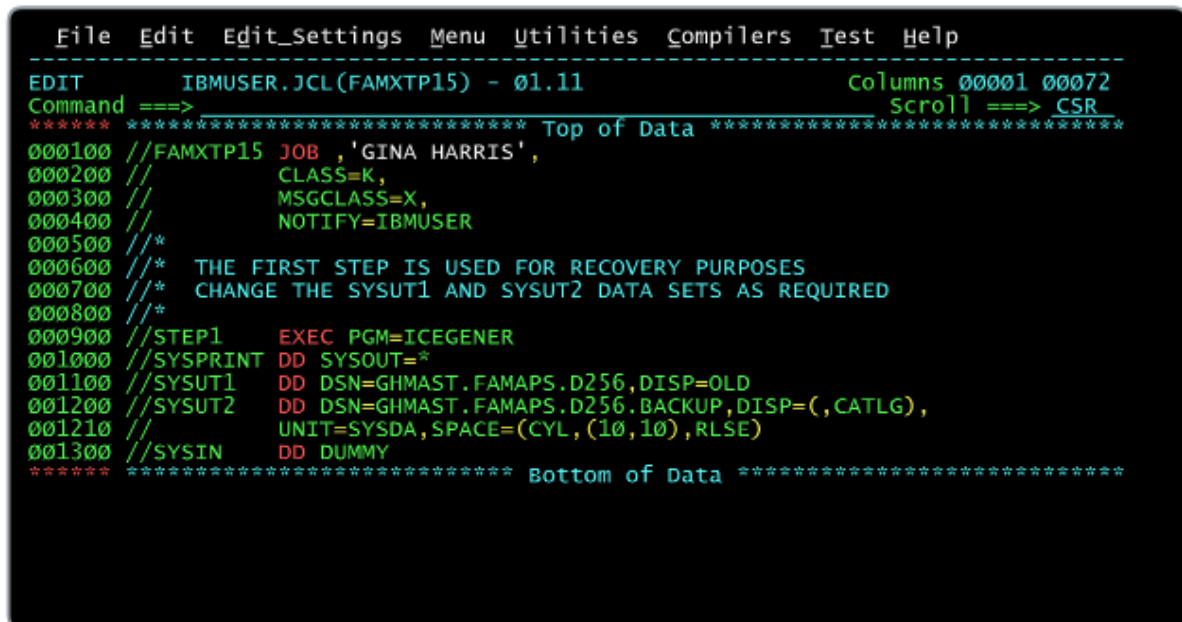
The programmer's name is another optional parameter that may be required by your organization. This field helps to provide identification about the owner of the job. This is often the name of an individual but could also be the name of your group. You can see here that the job's log at the bottom displays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



```
File Edit Edit_Settings Menu Utilities Compilers Test Help
-----
EDIT      IBMUSER.JCL(FAMXTP15) - 01.11                      Columns 00001 00072
Command ==>                                         Scroll ==> CSR
***** Top of Data *****
000100 //FAMXTP15 JOB  ,'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=GHEMAST.FAMAPS.D256,DISP=OLD
001200 //SYSUT2     DD DSN=GHEMAST.FAMAPS.D256.BACKUP,DISP=(,CATLG),
001210 //              UNIT=SYSDA,SPACE=(CYL,(10,10),RLSE)
001300 //SYSIN      DD DUMMY
***** Bottom of Data *****
```

## 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.

```
File Edit Edit_Settings Menu Utilities Compilers Test Help
-----
EDIT          IBMUSER.JCL(FAMXTP15) - 01.12          Columns 00001 00072
Command ==> SUBMIT                                   Scroll ==> CSR
***** ***** Top of Data *****
000100 //FAMXTP15 JOB 6910-D112,'GINA HARRIS',
000200 //          CLASS=K
000300 //*
000400 //* THE FIRST STEP IS USED FOR RECOVERY PURPOSES
000500 //* CHANGE THE SYSUT1 AND SYSUT2 DATA SETS AS REQUIRED
000600 //*
000700 //STEP1      EXEC PGM=ICEGENER

Display Filter View Print Options Search Help
-----
SDSF INPUT QUEUE DISPLAY ALL CLASSES          LINE 1-1 (1)
COMMAND INPUT ==>                             SCROLL ==> CSR
PREFIX=* DEST=(ALL) OWNER=* SYSNAME=
NP  JOBNAME JobID  Owner  JP  C  Pos PhaseName      Status
    FAMXTP15 JOB09417 IBMUSER      7  K      1 AWAIT MAIN SELECT
```

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.

```
File Edit Edit_Settings Menu Utilities Compilers Test Help
-----
EDIT          IBMUSER.JCL(FAMXTP15) - 01.12          Columns 00001 00072
Command ==>                                         Scroll ==> CSR
***** ***** Top of Data *****
000100 //FAMXTP15 JOB 6910-D112,'GINA HARRIS',
000200 //          CLASS=K,
000300 //          MSGCLASS=X
000400 //*
000500 //* THE FIRST STEP IS USED FOR RECOVERY PURPOSES
000600 //* CHANGE THE SYSUT1 AND SYSUT2 DATA SETS AS REQUIRED

Display Filter View Print Options Search Help
-----
SDSF OUTPUT ALL CLASSES ALL FORMS          LINES 79          LINE 1-1 (1)
COMMAND INPUT ==>                             SCROLL ==> CSR
PREFIX=FAM* DEST=(ALL) OWNER=* SORT=CrDate/D SYSNAME=
NP  JOBNAME JobID  Owner  Prty C Forms  Dest      Rec-Cnt
    FAMXTP15 JOB09420 IBMUSER      144 X STD      LOCAL      79
```

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.

```

File Edit Edit_Settings Menu Utilities Compilers Test Help
-----
EDIT      IBMUSER.JCL(FAMXTP15) - 01.12          Columns 00001 00072
Command ==>                                     Scroll ==> CSR
***** Top of Data *****
000100 //FAMXTP15 JOB 6910-D112,'GINA HARRIS',
000200 //          CLASS=K,
000300 //          MSGCLASS=Z
000400 //
000500 //* THE FIRST STEP IS USED FOR RECOVERY PURPOSES
000600 //* CHANGE THE SYSUT1 AND SYSUT2 DATA SETS AS REQUIRED

```

```

File Edit Edit_Settings Menu Utilities Compilers Test Help
-----
EDIT      VENDOR.PARMLIB(JES2420A) - 01.01      Columns 00001 00072
Command ==>                                     Scroll ==> CSR
000481 OUTCLASS(K) OUTDISP=(HOLD,HOLD),OUTPUT=PUNCH
000482 OUTCLASS(R) OUTDISP=(HOLD,HOLD)
000483 OUTCLASS(Z) OUTDISP=(WRITE,WRITE),OUTPUT=DUMMY
000484 OUTDEF    COPIES=100,

```

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 ▶

```

File Edit Edit_Settings Menu Utilities Compilers Test Help
-----
EDIT      VENDOR.PARMLIB(JES2420A) - 01.01      Columns 00001 00072
Command ==>                                     Scroll ==> CSR
000481 OUTCLASS(K) OUTDISP=(HOLD,HOLD),OUTPUT=PUNCH
000482 OUTCLASS(R) OUTDISP=(HOLD,HOLD)
000483 OUTCLASS(Z) OUTDISP=(WRITE,WRITE),OUTPUT=DUMMY
000484 OUTDEF    COPIES=100,

```

```

Display Filter View Print Options Search Help
-----
SDSF HELD OUTPUT DISPLAY ALL CLASSES LINES 49      LINE 1-1 (1)
COMMAND INPUT ==>                                SCROLL ==> CSR
PREFIX=FAM* DEST=(ALL) OWNER=* SORT=CrDate/D SYSNAME=
NP  JOBNAME  JobID  Owner  Prty C ODisp Dest      REC-Cnt  PAGE
   FAMXTP15 JOB09423 IBMUSER 144 R HOLD  LOCAL          49

```

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.

```
File Edit Edit_Settings Menu Utilities Compilers Test Help
-----
EDIT      IBMUSER.JCL(FAMXTP15) - 01.14          Columns 00001 00072
Command ==>                                     Scroll ==> CSR
***** Top of Data *****
000100 //FAMXTP15 JOB 6910-D112,'GINA HARRIS',
000200 //          CLASS=K,
000300 //          MSGCLASS=X,
000400 //          MSGLEVEL=(0,1)
```

```
Display Filter View Print Options Search Help
-----
SDSF JOB DATA SET DISPLAY - JOB FAMXTP15 (JOB09428)  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
   JESMSG LG JES2      2  IBMUSER  X LOCAL    18
   JESJCL  JES2      3  IBMUSER  X LOCAL     8
   JESYSMSG JES2      4  IBMUSER  X LOCAL    21
```

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 JOBCCLASS 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
JESMSG LG 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 (JOB09430) 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
JESMSG LG 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
JESMSG LG 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
JESMSG LG 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.

```
File Edit Edit_Settings Menu Utilities Compilers Test Help
-----
EDIT      IBMUSER.JCL(FAMXTP15) - 01.16          Columns 00001 00072
Command ==>                                     Scroll ==> CSR
***** Top of Data *****
000100 //FAMXTP15 JOB 6910-D112,'GINA HARRIS',
000200 //          CLASS=K,
000300 //          MSGCLASS=X,
000400 //          REGION=6M
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 //          UNIT=SYSDA,SPACE=(CYL,(10,10),RLSE)
001400 //SYSIN DD DUMMY
***** Bottom of Data *****
```

## 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.

```
File Edit Edit_Settings Menu Utilities Compilers Test Help
-----
EDIT      IBMUSER.JCL(FAMXTP15) - 01.18          Columns 00001 00072
Command ==>                                     Scroll ==> CSR
***** Top of Data *****
000100 //FAMXTP15 JOB 6910-D112,'GINA HARRIS',
000200 //          CLASS=K,
000300 //          MSGCLASS=X,
000400 //          REGION=0M,
000500 //          NOTIFY=&SYSUID
000600 //
000700 //* THE FIRST STEP IS USED FOR RECOVERY PURPOSES
000800 //* CHANGE THE SYSUT1 AND SYSUT2 DATA SETS AS REQUIRED
000900 //*
001000 //STEP1 EXEC PGM=ICEGENER
001100 //SYSPRINT DD SYSOUT=*
001200 //SYSUT1 DD DSN=GHMAST.FAMAPS.D256,DISP=OLD
001300 //SYSUT2 DD DSN=GHMAST.FAMAPS.D256.BACKUP,DISP=(,CATLG),
001400 //          UNIT=SYSDA,SPACE=(CYL,(10,10),RLSE)
001500 //SYSIN DD DUMMY
***** Bottom of Data *****
```

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.

```
000100 //FAMXTP15 JOB 4467-D032,'LAYLA SHULZ',
000200 //          CLASS=K,
000300 //          MSGCLASS=X
000400 //          *
000500 //NOT1      NOTIFY EMAIL='ghamlyn@bank.com',TYPE=EMAIL
```

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.

```
000100 //FAMXTP15 JOB 4467-D032,'LAYLA SHULZ',
000200 //          CLASS=K,
000300 //          MSGCLASS=X
000400 //          *
000500 //NOT1      NOTIFY USER=IBMUSER,TYPE=MSG
000600 //NOT2      NOTIFY USER=GTEDSWOT,TYPE=MSG
000700 //NOT3      NOTIFY USER=DZTPRD01,TYPE=MSG
```

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.

```
000100 //FAMXTP15 JOB 4467-D032,'LAYLA SHULZ',
000200 //          CLASS=K,
000300 //          MSGCLASS=X
000400 //          *
000500 //NOT1      NOTIFY EMAIL='ghamlyn@bank.com',TYPE=EMAIL
000600 //          WHEN='(RC=4 | RC=8)'
000700 //          *
000800 //NOT2      NOTIFY USER=LAYLA,TYPE=MSG,
000900 //          WHEN='(ABEND)'
001000 //          *
001100 //NOT3      NOTIFY USER=SIMON,TYPE=MSG,
001200 //          WHEN='(ABENDCC=S0C4 OR ABENDCC=U1024)'
001300 //          *
001400 //NOT4      NOTIFY USER=IBMUSER,TYPE=MSG,
001500 //          WHEN='(!RUN)'
```

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.