Job Control Language (JCL)

Job control language (JCL)

- A scripting language used on IBM mainframes.

- Job control language (JCL) is a set of statements that you code to tell the z/OS operating system about the work you want it to perform.
- These job control statements tell z/OS where to find the appropriate input, how to process that input (that is, what program or programs to run), and what to do with the resulting output.
- JCL is a mean of communication between an application program and the computers operating system (MVS-multiple virtual storage).

JCL features

- Consists of a set of statements called as Job Control Statements
- Group of related JCL statements is known as Job
- JCL consists of one or more Jobs
- Job consists of Job steps to execute the instructions (tasks)

What JCL does?

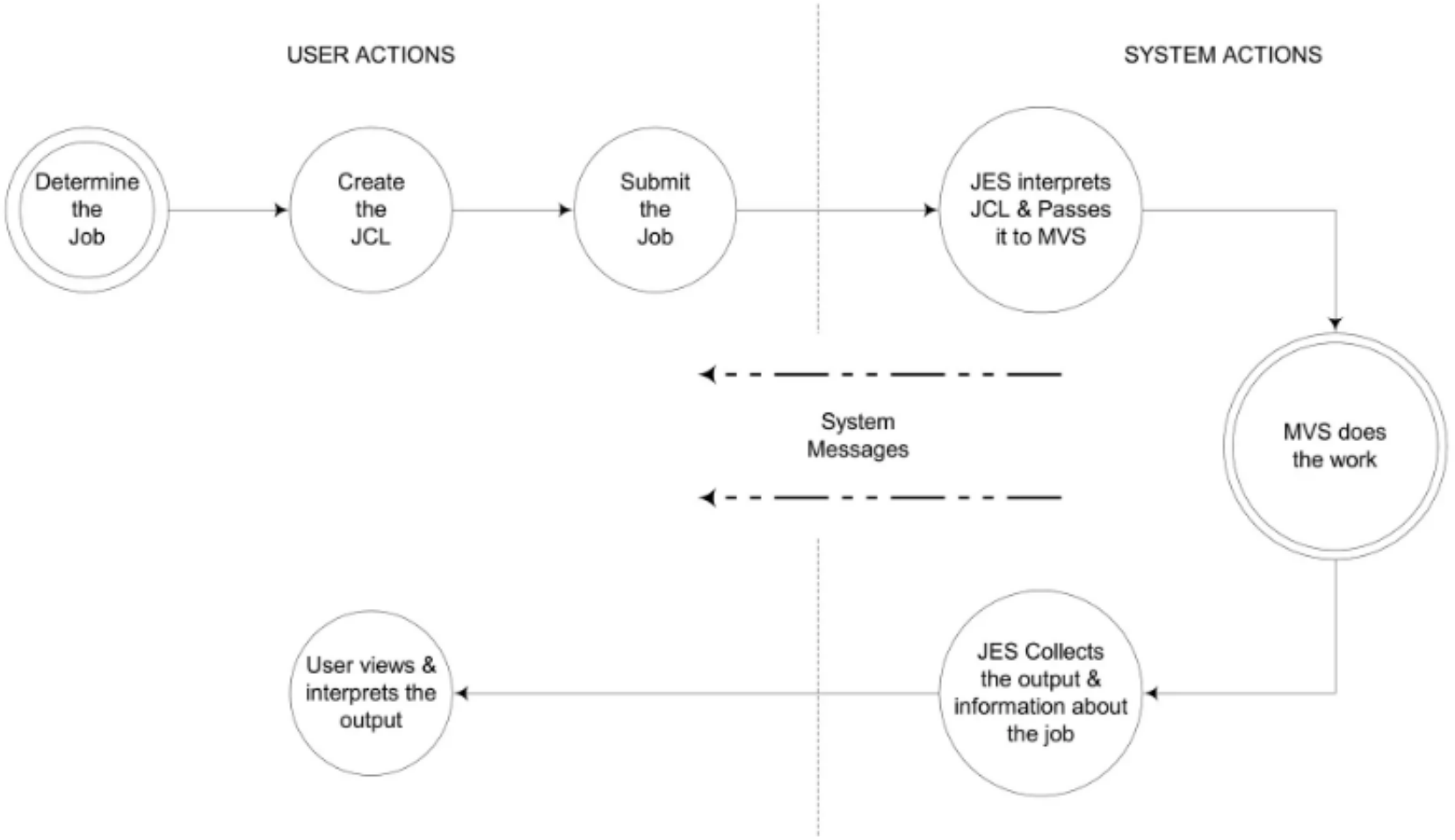
- JCL used to communicate with the O.S. about your requirements for running a job.
  - Telling O.S. who you are (allows priority distinctions).
  - Telling O.S. your space and time requirements.
  - Telling O.S. what programs (load modules) you need
  - Telling O.S. what data sets are needed by program(s)

What can be done with JCL?

1. Submit a job to the operating system.
2. Request resources needed to run the job.
3. Control the systems processing of the job.
4. You can pass parameter values through JCL to application program using PARM parameter in EXEC statement.

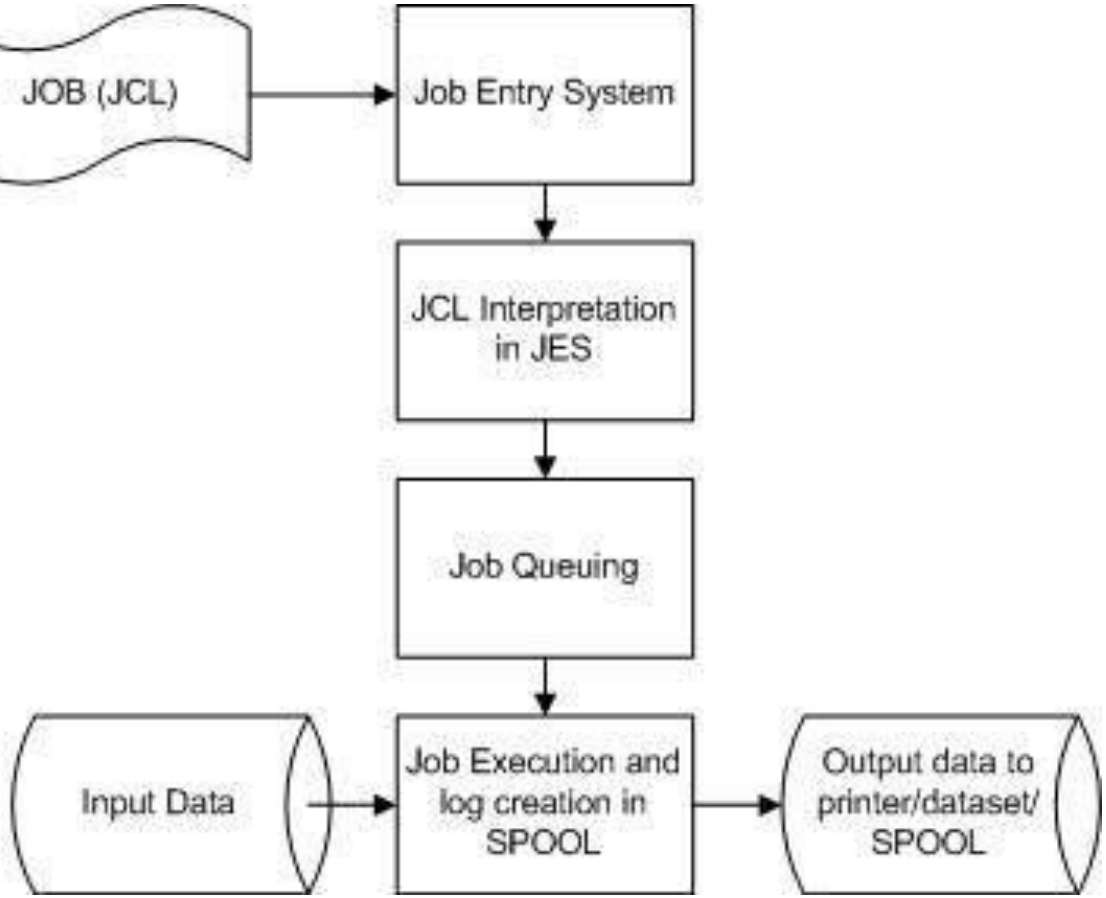
job

- A *job* is a separately executable collection of work defined by a user, and run by a computer.
- A job is defined as execution of one or more related programs in a sequence.
- Job flow :
- During execution a job goes through the following phases:
  - Input
  - Conversion
  - Execution
  - Output


USER ACTIONS
SYSTEM ACTIONS
Determine the Job
Create the JCL
Submit the Job
JES interprets JCL & Passes it to MVS
MVS does the work
System Messages
JES Collects the output & information about the job
User views & interprets the output

Job processing

The Operating System uses **Job Entry System (JES)** to receive jobs into the Operating System, to schedule them for processing and to control the output.
JES2 & JES3 are two JES provided by IBM
JES are the part of the OS that manages Batch jobs and outputs



Job processing

- **Job Submission -** Submitting the JCL to JES.
- **Job Conversion -** The JCL along with the PROC is converted into an interpreted text to be understood by JES and stored into a dataset, which we call as SPOOL.
- **Job Queuing -** JES decides the priority of the job based on CLASS and PRTY parameters in the JOB statement. The JCL errors are checked and the job is scheduled into the job queue if there are no errors.
- **Job Execution -** When the job reaches its highest priority, it is taken up for execution from the job queue. The JCL is read from the SPOOL, the program is executed and the output is redirected to the corresponding output destination as specified in the JCL.
- **Purging -** When the job is complete, the allocated resources and the JES SPOOL space is released. In order to store the job log, we need to copy the job log to another dataset before it is released from the SPOOL.

Installing JCL on Windows/Linux

- Free Mainframe Emulators available for Windows
- Download and install the Hercules emulator, which is available from the Hercules' home site - : www.hercules-390.eu

Types of JCL statements:

- **JOB**: Marks the beginning of a job and identifies the job name.
- **EXEC**: Marks the beginning of a job step and specifies the name of the program to be run.
- **DD**: Describes data sets to be used within individual job steps. Data sets are files that contain programs and data.

Job steps and job streams

- A job is identified by a JOB statement.
- Each job consists of one or more individual job steps.
- You define each job step with an EXEC statement, which is associated with the execution of one program.
- You define the data sets required for each job step by using a DD statement.
- On z/OS, you can submit jobs to the system through several facilities, including the Time Sharing Option/Extensions (known as TSO/E) and its menu-driven interface, Interactive System Productivity Facility (known as ISPF).
- In a z/OS system, each user is granted a user ID and a password authorized for TSO logon.

Defining job control statements

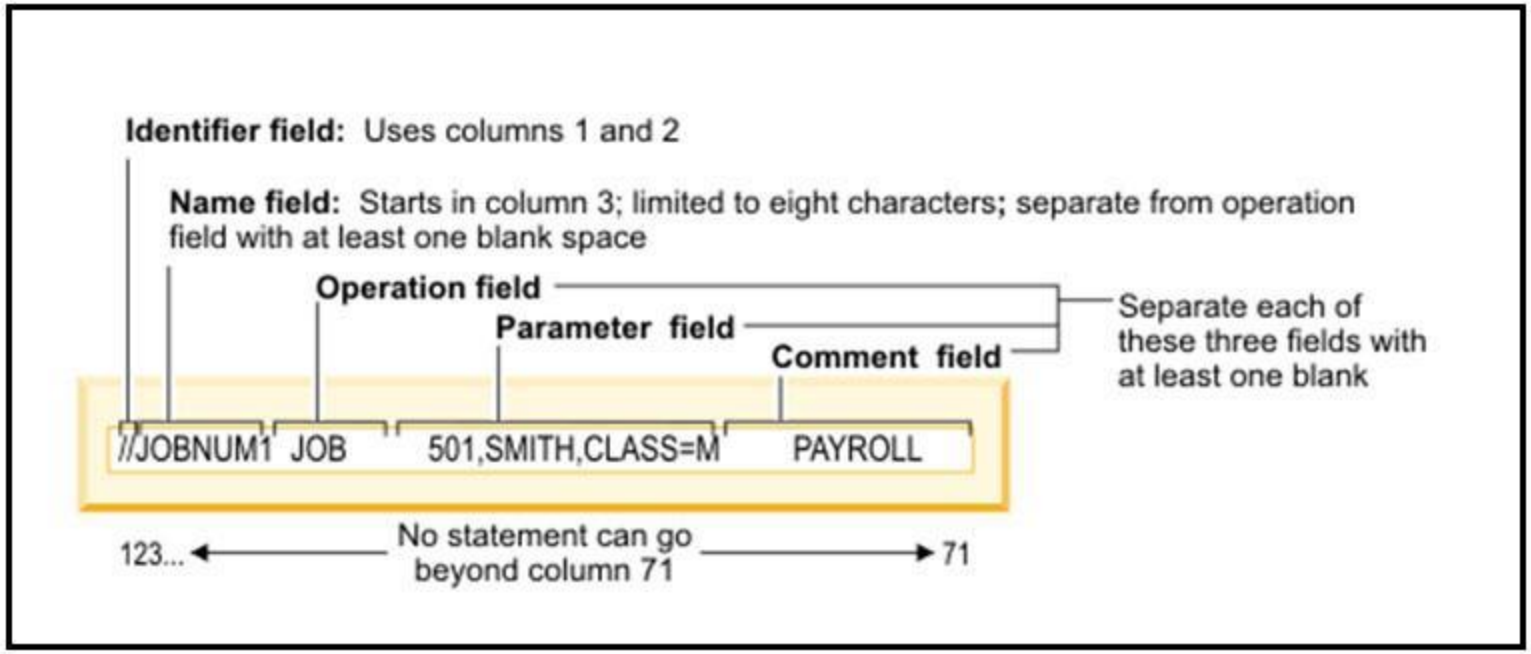
Grouping programs

- You might want to group programs together in a single job when those programs:
  - Have to run sequentially
  - Use the same resources, such as data sets
  - Are dependent on each other
- Consider two programs:
- Program 1 takes input from the INPUT data set and uses it to create the INOUT data set.
- Program 2 uses the data set created by Program 1 as its input and creates two more data sets called OUTPUT1 and OUTPUT2.
- Because these programs must run sequentially and Program 2 is dependent on Program 1, you should group them into a single job.

Grouping programs

JCL Statement Structure:

- JCL statements are coded in 80 bytes.
- 72 of the 80 columns are available to code JCL.
  Last 8 columns are reserved for an optional sequence number.



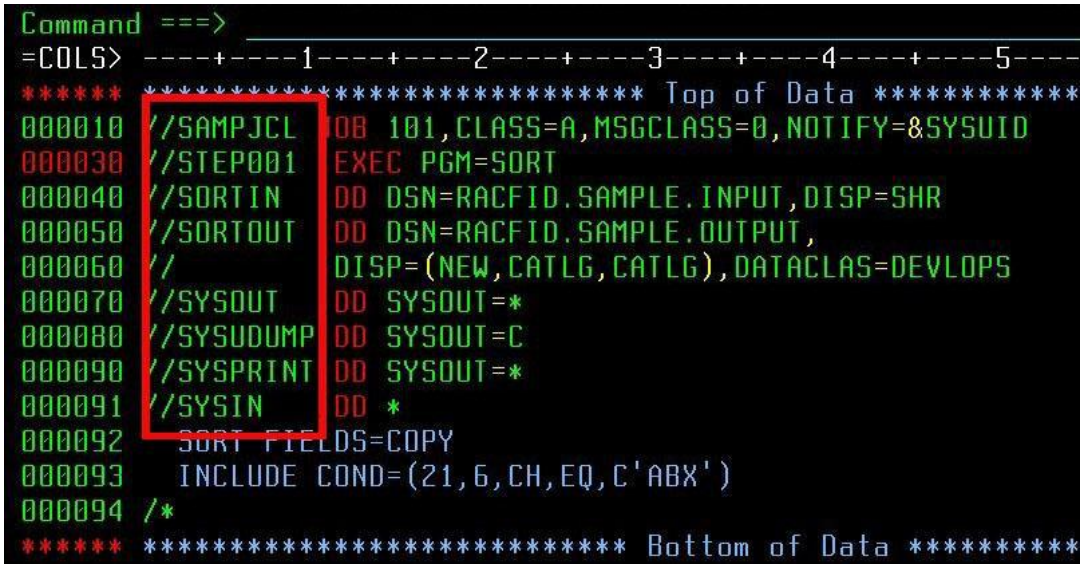
A comment field must follow the parameter field and must be preceded by one or more blanks.

JCL Statement Structure:

- Identifier Field :It occupies the first two character positions and must contain two slashes (//).
  - //* as comment statement
  - .* to mark end of data
- Name Field: A name field must begin in column 3, right after the identifier field. It consists of one to eight characters, which may be letters, numbers or national characters (#, @, and $) Also, the first character of a name should be a letter or a national character.
- Operation Field:JOB, EXEC, and DD are the different operation fields. it should be separated from the name field by at least one blank. Conventionally, the operation field should be started in column 12.

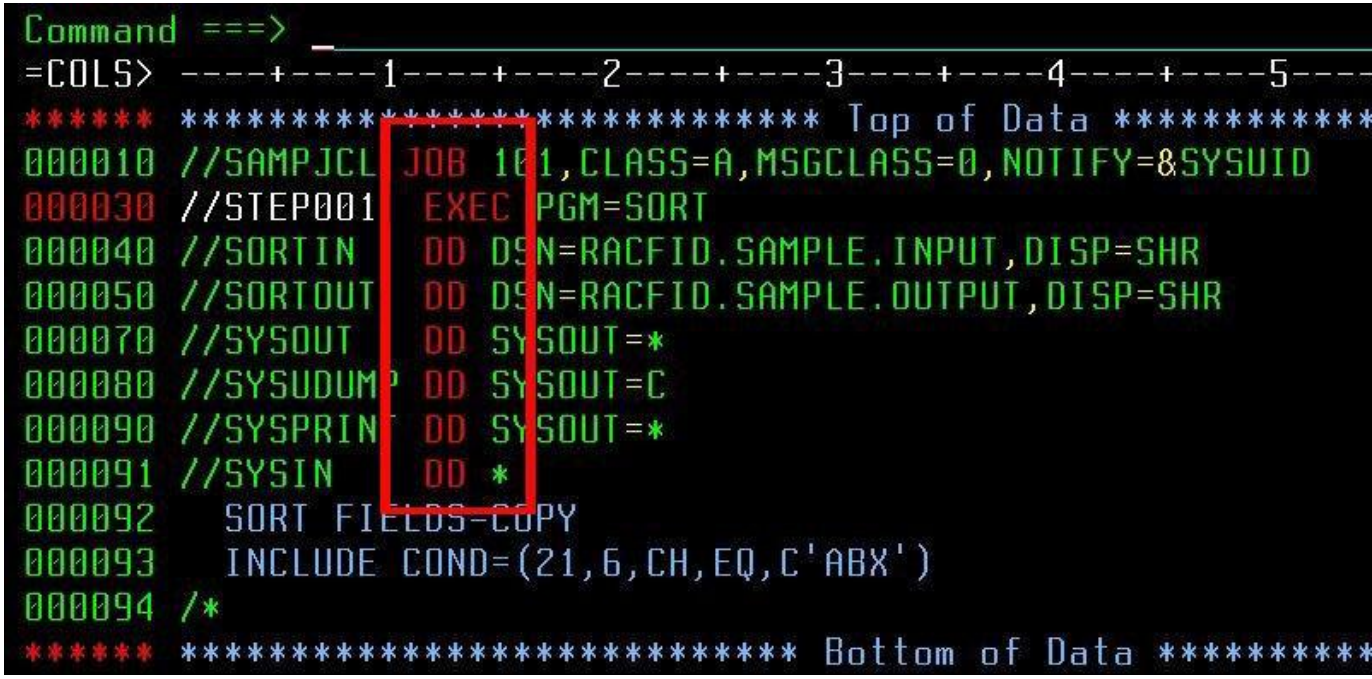
Rules for coding the name field in JCL statements

- The name must begin in position 3 of the JCL statement.
- The name can be one through eight characters in length.
- The first character in the name must be an alphabetic character (the letters A through Z) or a special character (the symbols #, @, and $)
- The remaining characters can be alphanumeric or special characters
- Alphabetic characters must be uppercase characters; lower case cannot be used in JCL except in a few specific parameters.
- Blank spaces cannot be included in a name.



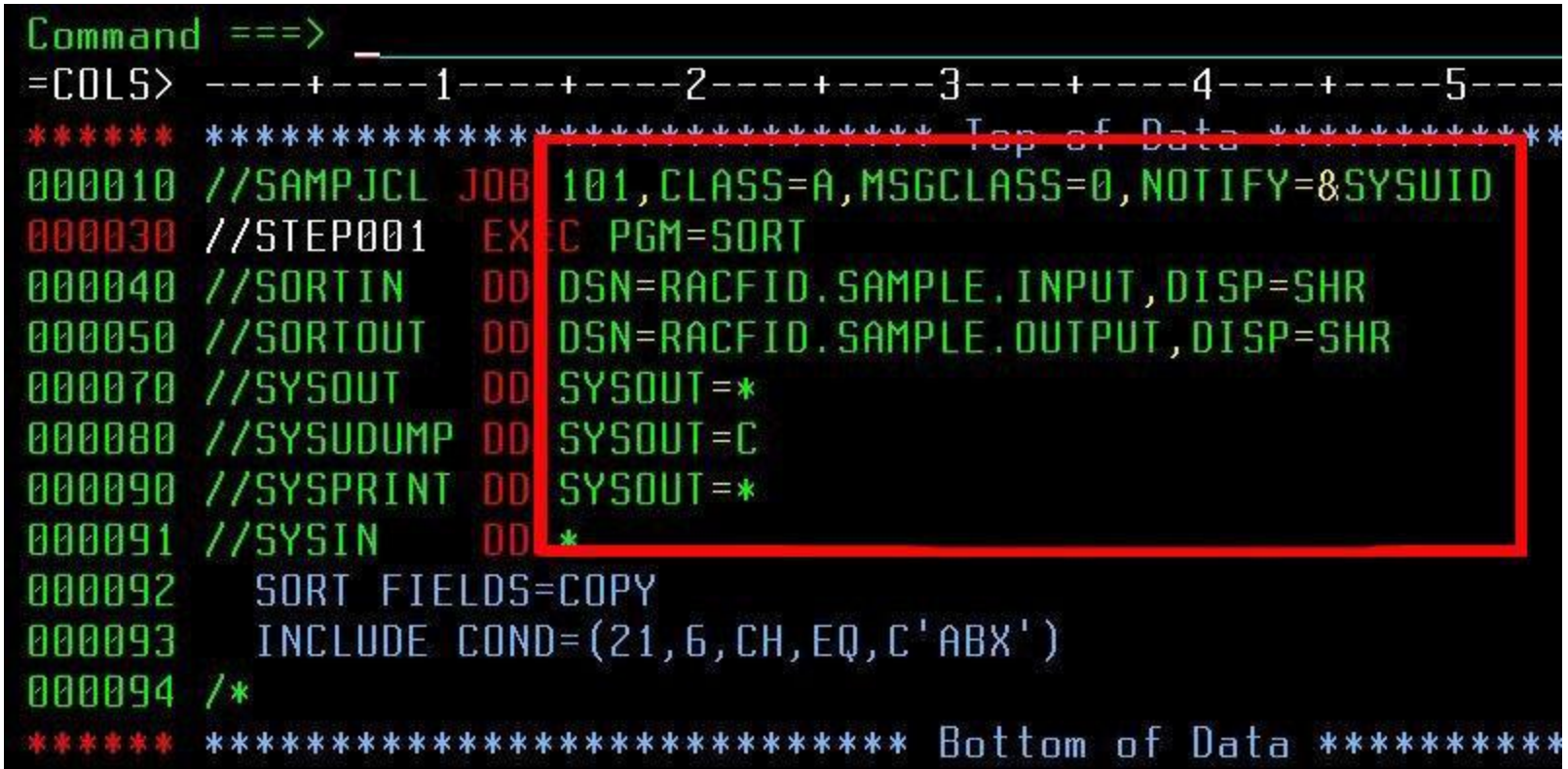
Identifier field and
Name field

Operation field



JCL Statement Structure:

- Parameters Field: operand field,it begins at least one position after the end of the operation field and can extend into column 71
- When a parameters field consists of more than one parameter, the individual parameters should be separated with commas.



Structure of a JCL

```
//SAMPJCL JOB 1,CLASS=6,MSGCLASS=0,NOTIFY=&SYSUID (1)
//*                                                (2)
//STEP010 EXEC PGM=SORT                            (3)
//SORTIN DD DSN=JCL.SAMPLE.INPUT,DISP=SHR          (4)
//SORTOUT DD DSN=JCL.SAMPLE.OUTPUT,                (5)
// DISP=(NEW,CATLG,CATLG),DATACLAS=DSIZE50 //SYSOUT
   DD SYSOUT=*                                     (6)
//SYSUDUMP DD SYSOUT=C                             (6)
//SYSPRINT DD SYSOUT=*                             (6)
//SYSIN DD *                                       (6)
SORT FIELDS=COPY INCLUDE COND=(28,3,CH,EQ,C'XXX')
/*                                                 (7)
```

Structure of a JCL

- **(1) JOB statement** - Specifies the information required for SPOOLing of the job such as job id, priority of execution, user-id to be notified upon completion of the job.
- **(2) //* statement** - This is a comment statement,can be coded till column 80
- **(3) EXEC statement** - Specifies the PROC/Program to be executed. In the above example, a SORT program is being executed (i.e., sorting the input data in a particular order)
- **(4) Input DD statement** - Specifies the type of input to be passed to the program mentioned in (3). In the above example, a Physical Sequential (PS) file is passed as input in shared mode (DISP = SHR).
- **(5) Output DD statement** - Specifies the type of output to be produced by the program upon execution. In the above example, a PS file is created. If a statement extends beyond the 70th position in a line, then it is continued in the next line, which should start with "//" followed by one or more spaces.
- **(6)** There can be other types of DD statements to specify additional information to the program (In the above example: The SORT condition is specified in the SYSIN DD statement) and to specify the destination for error/execution log (Example: SYSUDUMP/SYSPRINT). DD statements can be contained in a dataset (mainframe file) or as in stream data (information hard-coded within the JCL) as given in above example.
- **(7) /*** marks the end of in stream data.

JOB Parameter Types

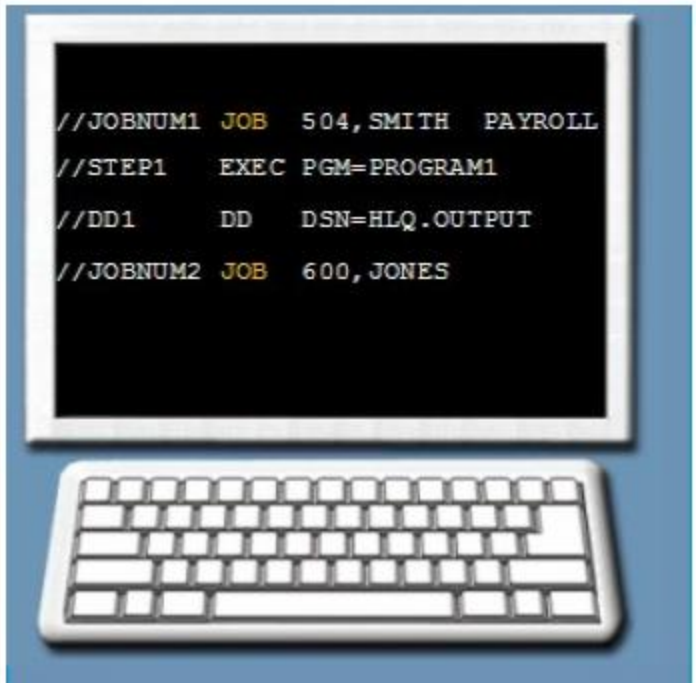
- Positional Parameters: Appears at pre-defined position and order in the statement.
- Positional Parameters are present in JOB and EXEC statements. In the above example, PGM is a positional parameter coded after the **EXEC** keyword.
- A positional parameter consists of: Characters that appear in uppercase in the syntax
- For example
  - DATA on a DD statement.
  - programmer's-name on a JOB statement.
  - PGM=program-name on an EXEC statement.

JOB Parameter Types

- Keyword Parameters :They are coded after the positional parameters, but can appear in any order. Keyword parameters can be omitted if not required.
- The generic syntax is KEYWORD= *value*
- CLASS, MSGCLASS and NOTIFY are keyword parameters of JOB statement. There can be keyword parameters in EXEC statement as well.

The JOB statement

- The JOB statement typically provides accounting information and details about the person who is submitting the job, including name, user ID, and password. This information applies to all job steps within the job. The JOB statement also may contain any comments that help describe the statement.
- In this diagram, the first JOB statement, named JOBNUM1, includes the accounting number (504), programmer name (SMITH), and a comment (PAYROLL) that further describes the job.
- The end of a job is marked by another JOB statement (in this diagram, JOBNUM2), a null statement, or by the end of the file containing the JCL.



The JOB statement

- format of job statement below,
- //Job_name **JOB** [accounting information] [,programmer name] [,USER=user-id] [,PASSWORD=password] [,NOTIFY=user-id] [,CLASS=class] [,MSGCLASS=class] [,MSGLEVEL=(stmt,msg)] [,TIME=(mins,secs)]

- JOB statement is getting extended beyond the 70th position in a line,so we continue in the next line which should start with "//" followed by one or more spaces.

The JOB statement

Positional Parameter	Description
Account information	This refers to the person or group to which the CPU time is owed. It is set as per the rules of the company owning the mainframes. If it is specified as (*), then it takes the id of the user, who has currently logged into the Mainframe Terminal.
Programmer name	This identifies the person or group, who is in charge of the JCL. This is not a mandatory parameter and can be replaced by a comma.

Keyword Parameter

The Job Statement (or Job Card) can contain the following Keyword parameters

- CLASS
- TIME
- REGION
- MSGLEVEL
- COND
- TYPRUN
- PRTY
- ADDRSPC
- MSGCLASS
- NOTIFY
- RESTART

Keyword Parameter

Keyword Parameter	Description
CLASS	Based on the time duration and the number of resources required by the job, companies assign different job classes. Valid values for CLASS parameter are A to Z characters and 0 to 9 numeric (of length 1). Following is the syntax: **CLASS=0 to 9 | A to Z**
PRTY	To specify the priority of the job within a job class. If this parameter is not specified, then the job is added to the end of the queue in the specified CLASS. Following is the syntax: **PRTY=N** Where N is a number in between 0 to 15 and higher the number, higher is the priority.
NOTIFY	The system sends the success or failure message (Maximum Condition Code) to the user specified in this parameter. Following is the syntax: **NOTIFY="userid | &SYSUID"** Here system sends the message to the user "userid" but if we use NOTIFY = &SYSUID, then the message is sent to the user submitting the JCL.

Keyword Parameter

- Class A -- 2 sec
- Class B -- 7 sec
- Class C -- 19 sec
- Class D -- 46 sec
- Class E -- 5 mins
- Class H -- Held Class
- Class T -- 5 mins (for tapes only)
- Class R -- 5 mins. (cartridges only)

Keyword Parameter

Keyword Parameters - PRTY

- To assign the selection priority to the JOB
- Syntax:

  PRTY=(number)

- Examples:

  //JOB4 JOB ACCT1,CLASS=E,PRTY=12...

Keyword Parameter

Keyword Parameter	Description
MSGCLASS	To specify the output destination for the system and Job messages when the job is complete. Following is the syntax: **MSGCLASS=CLASS** Valid values of CLASS can be from "A" to "Z" and "0" to "9". MSGCLASS = Y can be set as a class to send the job log to the JMR (JOBLOG Management and Retrieval: a repository within mainframes to store the job statistics).
MSGLEVEL	Specifies the type of messages to be written to the output destination specified in the MSGCLASS. Following is the syntax: **MSGLEVEL=(*ST*, *MSG*)** *ST* = Type of statements written to output log • When *ST* = 0, Job statements only. • When *ST* = 1, JCL along with symbolic parameters expanded. • When *ST* = 2, Input JCL only. *MSG* = Type of messages written to output log. • When *MSG* = 0, Allocation and Termination messages written upon abnormal job completion. • When *MSG* = 1, Allocation and Termination messages written irrespective of the nature of job completion.

Keyword Parameter

Keyword Parameters - MSGLEVEL

- Details in the listing of Joblog can be controlled, using MSGLEVEL parameter
- Syntax:

  MSGLEVEL=([statements][,messages])
- Job log contains the following information :
  - JCL Statements
  - Procedure Statements for any procedure job step calls
  - Messages about Job control statements, allocation of devices and volumes, execution and termination of job steps and disposition of datasets

Keyword Parameter

Statements sub-parameter - MSGLEVEL

MSGLEVEL=([*statements*][,messages])

Only job statement is printed for job(job1)

//JOB1 JOB AC1,MSGCLASS=X,MSGLEVEL=(0,1)..

NO parentheses ??

//JOB2 JOB AC1,MSGCLASS=X,MSGLEVEL=2..

Both Statements and Messages are printed

//JOB3 JOB AC1,MSGCLASS=X,MSGLEVEL=(1,1)

Keyword Parameter

MSGLEVEL=([statements][,messages])

Only JCL messages are printed for job(job1)

//JOB1 JOB AC1,MSGCLASS=X,MSGLEVEL=(0,0)..

No value for messages Sub-parameter. Takes default.

//JOB2 JOB ACC1,MSGCLASS=X,MSGLEVEL=2..

All messages are printed

//JOB3 JOB AC1,MSGCLASS=X,MSGLEVEL=(,1)

Keyword parameter

Keyword Parameter	Description
TYPRUN	Specifies a special processing for the job. Following is the syntax: **TYPRUN = SCAN | HOLD** Where SCAN and HOLD has the following description • TYPRUN = SCAN checks the syntax errors of the JCL without executing it. • TYPRUN = HOLD puts the job on HOLD in the job queue.To release the job, "A" can be typed against the job in the SPOOL, which will bring the job to execution.
TIME	Specifies the time span to be used by the processor to execute the job. Following is the syntax: **TIME=(mm, ss) or TIME=ss** Where mm = minutes and ss = seconds
REGION	Specifies the address space required to run a job step within the job. Following is the syntax: **REGION=nK | nM** K is kilobyte and M is Megabyte. When REGION = 0K or 0M, largest address space is provided for execution

Keyword Parameter

Syntax:

TIME=([*minutes*][,*seconds*])

Examples:

```
//JOB1 JOB ACCT1,CLASS=E,TIME=(2,30)....
```

Allows this job to use 2 min 30 sec of CPU time

```
//JOB2 JOB ACCT1,CLASS=E,TIME=(,30)...
```

Allows this job to use 30 sec of CPU time

```
//JOB3 JOB ACCT1,CLASS=E,TIME=2....
```

Allows this job to use 2 mins of CPU time

Keyword Parameter

- Syntax:

  TIME=([1440]| [NOLIMIT] | [MAXIMUM])
- Examples:

  //JOB4 JOB ACCT1,CLASS=E,TIME=1440....

  //JOB5 JOB ACCT1,CLASS=E,TIME=NOLIMIT....

  These jobs can use the processor for unlimited amount of time

  //JOB6 JOB ACCT1,CLASS=E,TIME=MAXIMUM....

  This job can run for maximum amount of time that is 357912 minutes.

Example **The JOB statement**

```
//URMISAMP JOB (*),"tutpoint",CLASS=6,PRTY=10,NOTIFY=&SYSUID,
// MSGCLASS=X,MSGLEVEL=(1,1),TYPRUN=SCAN,
// TIME=(3,0),REGION=10K
```

Keyword Parameter

Keyword Parameters - REGION

- Syntax:

  REGION={*valueK* | *valueM*}

- Examples:

  //JOB1 JOB ACT1,CLASS=A,ADDRSPC=REAL,REGION=200K.

  Specifies 200 kilo bytes of ***central storage*** is required for this job(JOB1)

  //JOB2 JOB ACT1,'TRG',CLASS=A,REGION=20M.

  Specifies 20 mega bytes of ***virtual storage*** is required for this job(JOB2)

Keyword Parameter

Keyword Parameters - ADDRSPC

- Syntax:

  ADDRSPC={*REAL* | *VIRT*}

- Examples:

  //JOB1 JOB ACT1,CLASS=A,ADDRSPC=REAL,REGION=20K

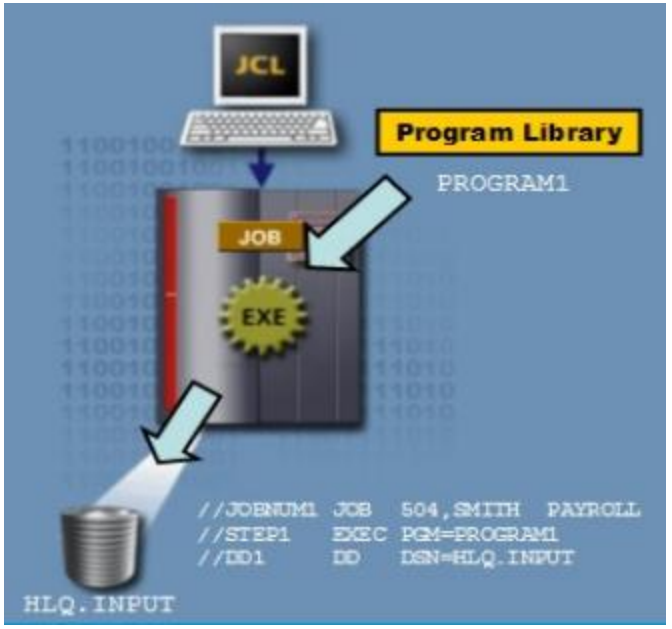
  Non pageable central storage of 20k
  is required for this job(JOB1)

  //JOB1 JOB ACT1,CLASS=A,ADDRSPC=VIRT,REGION=20K

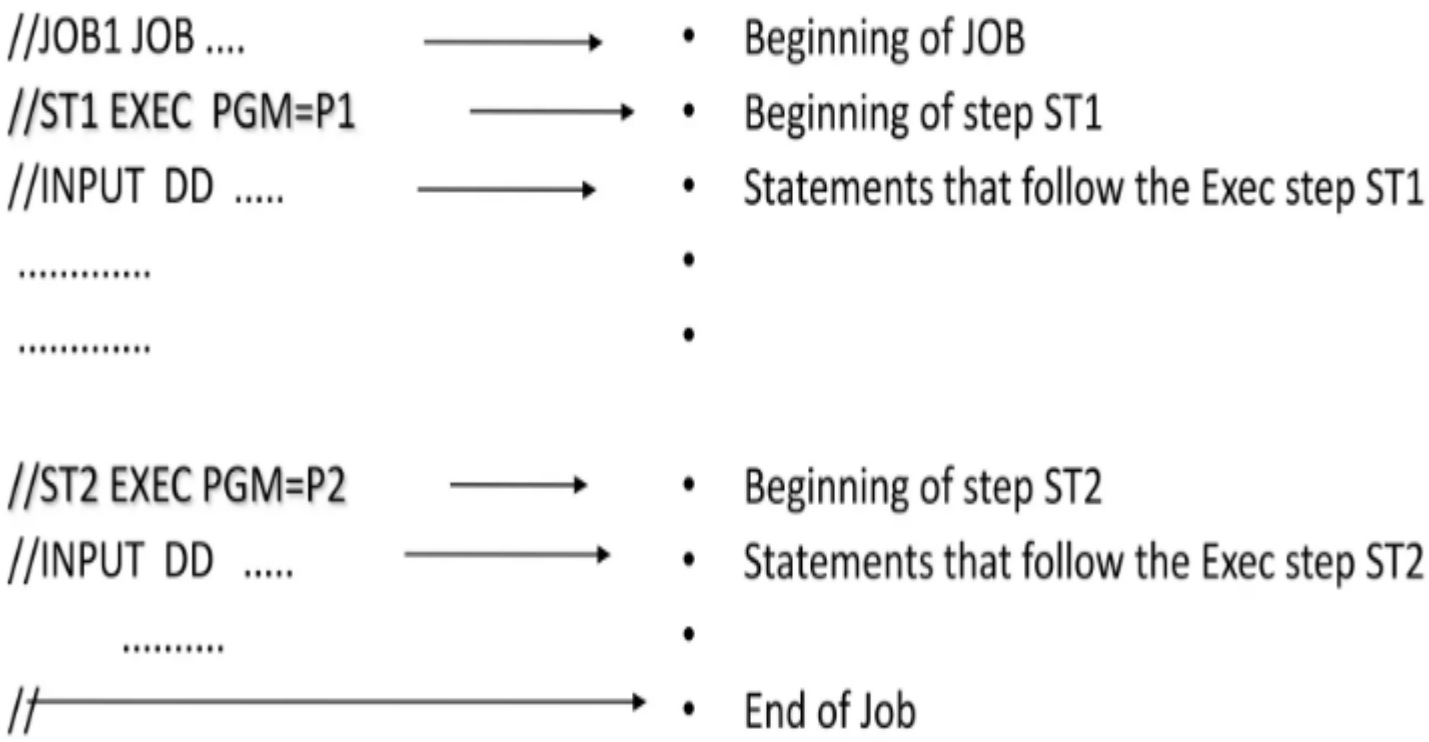
  Pageable virtual storage of 20k
  is required for this job(JOB1)

The EXEC statement

- Each JCL can be made of many job steps.
- Each job step can execute a program directly or can call a procedure, which in turn executes one or more programs (job steps).
- The statement, which holds the job step program/procedure information is the **EXEC statement.**
- The purpose of the EXEC statement is to provide required information for the program/procedure executed in the job step.



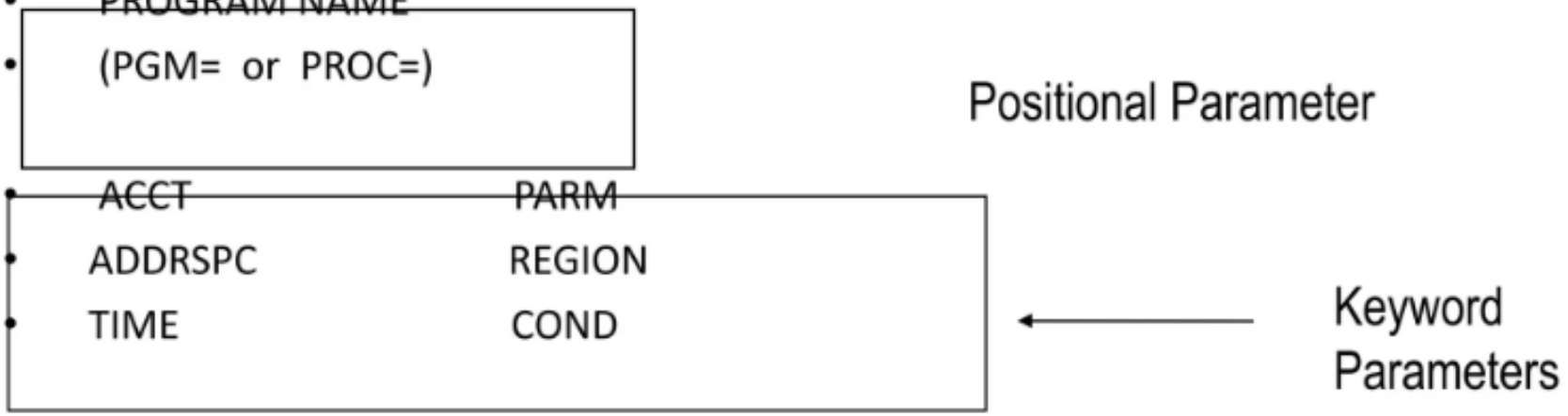
The EXEC statement



The PGM parameter identifies the name of the program that is to be executed.
Max characters –20

The EXEC statement: Parameter field

- //STEPNAME EXEC <Parameters>
- The following list shows the important parameters on Exec statement



The EXEC statement: Parameter field

Syntax:

PGM=<*program name*> OR

PROC=<*procedure name*> OR

<*procedure name*>

Examples:

//STEP1 EXEC PGM=TEST

STEP1 executes program **TEST**

//STEP2 EXEC PROC=COMPILE

//STEP3 EXEC COMPILE

STEP2 & STEP3 invokes JCL procedure **COMPILE**

The EXEC statement: Parameter field

The EXEC statement can contain the following Keyword parameters :

- ACCT
- PARM
- ADDRSPC
- REGION
- TIME
- COND

The EXEC statement

Positional Parameter	Description
PGM	This refers to the program name to be executed in the job step.
PROC	This refers to the procedure name to be executed in the job step.

Keyword Parameter	Description
PARM	Used to provide parameterized data to the program that is being executed in the job step. For example the value "CUST1000" is passed as an alphanumeric value to the program.
ADDRSPC	This is used to specify whether the job step require virtual or real storage for execution Following is the syntax: **ADDRSPC=VIRT | REAL** When an ADDRSPC is not coded, VIRT is the default one.
ACCT	This specifies the accounting information of the job step. Following is the synta **ACCT=(userid)** This is similar to the positional parameter **accounting information** in the JC statement

The EXEC statement: Parameter field

ACCT – provides accounting information for the job step

ADDRSPC – prevents the step from being paged

REGION - specifies the region size to allocate to a job step

TIME - imposes a CPU time limit on the job step

The EXEC statement: Parameter field

Syntax:

PARM=(<*sub parameter*,[*sub parameter*]>)

Examples:

//STEP1 EXEC PGM=P1,PARM=TRGXYZ

//STEP2 EXEC PGM=P2,PARM=(10,'25/01/01')

25/01/01 is enclosed in apostrophes because it contains special characters '/'

//STEP3 EXEC PGM=P3,PARM='25&&45'

Passes string **25&45** to program **P3**

COND parameter

1. COND parameter is JCL can be used to skip a step based on the return code from the previous step or steps of the JCL.

2. This COND parameter can also be used to skip a step based on the return code from a previous step or steps in PROC.

3. You can use the COND parameter to run a job even though, any of the previous steps have Abended (Abend means Abnormal Termination).

4. With the COND parameter, you can also run a step only and only if any of the previous steps have Abended.

COND parameter

You can set the COND parameter at –

1. **JOB level** –JOB Statement(JOB card).
2. **STEP level** – EXEC statement.

- It is a good practice to code the COND parameter at STEP Level but based on your requirement, you can code this at JOB level or STEP level or at both the places.

COND parameter

- When a job completes, a return code is set based on the status of execution. The return code can be a number between 0 (successful execution) to 4095 (non-zero shows error condition).
- 0 = Normal - all OK
- 4 = Warning - minor errors or problems.
- 8 = Error - significant errors or problems.
- 12 = Severe error - major errors or problems, the results should not be trusted.
- 16 = Terminal error - very serious problems, do not use the results.

COND parameter

- Syntax
- COND=(rc,logical-operator) or
- COND=(rc,logical-operator,stepname) or
- COND=EVEN or COND=ONLY

- **rc** : This is the return code
- **logical-operator** : This can be GT (Greater Than), GE (Greater than or Equal to), EQ (Equal to), LT (Lesser Than), LE (Lesser than or Equal to) or NE (Not Equal to).
- **stepname** : This is the job step whose return code is used in the test.

COND parameter

- When COND=EVEN is coded, the current job step is executed, even if any of the previous steps abnormally terminate.
- When COND=ONLY is coded, the current job step is executed, only when any of the previous steps abnormally terminate

Examples:

```
//STP1 EXEC PGM=P1,COND=EVEN
```

STP1 is executed even if a preceding step abends

```
//STP2 EXEC PGM=P3,COND=((4,LE,STP1),ONLY)
```

STP2 is executed only if a preceding step abends and return code of STP1 is less than 4

COND parameter

```
//CNDSAMP JOB CLASS=6,NOTIFY=&SYSUID
//*
//STP01 EXEC PGM=SORT
//* Assuming STP01 ends with RC0.

//STP02 EXEC PGM=MYCOBB,COND=(0,EQ,STP01)
//* In STP02, condition evaluates to TRUE and step bypassed.

//STP03 EXEC PGM=IEBGENER,COND=((10,LT,STP01),EVEN)
//* In STP03, condition (10,LT,STP01) evaluates to true,
//* hence the step is bypassed.
```

COND inside EXEC statement

```
//CNDSAMP JOB CLASS=6,NOTIFY=&SYSUID
//*
//STP01 EXEC PGM=SORT
//* Assuming STP01 ends with RC0
 //STP02 EXEC PGM=MYCOBB,COND=(0,EQ,STP01)
//* In STP02, condition evaluates to TRUE and step bypassed.
 //STP03 EXEC
    PGM=IEBGENER,COND=((10,LT,STP01),(10,GT,STP02))
 //* In STP03, first condition fails and hence STP03 executes.
//* Since STP02 is bypassed, the condition (10,GT,STP02) in
 //* STP03 is not tested.
```

IF-THEN-ELSE Construct

- Syntax

//name IF condition THEN

list of statements //* action to be taken when condition is true

//name ELSE

list of statements //* action to be taken when condition is false

//name ENDIF

- **name** : This is optional and a name can have 1 to 8 alphanumeric characters starting with alphabet, #,$ or @.
- **Condition** : A condition will have a format: **KEYWORD OPERATOR VALUE**, where **KEYWORDS** can be RC (Return Code), ABENDCC (System or user completion code), ABEND, RUN (step started execution). An **OPERATOR** can be logical operator (AND (&), OR (|)) or relational operator (<, <=, >, >=, <>).

IF condition example

```
//JOBEXP JOB
//STEP01  EXEC MYPROC01
//COND01  IF RC = 0 THEN
//STEP02  EXEC MYPROC02
//CONDE   ELSE
//STEP03  EXEC MYPROC03
//       ENDIF
```

- After STEP01 is completed we check the Return code of STEP01 step. if return code is zero then STEP02 step will execute else step STEP03 will be executed.
- Either the THEN clause or ELSE clause must contain at least one EXEC statement.

IF condition example

- **ABENDCC** - Using ABENDCC we can check System/User completion codes.

```
IF ABENDCC = S0C7 THEN
```

- **ABEND**- checks for an abnormal end of a program

```
IF ABEND THEN
IF STEP4.PROCAS01.ABEND = TRUE THEN
```

IF condition example

```
//MATEKSD JOB MSGLEVEL=(1,1),NOTIFY=&SYSUID
//*
//* EXAMPLE TO SHOW IF CONDITION IN JCL
//*
//STEP01   EXEC PGM=IFCOND1
//*
//IFSTMT1 IF STEP01.RC = 0  THEN  ---> This will check if the Return Code(RC)
//STEP02   EXEC PGM=IFCOND2           from STEP01 is 0, if yes then execute
//STEP03   EXEC PGM=IFCOND3           STEP02 and STEP03.
//       ENDIF
//STEP04   EXEC PGM=IFCOND4
//STEP05   EXEC PGM=IFCOND5
//*
//IFSTMT2 IF STEP05.RC = 04 THEN   ---> This will check if the Return Code(RC)
//STEP06   EXEC PGM=IFCOND6            from STEP05 is 04, if yes then execute
//       ELSE                    STEP06, else execute STEP07.
//STEP07   EXEC PGM=IFCOND7
//       ENDIF
//STEP08   EXEC PGM=IFCOND8
//*
```

IF condition example

```
//JOBIBM JOB CLASS=C,NOTIFY=&SYSUID
//*
//PRC1   PROC
//PST1            EXEC PGM=IDCAMS
//PST2            EXEC PGM=SORT
//       PEND
//STEP01  EXEC PGM=SORT
//IF1    IF STEP01.RC = 0 THEN
//STEP02     EXEC PGM=MYCOBB,PARM=321
//       ENDIF
//IF2    IF STEP01.RUN THEN
//STEP03a    EXEC PGM=IDCAMS
//STEP03b    EXEC PGM=SORT
//       ENDIF
//IF3    IF STEP03b.!ABEND THEN
//STEP04     EXEC PGM=MYCOBB,PARM=654
//       ELSE
//       ENDIF
//IF4    IF (STEP01.RC = 0 & STEP02.RC <= 4) THEN
//STEP05     EXEC PROC=PRC1
//       ENDIF
//IF5    IF STEP05.PRC1.PST1.ABEND THEN
//STEP06     EXEC PGM=IDCAMS
//       ELSE
//STEP07     EXEC PGM=SORT
//       ENDIF
```

IF condition example

- The return code of STEP01 is tested in IF1. If it is 0, then STEP02 is executed. Else, the processing goes to the next IF statement (IF2).
- In IF2, If STEP01 has started execution, then STEP03a and STEP03b are executed.
- In IF3, If STEP03b does not ABEND, then STEP04 is executed. In ELSE, there are no statements. It is called a NULL ELSE statement.
- In IF4, if STEP01.RC = 0 and STEP02.RC <=4 are TRUE, then STEP05 is executed.
- In IF5, if the proc-step PST1 in PROC PRC1 in jobstep STEP05 ABEND, then STEP06 is executed. Else STEP07 is executed.
- If IF4 evaluates to false, then STEP05 is not executed. In that case, IF5 are not tested and the steps STEP06, STEP07 are not executed.

JCL - Restart Parameter

- to Restart a JOB from any particular step instead of starting it from the beginning
- This is very useful for the jobs with multiple steps and abends after executing few steps
- job steps were already got executed and they generated required output.
- So starting from beginning is not required and this may cause abnormal results as the process has run already. RESTART is the solution in this case where it starts the JOB.

JCL - Restart Parameter

- **Syntax:**

```
RESTART= stepname or stepname.procstepname
```

- **Example:**

```
//LINES JOB '1/17/95',RESTART=STEP2
//STEP1  EXEC .....
//  .....
//STEP2  EXEC .....
//  .....
```

- This JOB statement indicates that the system is to restart execution at the beginning of the job step named STEP2.

JCL - Restart Parameter

```
//MATEKSD JOB MSGLEVEL=(1,1),NOTIFY=&SYSUID,RESTART=STEP05
//*
//* EXAMPLE TO RESTART PARAMETERS
//*
//STEP01   EXEC PGM=PGM1
//STEP02   EXEC PGM=PGM2
//STEP03   EXEC PGM=PGM3
//STEP04   EXEC PGM=PGM4
//STEP05   EXEC PGM=PGM5
//STEP06   EXEC PGM=PGM6
//STEP07   EXEC PGM=PGM7
//STEP08   EXEC PGM=PGM8
//*
```

- the system is to restart execution at the beginning of the job step named STEP05. So the step before STEP04 don't execute.

JCL - Restart Parameter

```
//TESTCTLG JOB
100,CLASS=C,MSGCLASS=Y,NOTIFY=&SYSUID,RESTART=STEP1.STEP07
//*
//MYLIBS1  JCLLIB  ORDER=userid.PROCS.JCL
//*
//STEP1 EXEC CTLGPROC
//*

//CTLGPROC PROC
//*
//STEP01   EXEC PGM=PGM1
//STEP02   EXEC PGM=PGM2
//STEP03   EXEC PGM=PGM3
//STEP04   EXEC PGM=PGM4
//STEP05   EXEC PGM=PGM5
//STEP06   EXEC PGM=PGM6
//STEP07   EXEC PGM=PGM7
//STEP08   EXEC PGM=PGM8
//*
```

JCL - Procedures

- A procedure library member contains only part of the JCL for a given task-usually the fixed, unchanging part of JCL. The user of the procedure supplies the variable part of the JCL for a specific job. In other words, a JCL procedure is like a macro.
- Set of Job control statements that are frequently used are defined separately as a procedure and it can be invoked as many times as we need from the job. The use of procedures helps in minimizing duplication of code and probability of error.
- There are two types of procedures available in JCL,

1. In-stream procedures.
2. Cataloged procedures.

3. Syntax

```
//Step-name EXEC procedure name
```

Instream Procedure

- When the procedure is coded within the same JCL member, it is called an Instream Procedure.
- An in-stream procedure must begin with a PROC statement, end with a PEND statement, and include only the following other JCL statements: CNTL, comment, DD, ENDCNTL, EXEC, IF/THEN/ELSE/ENDIF, INCLUDE, OUTPUT JCL, and SET.
- do not use nested in-stream procedures.

Instream Procedure-example

```
//TESTJCL1 JOB 100,CLASS=C,MSGCLASS=Y,NOTIFY=&SYSUID
//*
//INSMPROC   PROC
//*START OF PROCEDURE
//PROC1           EXEC PGM=SORT
//SORTIN          DD DSN=HLQ.FILE.INPUT,DISP=SHR
//SORTOUT         DD DSN=HLQ.FILE.OUTPUT,DISP=SHR
//SYSOUT          DD SYSOUT=*
//SYSIN           DD DSN=HLQ.APP.CARDLIB(SORTCARD),DISP=SHR
//         PEND
 //*END OF PROCEDURE
//*
//STEP1     EXEC INSMPROC
//*
//STEP2     EXEC INSMPROC
```

You must place the in-stream procedure before the EXEC statement that calls it.

Cataloged Procedures:

- When the procedure is separated out from the JCL and coded in a different data store, it is called a Cataloged Procedure.
- A PROC statement is not mandatory to be coded in a cataloged procedure.
- The library containing cataloged procedures is a partitioned data set (PDS) or a partitioned data set extended (PDSE).

JCL - Cataloged Procedure

```
//TESTCTLG JOB 100,CLASS=C,MSGCLASS=Y,NOTIFY=&SYSUID
//*
//MYLIBS1  JCLLIB  ORDER=MYCOBOL.BASE.PROCLIB
//*
//STEP1 EXEC CTLGPROC
//*
```

Your JCL begins with a JOB statement that names your job TESTCTLG.

The first step in TESTCTLG, named STEP1, Identifies CTLGPROC as the JCL procedure to be run.

Here, the procedure CTLGPROC is present in MYCOBOL.BASE.PROCLIB library.

JCL - Cataloged Procedure

```
//CTLGPROC PROC
//*
//PROC1                EXEC PGM=SORT
//SORTIN         DD DSN=HLQ.FILE.INPUT,DISP=SHR
//SORTOUT        DD DSN=HLQ.FILE.OUTPUT,DISP=SHR
//SYSOUT         DD SYSOUT=*
//SYSIN          DD DSN=HLQ.APP.CARDLIB(SORTCARD),DISP=SHR
//*
```

- //MYLIBS1 JCLLIB ORDER=MYCOBOL.BASE.PROCLIB
- The JCLLIB statement allows you to code and use procedures
- You can code only one JCLLIB statement per job.
- Proc CTLGPROC is present in MYCOBOL.BASE.PROCLIB library.

The DD statement

- Datasets are mainframe files with records organized in a specific format. Datasets are stored on the Direct Access Storage Device (DASD) or Tapes
- The input and output resources required by a job step needs to be described within a DD statement with information such as the dataset organisation, storage requirements and record length.
- It describes the data sets that are used by the program. We can code one DD statement for each data set. Always code the DD statements after the EXEC statement that identifies the job step.

- DD statements are the most complicated of the JCL statements.

The DD statement

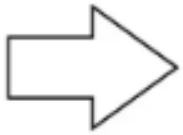
- Syntax
  - //DD-name DD Parameters

  A DD-NAME identifies the dataset or input/output resource.
- DD This is the keyword to identify it as an DD statement.

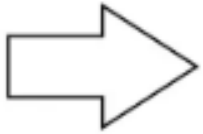
.

The DD statement

- ```
  //JOB1 JOB ACT1,NOTIFY=TRGXXX
  //STEP1 EXEC PGM=P1
  //DD1    DD ...
  //DD2    DD ...
  ....
  //STEP2 EXEC PGM=P2
  //INDD   DD ...
  //DD2    DD ...
  ....
  //
  ```

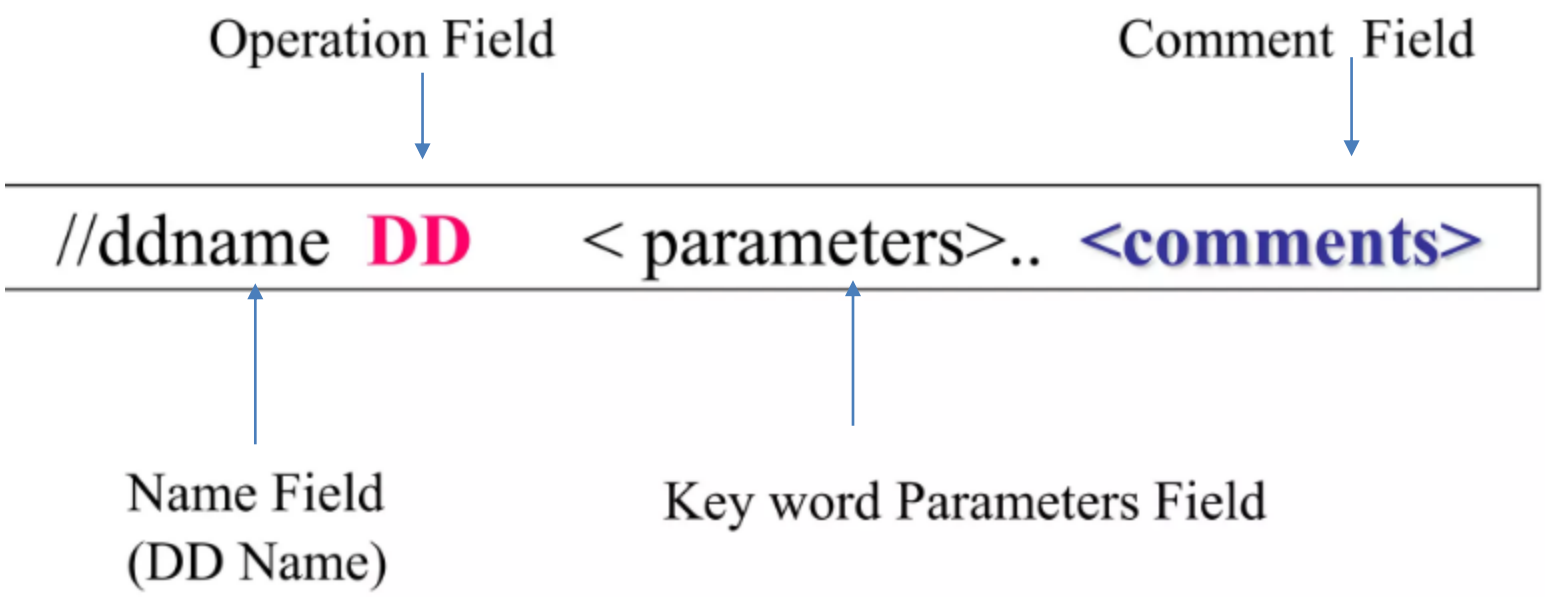


DD statements for step ***STEP1***



DD statements for step ***STEP2***
  ```

The DD statement



Keyward parameters for DD

```
//ddname  DD  DSNAME=dataset name
          ,DISP=file's status
        [     ,UNIT=device where the file exists    ]
        [     ,VOL=SER=serial number of the volume  ]
        [     ,SPACE=DASD space to be allocated     ]
        [     ,DCB=Options for file's data control block ]
```

- The name of the data set
- Simple Name : 1 to 8 chars
- Qualified Name : separated by periods each name 1 to 8 chars total 44 characters including periods

- Temporary data sets : &&TEMP or do not mention DSN parameter

DSN

- A dataset name specifies the name of a file and it is denoted by DSN in JCL. The DSN parameter refers to the physical dataset name of a newly created or existing dataset
- **DSN=&name | *.stepname.ddname**.
- **Temporary datasets** need storage only for the job duration and are deleted at job completion. Such datasets are represented as **DSN=&name** or simply without a DSN specified.
- If a temporary dataset created by a job step is to be used in the next job step, then it is referenced as **DSN=*.stepname.ddname**. This is called **Backward Referencing.**

DSN

```
//TTYYSAMP JOB 'TUTO',CLASS=6,MSGCLASS=X,REGION=8K,
 // NOTIFY=&SYSUID
//*
//STEP010 EXEC  PGM=ICETOOL,ADDRSPC=REAL
 //*
//INPUT1 DD DSN=TUTO.SORT.INPUT1,DISP=SHR
 //INPUT2 DD DSN=TUTO.SORT.INPUT2,DISP=SHR,UNIT=SYSDA,
 // VOL=SER=(1243,1244)
 //OUTPUT1 DD DSN=MYFILES.SAMPLE.OUTPUT1,DISP=(,CATLG,DELETE),
 // RECFM=FB,LRECL=80,SPACE=(CYL,(10,20))
//OUTPUT2 DD SYSOUT=*
```

Concatenating Datasets

- If there is more than one dataset of the same format, they can be concatenated and passed as an input to the program in a single DD name.

```
//CONCATEX JOB CLASS=6,NOTIFY=&SYSUID
 //*
 //STEP10 EXEC PGM=SORT
//SORTIN DD DSN=SAMPLE.INPUT1,DISP=SHR
//            DD DSN=SAMPLE.INPUT2,DISP=SHR
 //           DD DSN=SAMPLE.INPUT3,DISP=SHR
 //SORTOUT DD DSN=SAMPLE.OUTPUT,DISP=(,CATLG,DELETE),
//          LRECL=50,RECFM=FB
```

three datasets are concatenated and passed as input to the SORT program in the SORTIN DD name. The files are merged, sorted on the specified key fields and then written to a single output file SAMPLE.OUTPUT in the SORTOUT DD name.

Temporary dataset

Syntax:

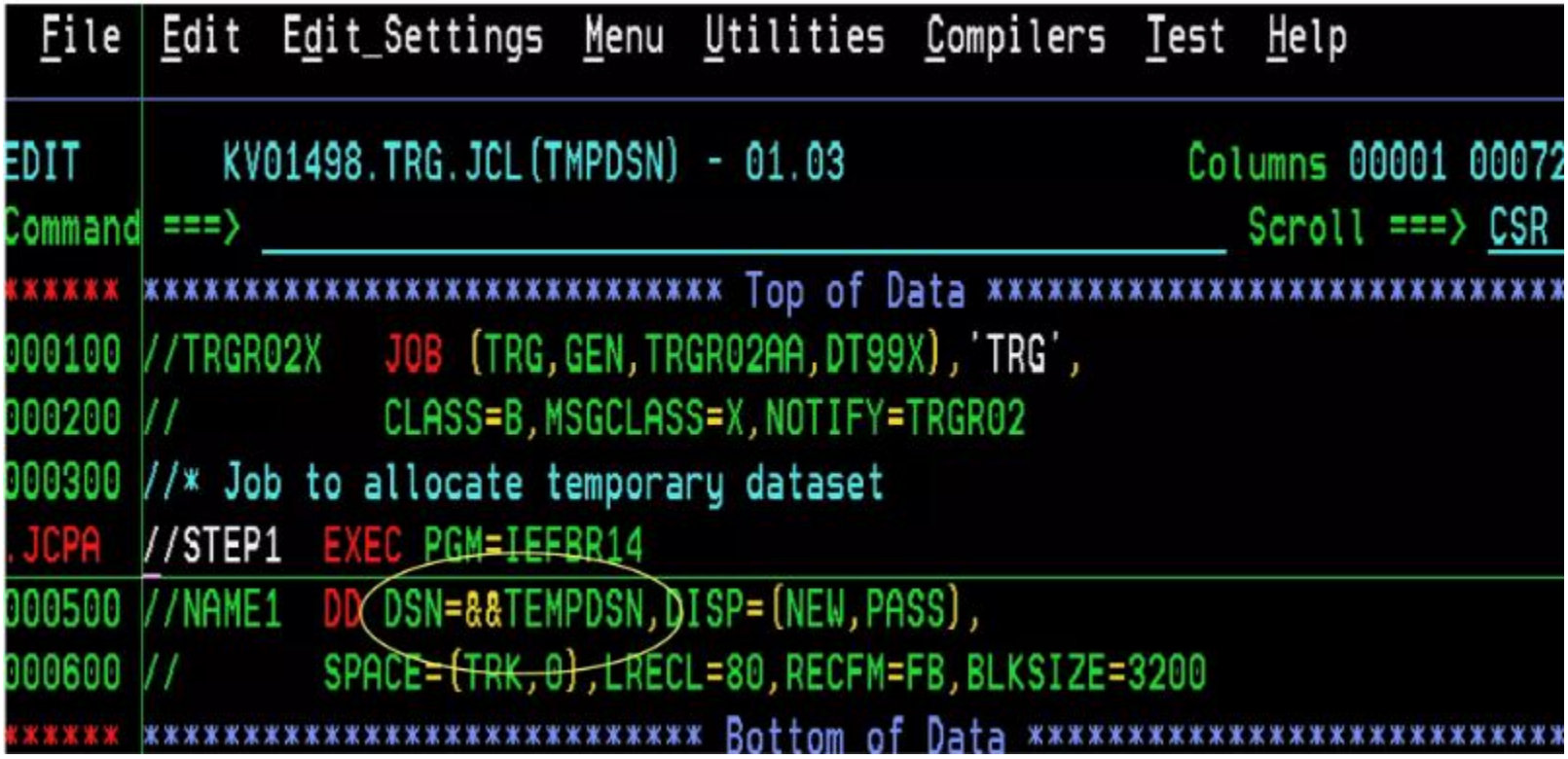
DSN=&&<*data set name*>

Dataset exists only during the execution of job. Usually called as Work files.

Always deleted at the end of job

Resides in Virtual Storage

Compound DSN, joined by period not allowed in temporary dataset


File Edit Edit_Settings Menu Utilities Compilers Test Help
EDIT KV01498.TRG.JCL(TMPDSN) - 01.03 Columns 00001 00072
Command ===> Scroll ===> CSR
****** ******************************** Top of Data ********************************
000100 //TRGR02X JOB (TRG,GEN,TRGR02AA,DT99X),'TRG',
000200 // CLASS=B,MSGCLASS=X,NOTIFY=TRGR02
000300 //* Job to allocate temporary dataset
.JCPA //STEP1 EXEC PGM=IEFBR14
000500 //NAME1 DD DSN=&&TEMPDSN,DISP=(NEW,PASS),
000600 // SPACE=(TRK,0),LRECL=80,RECFM=FB,BLKSIZE=3200
****** ******************************** Bottom of Data ********************************

Disposition parameter:DISP

The DISP parameter is used to describe the status of the dataset, disposition at the end of the job step on normal and abnormal completion.

Syntax:

DISP=<([*current status*][,*normal termination DISP*]
[,*abnormal termination DISP*])>

CURRENT STATUS	*NORMAL TERMINATION*	*ABNORMAL TERMINATION*
NEW	DELETE	DELETE
OLD	KEEP	KEEP
SHR	PASS	CATLG
MOD	CATLG	UNCATLG
	UNCATLG	

NEW : The dataset is newly created by the job step.

OLD : The dataset is already created and will be overwritten in the job step. The job step gains exclusive access on the dataset and no other job can access this dataset until the completion of the job step.

SHR : The dataset is already created and will be read in the job step. The dataset can be read by multiple jobs at the same time.

MOD : The dataset is already created. This disposition will be used when there is a need to append new records to the existing dataset (existing records will not be overwritten).

DISP-current status example

- Sharing the already existing data set.

  //INFILE DD DSN=TRGXX.SRC(MEM1),DISP=SHR

- Exclusive control over already existing data set or writing to existing data set. Existing records will be deleted.

  //OUTFILE DD DSN=TRGXX.OUT,DISP=OLD

- Creating new data set.

  //NEWDD DD DSN=TRGXX.NEW,DISP=NEW.....

- For updating the existing data set in exclusive access mode and is positioned at the end of the data, so the records may be added to the end.

  //DD1 DD DSN=TRGXX.TRN,DISP=MOD .....

Disposition parameter:DISP

- **CATLG** : The dataset is retained with a entry in the system catalog.
- **UNCATLG** : The dataset is retained but system catalog entry is removed.
- **KEEP** : The dataset is retained without changing any of the catalog entries. KEEP is the only valid disposition for VSAM files. This is to be used only for permanent datasets.
- **DELETE** : Dataset is deleted from user and system catalog.
- **PASS** : This is valid only for normal disposition. This is used when the dataset is to be passed and processed by the next job step in a JCL

- When any of the sub-parameters of DISP are not specified, the default values are as follows:
  - **status** : NEW is the default value.
  - **normal-disposition** : If status is NEW, default normal-disposition is DELETE, else it is KEEP.
  - **abnormal-disposition** : Same as normal disposition.

DISP-normal termination example

Request to keep the data set on normal completion of step.

```
//INFILE  DD  DSN=TRGXX.SRC,DISP=(SHR.KEEP)
```

To delete the data set on normal completion of step.

```
//OUTFILE DD DSN=TRGX.OUT,DISP=(OLD,DELETE)
```

To catalog the data set on normal termination of step.

```
//NEWDD  DD  DSN=TRGX.NEW,DISP=(NEW,CATLG)
```

To pass data set for use by subsequent step on normal termination of step.

```
//DD1  DD DSN=TRGXX.TRN,DISP=(MOD,PASS)
```

DISP-abnormal termination example

To Keep the data set on abnormal completion of step.

```
//FILE1  DD  DSN=TRGX.SRC,DISP=(SHR,PASS,KEEP)
```

To Uncatalog the data set on abnormal completion of step.

```
//OUT1 DD DSN=TRGX.OUT,DISP=(OLD,,UNCATLG)
```

To Catalog the data set on abnormal termination of step.

```
//NEWDD  DD  DSN=TRGX.NEW,DISP=(NEW,,CATLG)
```

To Delete data set on abnormal termination of step.

```
//DD1  DD DSN=TRGXX.TRN,DISP=(,PASS,DELETE)
```

DISP parameter -default

DISP coded on DD statement	*System Interpretation by default*
NO DISP PARAMETER	DISP=(NEW,DELETE,DELETE)
DISP=SHR	DISP=(SHR,KEEP,KEEP)
DISP=OLD	DISP=(OLD,KEEP,KEEP)
DISP=(,CATLG)	DISP=(NEW,CATLG,CATLG)
DISP=(OLD,,DELETE)	DISP=(OLD,KEEP,DELETE)
DISP=MOD (Treated as new)	DISP=(MOD,DELETE,DELETE)
DISP=MOD (Treated as old)	DISP=(MOD,KEEP,KEEP)

Input-Output Methods

Any batch program executed through a JCL requires input data, which is processed and output will be created. There are different ways of passing input data to the program and writing into output.
In batch mode, there is no user interaction required but input and output dataset or devices and required organisation are defined in JCL.

Input-Output Methods

- **INSTREAM DATA:**
- Instream data(also called inline data sets) to a program can be specified using a SYSIN DD statement. SYSIN statement used to start an instream data set.
- Data can be accepted in the program through ACCEPT statement.
- One row in the SYSIN will be equal to one ACCEPT in the program.
- In-stream data sets begin with a DD * or DD DATA statement.
- /* known as delimiter and by using /* we can end the instream data.Delimiter (/*) is mandatory with SYSIN. Delimiter (/*) always starts at column1 and ends at column2.
-

INSTREAM DATA

```
//JOBIBMKS  JOB (123),'IBMMAINFRAMER',CLASS=C,MSGCLASS=S,MSGLEVEL=(1,1),
//       NOTIFY=&SYSUID
//*
//STEP01 EXEC PGM=MYPROG
//IN1    DD DSN=userid.TEST.INPUT1,DISP=SHR
//OUT1   DD DSN=userid.TEST.OUTPUT1,DISP=(,CATLG,DELETE),
//       LRECL=50,RECFM=FB
//SYSIN  DD *
KALAI  1000
SRINI  2000
/*
```

- input to MYPROG is passed through SYSIN.
- Two records of data are passed to the program.
- "KALAI 1000" is record1 and "SRINI 2000" is record2.
- End of data condition is met when the symbol /* is encountered while reading the data.

INSTREAM DATA

```
//JOBIBMKS  JOB
(123),'IBMMAINFRAMER',CLASS=C,MSGCLASS=S,MSGLEVEL=(1,1),
//       NOTIFY=&SYSUID
//*
//STEP01 EXEC PGM=MYPROG
//OUT1   DD DSN=userid.TEST.OUTPUT2,DISP=(,CATLG,DELETE),
//       LRECL=50,RECFM=FB
//SYSIN  DD DSN=userid.SYSIN.DATA,DISP=SHR
```

In this example, the SYSIN data is held within a dataset, where userid.SYSIN.DATA is a PS file, which can hold one or more records of data.

Data Output in a JCL

```
//JOBIBMKS  JOB
(123),'IBMMAINFRAMER',CLASS=C,MSGCLASS=Y,MSGLEVEL=(1,1),
//      NOTIFY=&SYSUID
//STEP1    EXEC PGM=MYPROG
//IN1      DD DSN=userid.IBMMF.INPUT,DISP=SHR
//OUT1     DD SYSOUT=*
//SYSOUT   DD DSN=userid.IBMMF.SYSOUT,DISP=SHR
//SYSPRINT DD DSN=userid.IBMMF.SYSPRINT,DISP=SHR
//*
```

Job logs can also be saved into a dataset by mentioning an already created dataset for SYSOUT and SYSPRINT

JCL - Utility Programs

- IEFBR14- The utility program IEFBR14 performs no action other than return a completion code of 0; Dummy program often used for dataset operations.
- IDCAMS- Used for dataset management, such as cataloging, deleting, and renaming.
  - IEBCOPY- IEBCOPY is a data set utility that is used to copy or merge members between one or more partitioned data sets, or partitioned data sets extended (PDSEs), in full or in part.
  - IEBDG- Create a test data set consisting of patterned data
  - IEBGENER- It is used to copy or print sequential data sets. Also used to copy any TAPE file to DISK or DISK to TAPE.
  - IEBUPDTE- The IEBUPDTE utility creates multiple members in a partitioned data set, or updates records within a member. While it can be used for other types of records, its main use is to create or maintain JCL procedure libraries or assembler macro libraries.
  - IEBCOMPR – It is used to compare the contents of two PS or PDS datasets.

IEFBR14

- The IEFBR14 program is nothing more than a null program. Its name is derived from an assemble language instruction that is used to exit a procedure or program. and this is exactly what is does. It executed a single statment which specifies the end of the program.
- The utility program IEFBR14 performs no action other than return a completion code of 0;

1. Allocate/create datasets
2. Delete datasets
3. Uncatlog Datasets
4. Catalog Datasets
5. Setting return code to zero

- mostly use IEFBR14 utility to create or delete the PS(partition dataset) file.

IEFBR14

Example 1: Allocate/Create empty PS dataset.

```
//JOBIBMKS  JOB
    (123),'IBMMAINFRAMER',CLASS=C,MSGCLASS=S,MSGLEVEL=(
    1,1),
//      NOTIFY=&SYSUID
//*
//STEP001  EXEC PGM=IEFBR14
//DD01     DD DSN=userid.TEST.PSFILE,
//          DISP=(NEW,CATLG),
//          SPACE=(TRK,(12,33),RLSE),UNIT=SYSDA,
//          DCB=(DSORG=PS,RECFM=FB,LRECL=80,BLKSIZE=800)
//*
```

- **Example 2:** Delete PS dataset.

```
//JOBIBMKS JOB
(123),'IBMMAINFRAMER',CLASS=C,MSGCLASS=S,MSGLEVEL=(1,1),
//       NOTIFY=&SYSUID
//*
//STEP001  EXEC PGM=IEFBR14
//SYSPRINT DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//SYSDUMP  DD SYSOUT=*
//DD01     DD DSN=userid.TEST.PSFILE,
//            DISP=(OLD,DELETE,DELETE)
//*
```

IEBGENER

```
//JOBIBMKS JOB
(123),'IBMMAINFRAMER',CLASS=C,MSGCLASS=S,MSGLEVEL=
(1,1),
//      NOTIFY=&SYSUID
//STEP001  EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=A
//SYSUT1   DD DSN=userid.FILE1.INPUT,DISP=SHR
//SYSUT2   DD DSN=userid.FILE2.OUTPUT,
//       DISP=(NEW,CATLG,DELETE),UNIT=DISK1,
//       SPACE=(TRK,20,10),RLSE),
//SYSIN    DD DUMMY
//
```

JCL - IEBCOPY Utility

```
//JOBIBMKS JOB
(123),'IBMMAINFRAMER',CLASS=C,MSGCLASS=S,MSGLEVEL=(1,1),
//      NOTIFY=&SYSUID
//*
//STEP001  EXEC  PGM=IEBCOPY
//SYSPRINT DD  SYSOUT=A
//SYSOUT1  DD  DSNAME=userid.TEST.DATA.IN,
//           DISP=SHR,UNIT=DISK,
//           VOL=SER=1234
//SYSOUT2  DD  DSNAME=userid.TEST.DATA.OUT,
//           DISP=(NEW,KEEP),
//           SPACE=(TRK,(10,22,40),RLSE),
//           UNIT=DISK,VOL=SER=1234,
//SYSIN    DD DUMMY
/*
```