JCL
job control language

tutorialspoint

SIMPLYEASYLEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia

https://twitter.com/tutorialspoint

About the Tutorial

Job Control Language (JCL) is the command language of Multiple Virtual Storage (MVS), which is the commonly used Operating System in the IBM Mainframe computers. JCL identifies the program to be executed, the inputs that are required and the location of the input/output, and informs the Operating System through Job control Statements.

In mainframe environment, programs can be executed in batch and online modes. JCL is used for submitting a program for execution in batch mode.

Audience

This tutorial will be useful for software programmers who would like to understand the basics of Job Control Language. Also, this tutorial will be helpful to mainframe professionals in increasing their level of expertise in JCL.

Prerequisites

The tutorial is intended for readers who have a basic understanding of job management and data management in mainframe environment.

Disclaimer & Copyright

Table of Contents

1. JCL – OVERVIEW

JCL is used in mainframe environment to act as a bridge between a program (Example: COBOL, Assembler or PL/I) and the operating system. In a mainframe environment, programs can be executed in both **batch mode** as well as **online mode**.

In **batch mode**, programs are submitted to the operating system as a job through a JCL. For example, processing bank transactions through a VSAM (Virtual Storage Access Method) file and applying it to the corresponding accounts is a batch system. In contrast, a bank staff opening an account using a back office screen is an example of an **online system**.

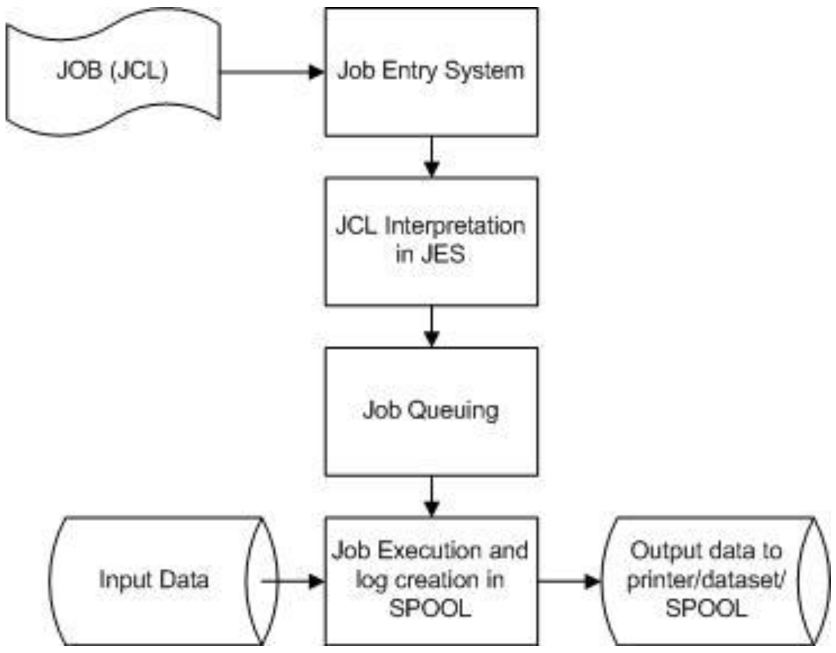
Batch and online processing differ in the way they receive their inputs and program execution requests. In batch processing, these parameters are fed into the JCL which is in turn received by the Operating System.

Job Processing

A job is a unit of work which can be made up of many job steps. Each job step is specified in the Job Control Language (JCL) through a set of **Job Control Statements**.

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.

Job processing goes through a series of steps as given below:


JOB (JCL)
Job Entry System
JCL Interpretation in JES
Job Queuing
Input Data
Job Execution and log creation in SPOOL
Output data to printer/dataset/ SPOOL

- **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.

2. JCL – ENVIRONMENT SETUP

Installing JCL on Windows/Linux

There are many Free Mainframe Emulators available for Windows which can be used to write and learn sample JCLs.

One such emulator is **Hercules**, which can be easily installed in Windows by following a few simple steps as given below:

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

- Once you install the package on a Windows machine, it will create a folder like **C:\Mainframes**.

- Run Command Prompt (CMD) and go to the directory C:\Mainframes on CMD.

- The complete guide on various commands to write and execute a JCL can be found at **www.jaymoseley.com/hercules/installmvs/instmvs2.htm**

Hercules is an open source software implementation of the mainframe System/370 and ESA/390 architectures, in addition to the latest 64-bit z/Architecture. Hercules runs under Linux, Windows, Solaris, FreeBSD, and Mac OS X.

Running JCL on Mainframes

A user can connect to a mainframe server in a number of ways such as a thin client, a dummy terminal, a Virtual Client System (VCS), or a Virtual Desktop System (VDS).

Every valid user is given a login id to enter into the Z/OS interface (TSO/E or ISPF). In the Z/OS interface, the JCL can be coded and stored as a member in a Partitioned Dataset (PDS). When the JCL is submitted, it is executed and the output is received as explained in the job processing section of the previous chapter.

Structure of a JCL

The basic structure of a JCL with the common statements is given below:

```
//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)
```

Program Description

The numbered JCL statements are explained below:

(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.

(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 the above example.

(7) /* marks the end of instream data.

All the JCL statements except instream data start with //. There should be at least one space before and after JOB, EXEC, and DD keywords and there should not be any spaces in the rest of the statement.

JOB Parameter Types

Each of the JCL statements is accompanied by a set of parameters to help the Operating System in completing the program execution. The parameters can be of two types:

Positional Parameters

- Appears at predefined position and order in the statement. Example: Accounting information Parameter can appear only after the **JOB** keyword and before the programmer name parameter and the Keyword Parameters. If a positional parameter is omitted, it has to be replaced with a comma.

- Positional Parameters are present in JOB and EXEC statements. In the above example, PGM is a positional parameter coded after the **EXEC** keyword.

Keyword Parameters

- Keyword Parameters 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*. Example: MSGCLASS=X, i.e., the job log is redirected to the output SPOOL after the job completion.

- In the above example, CLASS, MSGCLASS, and NOTIFY are keyword parameters of JOB statement. There can be keyword parameters in EXEC statement as well.

These parameters have been detailed out in the subsequent chapters along with appropriate examples.

3. JCL – JOB STATEMENT

JOB Statement is the first control statement in a JCL. This gives the identity of the job to the Operating System (OS), in the spool and in the scheduler. The parameters in the JOB statement help the OS in allocating the right scheduler, required CPU time, and issuing notifications to the user.

Syntax

Following is the basic syntax of a JCL JOB statement:

```
   //Job-name JOB Positional-param, Keyword-param
```

Description

Let us take a closer look at the terms used in the above JOB statement syntax.

Job-name

It gives an id to the job while submitting it to the OS. It can be of the length of 1 to 8 with alphanumeric characters and starts just after //.

JOB

This is the keyword to identify it as a JOB statement.

Positional-param

Positional parameters can be of two types:

Positional Parameter	Description
Account information	It 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	It identifies the person or group who is in charge of the JCL. It is not a mandatory parameter and can be replaced by a comma.

Keyword-param

Following are the various keyword parameters, which can be used in a JOB statement. You can use one or more parameters (separated by comma) based on your requirements.

Keyword Parameter	Description
CLASS	Based on the time duration and the number of resources required by the job, companies assign different job classes. These can be visualized as individual schedulers used by the OS to receive the jobs. Placing the jobs in the right scheduler will aid in easy execution of the jobs. Some companies have different classes for jobs in test and production environment. 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. 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 the 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.
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.
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 a job, "A" can be typed against the job in the SPOOL, which will bring the job to execution.
TIME	Specifies the timespan 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 This parameter can be useful while testing a newly coded program. In order to ensure that the program does not run for a long time due to looping errors, a time parameter can be coded so that the program aborts when the specified CPU time is reached.
REGION	Specifies the address space required to run a job step within the job. Following is the syntax: **REGION=nK | nM** Here, *region* can be specified as nK or nM where n is a number, K is kilobyte, and M is Megabyte. When REGION = 0K or 0M, largest address space is provided for execution. In critical applications, coding of 0K or 0M is prohibited to avoid wasting the address space.

Example

```
//URMISAMP JOB (*),"tutpoint",CLASS=6,PRTY=10,NOTIFY=&SYSUID,

//   MSGCLASS=X,MSGLEVEL=(1,1),TYPRUN=SCAN,

//   TIME=(3,0),REGION=10K
```

Here, the 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.

Miscellaneous Parameters

There are other parameters which can be used with a JOB Statement, but they are not frequently used:

ADDRSPC	Type of storage used: Virtual or Real
BYTES	Size of data to be written to output log and the action to be taken when the size is exceeded.
LINES	Maximum number of lines to be printed to output log.
PAGES	Maximum number of pages to be printed to output log.