JCL Note

- JCL(Job Control Language)

JCL stands for Job Control Language. It is used on IBM mainframe systems to control the execution of programs and to manage system resources. Here are some key points about JCL:

1. Purpose: JCL is primarily used to instruct the system on how to run a batch job or start a subsystem. It specifies which programs to execute, which files to use, and what resources are needed.

2. Components:

- Job Statement (JOB): Defines the start of a job and provides job-level parameters such as job name, accounting information, and priority.

- Exec Statement (EXEC): Specifies the program to be executed within the job and provides parameters for program execution.

- DD Statement (Data Definition): Describes the data sets (files) and devices needed by the job or program. It specifies the input and output files, their locations, and attributes.

3. Syntax:

- JCL statements are composed of fixed-length records, typically 80 characters.

- Statements begin with // in columns 1 and 2.

- Parameters and keywords follow specific formats and positions.

4. Example:

```
//JOBNAME  JOB  (ACCT#),'JOB DESCRIPTION',CLASS=A,MSGCLASS=A
//STEP1    EXEC PGM=MYPROG
//INPUT1   DD   DSN=MY.INPUT.FILE,DISP=SHR
//OUTPUT1  DD   DSN=MY.OUTPUT.FILE,DISP=(NEW,CATLG,DELETE),
//          UNIT=SYSDA,SPACE=(CYL,(10,5),RLSE)
```

5. Functions:

- Resource Allocation: Allocates system resources such as memory, CPU time, and disk space.

- Job Scheduling: Specifies job dependencies, priorities, and conditions for job execution.

- Error Handling: Defines actions to be taken in case of errors during job execution.

6. Importance:

- JCL is critical for batch processing environments, where large volumes of data are processed without user interaction.

- It ensures efficient use of system resources and helps maintain system performance and reliability.

- JOB, EXEC and DD statements

JOB Statement

The JOB statement marks the beginning of a job and provides job-level control information to the operating system.

Syntax

//JOBNAME JOB (accounting_info),'programmer_name',CLASS=class,MSGCLASS=msgclass

Parameters

- JOBNAME: A name for the job, which must be unique within a system.

- accounting_info: Account number or other information used for billing or tracking resource usage.

- programmer_name: A comment field often used to identify the person or team responsible for the job.

- CLASS: Determines the priority and scheduling of the job. It defines job categories based on resources required or service levels.

- MSGCLASS: Specifies the output class for job-related messages. Each class routes messages to different locations (e.g., console, printer, or dataset).

Example

//PAYROLL JOB (1234),'J DOE',CLASS=A,MSGCLASS=X

EXEC Statement

The EXEC statement specifies the program or procedure to be executed in a job step.

Syntax

//STEPname EXEC PGM=program_name,PARM=parameters

//STEPname EXEC PROC=procedure_name

Parameters

- STEPname: A name for the step, used for identification and reference within the job.

- PGM: Specifies the program to be executed.

- PARM: Passes parameters to the program.

- PROC: Specifies a procedure (a set of JCL statements) to be executed. A procedure can contain multiple EXEC and DD statements.

Example

```
//STEP1  EXEC  PGM=MYPROG
```

or

```
//STEP2  EXEC  PROC=MYPROC
```

DD Statement

The DD (Data Definition) statement describes the datasets and devices used in a job step. It defines the input and output data for programs.

Syntax

```
//DDname  DD  parameters
```

Parameters

- DDname: A name for the dataset, which must match the dataset name expected by the program.

- DSN: Specifies the dataset name.

- DISP: Defines the dataset's status and disposition (e.g., NEW, OLD, SHR).

- UNIT: Specifies the device type (e.g., disk, tape).

- SPACE: Defines the amount of space required for the dataset.

- VOL: Identifies the volume where the dataset resides.

Example

```
//INPUT1  DD  DSN=MY.INPUT.FILE,DISP=SHR
//OUTPUT1 DD  DSN=MY.OUTPUT.FILE,DISP=(NEW,CATLG,DELETE),
//           UNIT=SYSDA,SPACE=(CYL,(10,5),RLSE)
```

Note:

EXEC Statements: Up to 255 per job.
DD Statements: Typically up to 3273 per job, and up to 3273 per step.

- Different types of parameters in JCL

=> Positional Parameters

- The JOB card parameters that are very specific about their position are called as positional parameters.
- These parameters are mandatory and required for OS.
- Positional parameters start immediately after the JOB keyword with a space gap.
- The order is very important for positional parameters. If we try to change the order of them, the JCL error occurs.

There are two positional parameters in the JOB card -

1. Accounting information.
2. Programmer name.

=> Keyword Parameters

- All Keyword parameters are optional and can code in any order.
- Keyword parameters follow the positional parameter programmer name.
- A comma separates each keyword parameter (,).
- Keyword parameter information applies to all steps.
- The total keyword parameters are more than 20.

➢ Detailed keyword parameters used on different levels in JCL

In JCL, parameters are used to control various aspects of job execution, resource allocation, and dataset handling. Here are some of the key parameters found in JOB, EXEC, and DD statements:

JOB Statement Parameters

1. CLASS: Specifies the job class, which determines the job's priority and resource allocation.

CLASS=A

2. MSGCLASS: Defines the output class for job-related messages.

MSGCLASS=X

3. MSGLEVEL: Controls the level of detail for messages written to the job log.

MSGLEVEL=(1,1)

4. TIME: Sets a maximum amount of CPU time the job can use.

   TIME=1440

5. REGION: Specifies the maximum amount of storage a job can use.

   REGION=4M

6. NOTIFY: Indicates who should be notified upon job completion.

   NOTIFY=&SYSUID

7. USER and PASSWORD: Used for security purposes to identify the user and provide a password.

   USER=USERID,PASSWORD=PASSWD

EXEC Statement Parameters

1. PGM: Specifies the program to be executed.

   PGM=MYPROG

2. PROC: Indicates a procedure to be executed.

   PROC=MYPROC

3. PARM: Passes parameters to the program.

   PARM='parameter_list'

4. TIME: Limits the CPU time for the specific step.

   TIME=(,10)

5. REGION: Limits the storage for the specific step.

   REGION=2M

6. COND: Specifies conditions under which a job step should be executed or bypassed.

   COND=(4,LT)

7. ADDRSPC: Defines the address space in which the step should run.

   ADDRSPC=REAL

DD Statement Parameters

1. DSN: Specifies the dataset name.

DSN=MY.DATASET.NAME

2. DISP: Defines the status and disposition of the dataset.

DISP=(NEW,CATLG,DELETE)

3. UNIT: Specifies the type of device.

UNIT=SYSDA

4. SPACE: Defines the amount of space required.

SPACE=(CYL,(10,5),RLSE)

5. VOL: Indicates the volume serial number.

VOL=SER=VOLSER

6. DCB: Specifies data control block parameters for the dataset.

DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)

7. SYSOUT: Directs output to a specified output class.

SYSOUT=A

8. DDNAME: Provides an alternative dataset name for temporary datasets.

DDNAME=TEMPDSN

9. DATACLAS, MGMTCLAS, STORCLAS: Specifies the data, management, and storage classes for SMS-managed datasets.

DATACLAS=DC1

MGMTCLAS=MC1

STORCLAS=SC1

10. PATH and PATHDISP: Used for UNIX System Services files.

PATH='/u/mydir/myfile',PATHDISP=(KEEP,DELETE)

- CLASS parameter

The CLASS parameter in JCL is used to categorize and prioritize jobs and job steps based on their resource requirements and scheduling preferences. It is specified in the JOB statement and can also be used in the EXEC statement for steps within a job. The actual implementation and characteristics of each class are defined by the system administrator and can vary between installations.

CLASS Parameter in JOB Statement

The CLASS parameter in the JOB statement defines the job's overall priority and determines which execution queue the job will be placed in. Different classes may have different resource limits, priorities, or execution windows.

Syntax

CLASS=class

Example

//MYJOB JOB (1234),'MY JOB',CLASS=A,MSGCLASS=X

CLASS Parameter in EXEC Statement

In the EXEC statement, the CLASS parameter specifies the class for a particular job step, which can override the job-level class setting for that step.

Syntax

CLASS=class

Example

//STEP1 EXEC PGM=MYPROG,CLASS=B

Understanding Class Definitions

The specific characteristics of each class (A, B, C, etc.) are defined by the system administrator. They might include:

- Priority: Determines the job's position in the execution queue relative to other jobs.

- Resource Limits: Limits on CPU time, memory usage, and other resources.

- Execution Windows: Time periods when jobs of a certain class are allowed to run.

- Routing: Directs output to specific devices or locations based on the class.

Class Configuration

Here’s an example of how classes might be configured on a specific mainframe system:

- CLASS=A: Standard batch jobs with normal priority, moderate resource limits.

- CLASS=B: High-priority jobs with higher resource limits, suitable for critical tasks.

- CLASS=C: Low-priority jobs, possibly background tasks with minimal resource allocation.

- CLASS=SYS: System jobs with highest priority, reserved for system maintenance and operations.

- MSGCLASS Parameter

The MSGCLASS parameter in JCL (Job Control Language) is used to specify the output class for job-related messages. These messages can include system messages, output from the job, and job logs. The MSGCLASS parameter is crucial for directing these messages to the appropriate destinations, such as system printers, spool datasets, or other output devices.

Importance of MSGCLASS

1. Job Monitoring: The MSGCLASS parameter helps in monitoring jobs by directing job logs and system messages to specific output locations where they can be reviewed. This is essential for debugging, performance monitoring, and auditing purposes.

2. Output Management: By using different message classes, administrators can manage and organize job outputs more efficiently. For instance, high-priority job outputs can be routed to specific printers or storage areas.

3. Resource Optimization: Proper use of MSGCLASS can help in optimizing system resources by segregating high-volume job logs from regular system output, ensuring that important logs are not delayed by less critical messages.

4. Security and Access Control: Different MSGCLASS values can be used to control access to job logs. Sensitive jobs can have their output directed to secure locations, ensuring that only authorized personnel can access them.

Syntax

The MSGCLASS parameter is specified in the JOB statement.

```
//jobname JOB  (accounting_info),'programmer_name',CLASS=class,MSGCLASS=msgclass
```

Example

```
//MYJOB  JOB  (1234),'MY JOB',CLASS=A,MSGCLASS=X
```

In this example, the job MYJOB is assigned to message class X.

Usage and Configuration

The actual implementation and characteristics of each message class (A, B, C, etc.) are defined by the system administrator and can vary between installations. Common configurations might include:

- MSGCLASS=A: Output directed to a general-purpose spool dataset.

- MSGCLASS=X: Output directed to a specific printer or high-priority spool dataset.

- MSGCLASS=H: Output directed to a long-term storage spool dataset for historical archiving.

- MSGCLASS=J: Output directed to a job monitoring system or console.

Example of Job Using MSGCLASS

```
//PAYROLL  JOB  (1234),'PAYROLL PROCESSING',CLASS=A,MSGCLASS=H,MSGLEVEL=(1,1)
//STEP1    EXEC PGM=PAYROLL
//INPUT    DD  DSN=PAYROLL.DATA,DISP=SHR
//OUTPUT   DD  DSN=PAYROLL.REPORT,DISP=(NEW,CATLG,DELETE),
//           UNIT=SYSDA,SPACE=(CYL,(10,5),RLSE)
```

In this example:

- The job PAYROLL is assigned to message class H.

- All system messages and job logs generated by this job will be directed to the output destination associated with message class H.

Detailed Breakdown

1. Job Submission and Execution:

  - When the job is submitted, the system reads the MSGCLASS parameter to determine where to route the job's output messages.

  - The job's steps are executed as specified in the EXEC and DD statements.

2. Message Routing:

  - System messages such as job start, end, and step completion messages are routed to the output destination defined by the MSGCLASS parameter.

  - Any output generated by the job, including program output and error messages, is also directed to the same destination.

3. Review and Analysis:

  - After job completion, administrators and users can review the job logs and system messages in the specified output location.

  - This is crucial for diagnosing issues, verifying job success, and analyzing performance.

Custom Configuration

System administrators can customize message classes based on organizational requirements. This includes defining specific classes for different departments, prioritizing critical jobs, or segregating output types for better management.

Example Scenario with Multiple MSGCLASS Values

```
//JOB1  JOB  (1111),'DAILY REPORT',CLASS=A,MSGCLASS=A
//STEP1 EXEC PGM=REPORTGEN
//INPUT DD  DSN=DAILY.DATA,DISP=SHR
//OUTPUT DD  DSN=DAILY.REPORT,DISP=(NEW,CATLG,DELETE),
//          UNIT=SYSDA,SPACE=(CYL,(5,2),RLSE)

//JOB2  JOB  (2222),'MONTHLY ANALYSIS',CLASS=B,MSGCLASS=H
//STEP1 EXEC PGM=ANALYSIS
//INPUT DD  DSN=MONTHLY.DATA,DISP=SHR
//OUTPUT DD  DSN=MONTHLY.REPORT,DISP=(NEW,CATLG,DELETE),
//          UNIT=SYSDA,SPACE=(CYL,(10,5),RLSE)
```

In this scenario:

- JOB1 uses MSGCLASS=A, directing its messages to a general-purpose output location.

- JOB2 uses MSGCLASS=H, directing its messages to a high-priority or long-term storage location.

Using the MSGCLASS parameter effectively allows for better management and organization of job outputs, enhancing the overall efficiency and reliability of job execution and monitoring on an IBM mainframe system.

- MSGLEVEL Parameter

The MSGLEVEL parameter in JCL is used to control the level of detail included in the job's output listing, also known as the job log or JES (Job Entry Subsystem) log. It determines the type and amount of information recorded about the job's execution. Proper use of MSGLEVEL can aid in debugging, auditing, and resource management.

Syntax

The MSGLEVEL parameter is specified in the JOB statement and has the following syntax:

MSGLEVEL=(statements, messages)

- statements: Controls the listing of JCL statements.

  - 0: Only JOB Statements (JOB statement, all comments before EXEC statement).

  - 1: All Statements (JCL statements, JES2 or JES3 control statements, the procedure statements, and symbolic parameter-related statements).

  - 2: Only JCL statements (JCL statements and JES2/JES3 control statements).

- messages: Controls the listing of messages.

  - 0: Print allocation and termination messages only when JOB ended abnormally

  - 1: Print allocation and termination when JOB ended either normally or abnormally

MSGLEVEL Parameter	Description
MSGLEVEL=(0,0)	Only JOB Statements, its allocation and termination messages when JOB ended abnormally (unsuccessfully).
MSGLEVEL=(0,1)	Only JOB Statements, its allocation and termination messages when JOB ended either normally or abnormally.
MSGLEVEL=(1,0)	All Statements, its allocation and termination messages when JOB ended abnormally (Unsuccessfully).
MSGLEVEL=(1,1)	All Statements, its allocation and termination messages when JOB ended either normally or abnormally. System Default.
MSGLEVEL=(2,0)	Only JCL statements, its allocation and termination messages when JOB ended abnormally (unsuccessfully).
MSGLEVEL=(2,1)	Only JCL statements, its allocation and termination messages when JOB ended either normally or abnormally.

Importance of MSGLEVEL

1. Debugging: By controlling the level of detail in the job log, you can include enough information to diagnose errors and issues in job execution without overwhelming the log with unnecessary details.

2. Auditing: Detailed logs can be crucial for auditing purposes, ensuring that all relevant events and actions are recorded.

3. Performance Monitoring: Monitoring job performance and resource usage can be easier with detailed logs that include allocation and control messages.

4. Resource Management: By adjusting the level of detail, you can optimize the amount of data stored and processed, thereby managing system resources more effectively.

By adjusting the MSGLEVEL parameter, users can control the verbosity of job logs, making it an essential tool for effective job management in a mainframe environment.

- PRTY Parameter

The PRTY (Priority) parameter in JCL is used to specify the priority of a job within its assigned class. It helps the Job Entry Subsystem (JES) determine the order in which jobs should be selected for execution from the job queue. Higher priority jobs are selected before lower priority ones within the same class.

Syntax

The PRTY parameter is specified in the JOB statement.

PRTY=priority

- priority: A numerical value, typically in the range from 0 to 15, where 0 is the lowest priority and 15 is the highest priority. The exact range can vary depending on the system configuration.

Example

```
//MYJOB  JOB  (1234),'MY JOB',CLASS=A,MSGCLASS=X,PRTY=10
```

In this example, the job MYJOB is assigned a priority of 10 within class A.

Importance of PRTY

1. Job Scheduling: The PRTY parameter helps in managing the job scheduling process by indicating which jobs should be executed first within their class. This can be crucial for ensuring that high-priority jobs are processed promptly.

2. Resource Management: Assigning priorities helps in optimizing resource usage by allowing critical jobs to access resources ahead of less important jobs.

3. Performance Optimization: High-priority jobs that require faster turnaround times can be prioritized to improve overall system performance and meet service level agreements (SLAs).

4. Operational Efficiency: By effectively using the PRTY parameter, system administrators can balance the workload and ensure that important jobs are not delayed by less critical ones.

Consider a system with the following jobs submitted:

```
//JOB1  JOB  (1234),'JOB ONE',CLASS=A,MSGCLASS=X,PRTY=5
//JOB2  JOB  (5678),'JOB TWO',CLASS=A,MSGCLASS=Y,PRTY=10
//JOB3  JOB  (9101),'JOB THREE',CLASS=A,MSGCLASS=Z,PRTY=1
```

- JOB2 has the highest priority (10) and will be executed first.
- JOB1 has a moderate priority (5) and will be executed next.
- JOB3 has the lowest priority (1) and will be executed last.

Summary:

The PRTY parameter is a powerful tool for managing job execution priorities in a mainframe environment. By assigning appropriate priority values, system administrators can ensure that critical jobs are processed efficiently, resource utilization is optimized, and overall system performance is maintained.

- REGION Parameter

The REGION parameter in JCL is used to specify the amount of storage (memory) a job or job step can use during execution. It is an important parameter because it helps manage the allocation of system resources, ensuring that jobs have sufficient memory to run efficiently without causing contention for resources.

Syntax

The REGION parameter can be specified in both the JOB statement and the EXEC statement.

In the JOB statement:

REGION=amount

In the EXEC statement:

REGION=amount

- amount: Specifies the amount of memory to allocate. This can be in kilobytes (K), megabytes (M), or in bytes (if no unit is specified). The value can also be 0K or 0M, which means the system's default maximum value is used.

Example

In the JOB statement:

```
//MYJOB  JOB  (1234),'MY JOB',CLASS=A,MSGCLASS=X,REGION=4M
```

In the EXEC statement:

```
//STEP1  EXEC PGM=MYPROG,REGION=2M
```

In these examples:

- The job MYJOB is allocated 4 megabytes of memory.

- The job step STEP1 is allocated 2 megabytes of memory.

Importance of the REGION Parameter

1. Resource Management: By specifying memory requirements, the REGION parameter helps in optimal resource allocation. This ensures that jobs have the necessary memory to run efficiently without over-allocating resources.

2. Performance Optimization: Adequate memory allocation can prevent job failures due to insufficient memory and can improve job performance by reducing paging and swapping.

3. System Stability: Proper use of the REGION parameter helps maintain system stability by preventing any single job from monopolizing system memory, which can affect other jobs and overall system performance.

4. Flexibility: It allows different jobs and job steps to request the memory they need, providing flexibility to handle various types of workloads, from small batch jobs to large data processing tasks.

Question:

What if REGION value on step level is more than job level, which will override?

In JCL, if the REGION parameter is specified at both the job level and the step level, the step-level REGION value overrides the job-level REGION value for that particular step. This allows more granular control over memory allocation for individual steps within a job.

Example

Consider the following JCL:

```
//MYJOB  JOB  (1234),'MY JOB',CLASS=A,MSGCLASS=X,REGION=4M

//STEP1  EXEC PGM=PROG1,REGION=6M

//STEP2  EXEC PGM=PROG2
```

- The job MYJOB is allocated 4 megabytes of memory at the job level.

- For STEP1, the REGION parameter is explicitly set to 6 megabytes, which overrides the job-level value of 4 megabytes.

- For STEP2, since no REGION parameter is specified at the step level, it uses the job-level value of 4 megabytes.

Detailed Breakdown

1. Job Level: When a job is submitted, the REGION parameter specified at the job level sets the default memory allocation for the entire job.

2. Step Level: If the REGION parameter is specified at the step level, it overrides the job-level REGION value for that particular step.

3. Memory Allocation: The system allocates memory based on the REGION value specified. If no step-level value is provided, the job-level value is used.

- NOTIFY Parameter

NOTIFY is used to notify the user about the job status (either successful or unsuccessful). NOTIFY is a keyword parameter and is optional.

Syntax -

NOTIFY=&SYSUID
or
NOTIFY=USERID

USERID	Specifies the userid to notify. It should be a valid TSO userid.
&SYSUID	The system identifies the userid who submits the job and sends the notification. While running the JOB, the system replaces &SYSUID with the userID who submitted the JOB.

- RESTART Parameter

RESTART is used to restart the job execution from a specific step (at which step the job failed). The step can be a job step, a procedure step, or a checkpoint.

RESTART is a keyword parameter and is optional. RESTART parameter is not always coded with the job card and only codes when resubmitting the job after abend.

When to RESTART?

Let us assume a job has multiple steps, and the job abended after a few steps of execution. The few steps that are completed successfully doesn't require to execute again. In this case, we will code the RESTART parameter at the job card with the failed step. Due to this, the completed steps are bypassed, and the job execution starts from the given step.

Syntax -

RESTART=stepname[.procname]
(or)
RESTART=*

*	Specifies the job execution starts from the first step. It is the default value if the RESTART parameter is not coded.
Stepname	Specifies the step name where the execution should start.
Procname	Specifies the PROC name in which procedure the RESTART step is coded. It is optional if the step is not in PROC.

Default and overrides -

If no RESTART is coded in the job, the execution will start from the beginning.

JOB starts execution from the first step of the JCL by default. If the restart step is different from the first step of the procedure, then the user needs to specify the step name in the PROC.