What is a Job?

On mainframes, you set up a Job to run a program. A job pre-specifies 3 things – which program to run, what are the input files and what are the output files. The computer program will read the set of input files, process the data, and store the results in the output files.

Once setup is complete, the computer operator submits(SUB) the job for execution. The job runs without any manual intervention. Depending upon the demand on the computer, the job may take a while to complete. On completion, a notification or alert is sent to the operator.

Batch-oriented applications are offline systems. You do not get immediate output/response to the inputs you supply. During execution, there is no interaction between the application program and the users.

What is the JOB Statement?

The //JOB statement identifies the job to zOS and supplies accounting-information.

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7-
//F0828A0B JOB A123,'BIN-7 QUASAR',CLASS=A,MSGCLASS=Y,MSGLEVEL=(1,1),
//            NOTIFY=&SYSUID
```

Job-name

F0828A0B is the job-name.

Account-number

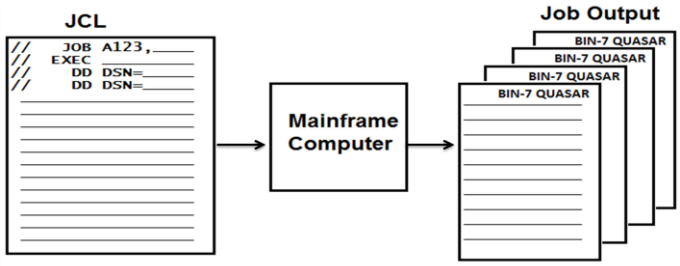
Just like you pay for the cellular phone services, mainframe computer-time is a paid-service. If your job takes 10 seconds of CPU-Time to complete, you will have to pay $0.40 x 10 sec = $4.00. The account number A123 will be charged $4.00. You may also specify additional accounting information, such as the account-name.

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7-
//F0828A0B JOB (A123,'QUASAR C'),'BIN-7 QUASAR',CLASS=A,MSGCLASS=Y,
//            MSGLEVEL=(1,1),NOTIFY=&SYSUID
```

Label/routing-information

As a job executes, the zOS prints messages about what the job is doing. You can code a label like BIN-7 QUASAR, so that when the job output is printed, each of the "separator pages" will be labelled at the top.

In the early days, when job outputs were printed and put in bins, 'BIN-7 QUASAR' would tell personnel operating the mainframe printer, to put my output in Bin number 7.



CLASS

CLASS parameter helps categorize a job. Generally, at most sites/projects, system programmers setup these categories, during the installation of the zOS. For example, in my project, they follow these settings – CLASS=A is used for small jobs, CLASS=E for express jobs, CLASS=G for medium-jobs and CLASS=L for long-running jobs.

Input Queue, Initiator and Output-Queue

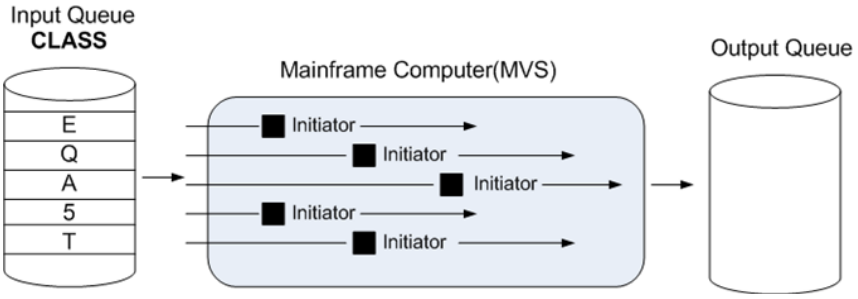
When you submit a job, the job(JCL) first gets stored in a disk-dataset(file), a staging area. This dataset is called the Input-Queue. Your job has to await its turn on the CPU, if there are other jobs in the queue.

There is not just one input-queue, but multiple queues(divided by CLASS). One for short jobs(CLASS=A), another for medium jobs(CLASS=G), another for quick express jobs(CLASS=E), another for long-running jobs(CLASS=L). The CLASS parameter on the JOB statement indicates, which Input-Queue CLASS(category) the job will go to.

When the job has reached the front of the queue, it is assigned an Initiator. We say, "the job is initiated".

When a job completes, the job outputs are stored in Output-Queue. Output-Queue is also called Spool.

The Initiator is like a personal assistant(helper). It guides the job, advises the job and assists it, all along its journey, right from picking it from the Input-Queue, throughout execution, till the job reaches the output-queue. Once an initiator becomes free, it goes back to the input-queue to pick another job.



MSGCLASS
Your job's output is brought to the Output-Queue. There's not just one output-queue, but there are several classes of Output Queues. MSGCLASS parameter tells in which output-queue, you want to store your job-output.

Generally, installation guys define different class codes (A,B,C,D,...,X) for different output queues. If you want to send your job-output for printing to the mainframe printer, you put it in the printer output-queue by coding MSGCLASS=P. If you want to send your job-output for punching, you put in Punch-queue by coding MSGCLASS=X.

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7-
//F0828A0B JOB (A123,'QUASAR C'),'BIN-7 QUASAR',CLASS=A,MSGCLASS=P,
//             MSGLEVEL=(1,1),NOTIFY=&SYSUID
```

MSGLEVEL
The MSGLEVEL parameter tells, how much level-of-detail should be printed in the Job-Output?

MSGLEVEL(0,1) – Only prints the job-statement.
MSGLEVEL(1,1) – Prints the maximum amount of detail. Complete original JCL plus Expanded PROCS

NOTIFY
The NOTIFY parameter is used to send an alert or notification the user-id specified in the NOTIFY Parameter.

What are the additional parameters you can code on JOB Statement?
TIME – The CPU-Time that a job is allowed to run. TIME=NOLIMIT or TIME=1440 means the job can run indefinitely.

REGION – The maximum memory space, the job can use.

PRTY – Inside an input queue, a job with higher PRTY(0-15) will be picked up first.

BYTES,CARDS,LINES,PAGES – Maximum amount of data in SYSOUT. For example, LINES=99999 means SYSOUT can store upto 99,999 lines.

TYPRUN – Special processing. TYPRUN=SCAN checks the job for JCL Errors, the job isn't executed.

What is the EXEC Statement?
The EXEC Statement in a job, represents one step. You code one exec statement in your job for each step. If your job has a dozen steps, you must code a dozen EXEC statements in your JCL.

One step executes(runs) one program. A program takes the data from input-files, processes it, and stores the results in the output-files. In JCL, you code DD Statements to pre-select input and output files.

```
EDIT       SYSADM.DEMO.JCLLIB(COPYPS) - 01.05              Columns 00001 00072
Command ===> _____________________________________________  Scroll ===> CSR
****** ***************************** Top of Data ******************************
=COLS> ----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
000001 //SYSADMA  JOB A123,'BIN-7 QUASAR',CLASS=A,MSGCLASS=Y,
000002 //             NOTIFY=&SYSUID
000003 //STEP01  EXEC PGM=IEBGENER
000004 //SYSUT1   DD  DSN=SYSADM.INPUT.DATA,DISP=SHR
000005 //SYSUT2   DD  DSN=SYSADM.OUTPUT.DATA,DISP=SHR
000006 //SYSIN    DD  DUMMY
000007 //SYSPRINT DD  SYSOUT=*
000008 //
****** **************************** Bottom of Data ****************************
```

What is PARM Parameter?
PARM Parameter can be used to pass data from JCL-to-program. This data is available in LS- Fields.

Program PROG62

```
=COLS> ----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
000001         IDENTIFICATION DIVISION.
000002         PROGRAM-ID. PROG62.
000003
000004         ENVIRONMENT DIVISION.
000005
000006         DATA DIVISION.
000007         WORKING-STORAGE SECTION.
000008         01  WS-PARM-DATA                    PIC X(10).
000009
000010         LINKAGE SECTION.
000011         01  LS-PARM-DATA.
000012             05 LS-PARM-LENGTH               PIC S9(04) COMP.
000013             05 LS-PARM                      PIC X(10).
000014
000015         PROCEDURE DIVISION USING LS-PARM-DATA.
000016         0000-MAIN.
000017             DISPLAY 'PARM LENGTH : ' LS-PARM-LENGTH ' BYTES'
000018             DISPLAY 'PARM DATA   : ' LS-PARM
000019             STOP RUN.
```

JCL to run the Program PROG62 which passes data 'HELLO PAL!'

```
000001 //SYSADMD  JOB (ABCDE),'QUASAR CHUNA',MSGCLASS=Y,
000002 //             MSGLEVEL=(1,1),CLASS=A,NOTIFY=&SYSUID,REGION=200M
000003 //JOBLIB  DD   DSN=SYSADM.DEMO.LOADLIB,DISP=SHR    <== Load Library
000004 //STEP01 EXEC  PGM=PROG62,PARM='HELLO PAL!'
000005 //SYSUDUMP DD  SYSOUT=*
000006 //SYSPRINT DD  SYSOUT=*
000007 //SYSOUT   DD  SYSOUT=*
000008 //
```

Output
```
PARM LENGTH : 0010 BYTES
PARM DATA   : HELLO PAL!
```

The first two bytes in the LS, always contain the length of the PARM data.

What is REGION and TIME Parameter?
TIME parameter specifies an upper-bound, an upper limit on the CPU Time a program gets. You code TIME=(min,sec). If a program exceeds its time limit, it abends with a code S322.

REGION parameter specifies an upper-limit on the amount of memory-space, that the program can use. If a program exceeds it region, it abends with a code S804.

```
000001 //SYSADMD  JOB (ABCDE),'QUASAR CHUNA',MSGCLASS=Y,
000002 //             MSGLEVEL=(1,1),CLASS=A,NOTIFY=&SYSUID,REGION=200M
000003 //JOBLIB  DD   DSN=SYSADM.DEMO.LOADLIB,DISP=SHR    <== Load Library
000004 //STEP01 EXEC  PGM=PROG62,REGION=256K
```

```
000005 //SYSUDUMP DD  SYSOUT=*
000006 //SYSPRINT DD  SYSOUT=*
000007 //SYSOUT  DD   SYSOUT=*
000008 //
```

JOB Level REGION and TIME limits always override step-level REGION and TIME.

Q. What is return-code? What is a job-abend? What is COND Parameter?

RETURN-CODE or COND CODE
When you run Cobol programs by setting up a job, and submitting it, how do you know, whether the Cobol program ran fine or not? Cobol programs leave behind a signature(trail) in the form of a two-digit status code, that indicates whether the program completed successfully, or it completed with errors. This 2-digit number to indicate success/failure is called Return-Code(condition-code).

The COND CODE is set by the Cobol program. A COND CODE = 00 could be successful completion, a code 05 could mean warnings, a code 10 might imply bad data(data-quality issue). If you have written your own custom Cobol program, you are the programmer, and you would set the return codes in Cobol program, by coding MOVE 00 TO RETURN-CODE, or MOVE 04 TO RETURN-CODE.

Generally IBM software such as the free programs IEBGENER, SORT, IEFBR14, IEBPTPCH, compiler IGYCRCTL, Linkage Editor IEWL and other ready-made utility programs, follow this convention with COND Code values –

COND CODE 00 – Program execution was completely successful.
COND CODE 04 – Program execution was successful, but with warnings.
COND CODE 08 – Program execution was flawed, it failed.
COND CODE 12 – Program execution was very seriously flawed, severe errors.
COND CODE 16 – Program execution disastrously failed.

COND Parameter
In a multi-step job-stream, that contains several steps, you often wish to turn off a certain step. When you want to turn off(shut down), when you don't want to run a step in a Job(JCL), you code COND parameter on the EXEC statement.

Thus, COND parameter is used to turn-off or skip a step.

I have written a simple ten-step toy job, to illustrate this idea.

```
=COLS> ----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
000001 //SYSADMA  JOB A123,'BIN-7 QUASAR',CLASS=A,MSGCLASS=Y,NOTIFY=&SYSUID
000002 //STEP010 EXEC PGM=IEFBR14
000003 //STEP020 EXEC PGM=IEFBR14
000004 //STEP030 EXEC PGM=IEFBR14
000005 //STEP040 EXEC PGM=IEFBR14
000006 //STEP050 EXEC PGM=IEFBR14,COND=(0,EQ,STEP030)
000007 //STEP060 EXEC PGM=IEFBR14
000008 //STEP070 EXEC PGM=IEFBR14
000009 //STEP080 EXEC PGM=IEFBR14
000010 //STEP090 EXEC PGM=IEFBR14
000011 //STEP100 EXEC PGM=IEFBR14
```

When you code COND parameter on a step, before running the step, the COND Parameter will be checked. If the COND condition is true, the step is turned off(shut off) or flushed. Only when condition is false, step runs.

In the above example, //STEP030 runs the ready-made program IEFBR14. IEFBR14 completes successfully and sets COND CODE=00. Before running //STEP050, the COND condition is checked. As 0 is equal To COND CODE of STEP030 i.e. 00, the STEP050 will be flushed.

Output in Spool

```
IEF142I SYSADMA STEP010 - STEP WAS EXECUTED - COND CODE 0000
IEF373I STEP/STEP010 /START 2012306.0641
IEF374I STEP/STEP010 /STOP  2012306.0641 CPU    0MIN 00.17SEC SRB    0MIN 00.00S
```

```
IEF142I SYSADMA STEP020 - STEP WAS EXECUTED - COND CODE 0000
IEF373I STEP/STEP020 /START 2012306.0641
IEF374I STEP/STEP020 /STOP  2012306.0641 CPU    0MIN 00.03SEC SRB    0MIN 00.00S
IEF142I SYSADMA STEP030 - STEP WAS EXECUTED - COND CODE 0000
IEF373I STEP/STEP030 /START 2012306.0641
IEF374I STEP/STEP030 /STOP  2012306.0641 CPU    0MIN 00.02SEC SRB    0MIN 00.00S
IEF142I SYSADMA STEP040 - STEP WAS EXECUTED - COND CODE 0000
IEF373I STEP/STEP040 /START 2012306.0641
IEF374I STEP/STEP040 /STOP  2012306.0641 CPU    0MIN 00.02SEC SRB    0MIN 00.00S
IEF202I SYSADMA STEP050 - STEP WAS NOT RUN BECAUSE OF CONDITION CODES
IEF272I SYSADMA STEP050 - STEP WAS NOT EXECUTED.
IEF373I STEP/STEP050 /START 2012306.0641
IEF374I STEP/STEP050 /STOP  2012306.0641 CPU    0MIN 00.00SEC SRB    0MIN 00.00S
IEF142I SYSADMA STEP060 - STEP WAS EXECUTED - COND CODE 0000
IEF373I STEP/STEP060 /START 2012306.0641
IEF374I STEP/STEP060 /STOP  2012306.0641 CPU    0MIN 00.03SEC SRB    0MIN 00.00S
```

EQ – Equal to **NE** – Not Equals
LT – Less than **LE** – Less than or equal to
GT – Greater than **GE** – Greater than or equal to

To turn off //STEP050, if any of the prior step fails.

```
//STEP050 EXEC PGM=IEFBR14,COND=(0,EQ)
```

If you want to make a compound-test(a test involving two to eight individual tests), you code COND in this way.

```
//STEP050 EXEC PGM=IEFBR14,(COND=(4,LT,STEP020),(0,NE,STEP040))
```

Job-Abend
Sometimes the program attempts to execute an instruction at the machine-code level, which is logically impossible to perform. The zOS has to then kill the program. The abnormal termination of the program or crash is called an Abend(like the blue screen of death on Windows). Every abend leaves behind a System-abend code Sxxxx.

There are also other problems like, out-of-space issues(dataset runs of out space), I-O Errors while opening a dataset that cause a crash(abend).

A Toy job

```
=COLS> ----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
000001 //SYSADMA  JOB A123,'BIN-7 QUASAR',CLASS=A,MSGCLASS=Y,NOTIFY=&SYSUID
000002 //STEP010 EXEC PGM=IEFBR14
000003 //STEP020 EXEC PGM=IEFBR14
000004 //STEP030 EXEC PGM=IEFBR14
000005 //STEP040 EXEC PGM=IEFBR14
000006 //STEP050 EXEC PGM=S806                <== RAISE AN ABEND
000007 //STEP060 EXEC PGM=IEFBR14             - THE JOB IS IN A RUSH TO
000008 //STEP070 EXEC PGM=IEFBR14             | GO TO THE ABEND EXIT DOOR.
000009 //STEP080 EXEC PGM=IEFBR14             | ALL THE REMAINING STEPS
000010 //STEP090 EXEC PGM=IEFBR14             | ARE FLUSHED OUT.
000011 //STEP100 EXEC PGM=IEFBR14             -
```

I have coded STEP050 to manually raise an Abend – it runs S806 program(well there's no such program really by the name S806), so the job abends at STEP050(Load Module not found – system abend code S806).

Ordinarily, a job is in Normal-Processing mode(all steps will run). When the job abends at STEP050, the job is in a hurry(rush) to go to the Abend Exit Door. It goes into Abnormal Termination mode. The Operating System says, "Nothing doing, you'll be kicked out, flushed!". The job can no longer continue any further processing on the Mainframe Computer, all the remaining steps STEP060, STEP070, STEP080, STEP090 and STEP100 are flushed(skipped), and the job is kicked out to the Output Queue Area.

COND=EVEN
When you code a COND=EVEN on a step, its like Force-executing a step.

```
=COLS> ----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
000001 //SYSADMA  JOB A123,'BIN-7 QUASAR',CLASS=A,MSGCLASS=Y,NOTIFY=&SYSUID
000002 //STEP010 EXEC PGM=IEFBR14
```

```
000003 //STEP020 EXEC PGM=IEFBR14
000004 //STEP030 EXEC PGM=IEFBR14
000005 //STEP040 EXEC PGM=IEFBR14
000006 //STEP050 EXEC PGM=S806                <== RAISE AN ABEND
000007 //STEP060 EXEC PGM=IEFBR14             - THE JOB IS IN A RUSH TO
000008 //STEP070 EXEC PGM=IEFBR14,COND=EVEN   | GO TO THE ABEND EXIT DOOR.
000009 //STEP080 EXEC PGM=IEFBR14             | ALL THE REMAINING STEPS
000010 //STEP090 EXEC PGM=IEFBR14             | ARE FLUSHED OUT.
000011 //STEP100 EXEC PGM=IEFBR14             -
```

Even if the job abends at any of the previous steps, force-run this STEP070. Even if the job is in Abnormal Termination Mode, run this step! It doesn't matter, if the job is in Normal processing mode or abnormal termination mode, //STEP070 will always run.

COND=ONLY
When you code COND=ONLY on a step, the step will run, only if the job is in abnormal termination mode.

Q. How do you run a particular step in a job using RESTART and COND Parameters?
You can also code the COND parameter at a Job-level on the //JOB statement. If the COND condition on the //JOB statement is true, then ALL steps of the job are turned off and flushed.

The RESTART parameter is used to RESTART the job from a specified step.

Suppose in a ten-step job, you want to run only the 5th Step.

```
=COLS> ----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
000001 //SYSADMA  JOB A123,'BIN-7 QUASAR',CLASS=A,MSGCLASS=Y,NOTIFY=&SYSUID,
000002 //            COND=(0,EQ),RESTART=STEP050
000003 //STEP010 EXEC PGM=IEFBR14
000004 //STEP020 EXEC PGM=IEFBR14
000005 //STEP030 EXEC PGM=IEFBR14
000006 //STEP040 EXEC PGM=IEFBR14
000007 //STEP050 EXEC PGM=IEFBR14
000008 //STEP060 EXEC PGM=IEFBR14
000009 //STEP070 EXEC PGM=IEFBR14
000010 //STEP080 EXEC PGM=IEFBR14
000011 //STEP090 EXEC PGM=IEFBR14
000012 //STEP100 EXEC PGM=IEFBR14
```

First, RESTART=STEP050 causes the job to begin execution from //STEP050. Since, this is the first step, the COND condition (0,EQ) is false. No steps ran before this, so there is no return code to compare. As COND condition is false, //STEP050 runs and completes, with COND CODE = 00.

Now, the COND condition (0,EQ) is true, as //STEP050's return code = 0. So, all the remaining steps of the job are turned-off.

Q. What is JOBLIB and STEPLIB? What is JCLLIB?
While executing a job, the mainframe computer has to first load the program(executable) from the DASD(Disk) into main-memory. Generally, you store programs(executables) together in a program library.

//JOBLIB and //STEPLIB DD statements are used to specify the library on the disk, from which the program has to be picked up.

The search-order is as follows:
STEPLIB
JOBLIB
System library SYS1.LINKLIB

If the program is not found in any library, the job abends with an abend-code S806.

During job execution, any PROCs in the main JCL, need to be expanded and inflated. PROCs are stored together in PROC library.

//JCLLIB ORDER statement is used to specify PROC libraries, from which the PROC should be picked up.

Q. What is DD Statement? What are DD DSN, DD *, DD DUMMY statements?
The DD statement is used to specify the input and output files.

```
=COLS> ----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
000001 //SYSADMA  JOB A123,'BIN-7 QUASAR',CLASS=A,MSGCLASS=Y,
000002 //            NOTIFY=&SYSUID
000003 //STEP01  EXEC PGM=IEBGENER
000004 //SYSUT1   DD  DSN=SYSADM.INPUT.DATA,DISP=SHR
000005 //SYSUT2   DD  DSN=SYSADM.OUTPUT.DATA,DISP=SHR
000006 //SYSIN    DD  DUMMY
000007 //SYSPRINT DD  SYSOUT=*
000008 //
```

Take a look at the job above. The free program IEBGENER is used to copy the contents of one file to another file. It requires 4 files(four DD statements).

```
//SYSUT1   - Input file
//SYSUT2   - Output file, to which the data will be copied.
//SYSIN    - Special control instructions file
//SYSPRINT - IEBGENER Messages
```

IEBGENER reads the data from //SYSUT1 file and copies it to //SYSUT2 file. You can supply special instructions to IEBGENER program in //SYSIN file. IEBGENER prints messages – about how many records were copied etc. in the //SYSPRINT file.

- DD DSN – Used to specify a dataset name.
- DD DUMMY – Turn off an input. For example, in the above job, if do not wish to supply //SYSIN special instructions, you can turn-off, seal the input by coding DD DUMMY.
- DD * - When you want to embed data/control instructions inside the body of the JCL inline instead of a separate file, you can code DD *.

Example –

```
=COLS> ----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
000001 //SYSADMA  JOB A123,'BIN-7 QUASAR',CLASS=A,MSGCLASS=Y,
000002 //            NOTIFY=&SYSUID
000003 //STEP01  EXEC PGM=IEBGENER
000004 //SYSUT1   DD  *
000005 001 RAM
000006 002 RAJ
000007 003 RAKESH
000008 004 RAVI
000009 005 RAJESH
000010 /*
000011 //SYSUT2   DD  DSN=SYSADM.OUTPUT.DATA,DISP=SHR
000012 //SYSIN    DD  DUMMY
000013 //SYSPRINT DD  SYSOUT=*
000014 //
```

Q. What are the different DISP parameters? What are DISP=MOD and DISP=PASS used for?

DISP Parameter describes the status of the dataset and directs the system, what to do with the dataset after completion of the job.

```
DISP=(status,successful-completion,abend)
      NEW     CATLG                   CATLG
      OLD     UNCATLG                 UNCATLG
      MOD     DELETE                  DELETE
      SHR     KEEP                    KEEP
              PASS
```

DISP Parameter for Output Files

- **DISP=(NEW,CATLG,DELETE)** is used to create a new output file. If the step completes successfully, the file is cataloged(added to the directory), so you can find it later in ISPF 3.4. If the step abends, the file is deleted.
- What if the dataset already exists? We use **DISP=OLD**, if the output file is already there.

DISP Parameter for Input Files

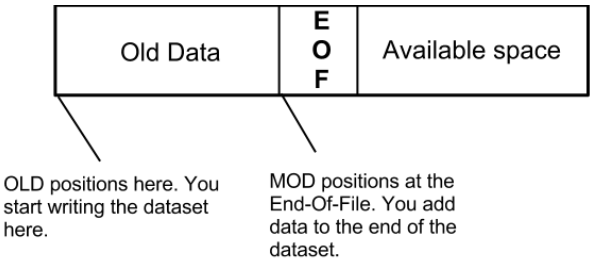
- **DISP=SHR** is used to **read data** from an existing Input File. **DISP=SHR** allows shared-access to the dataset. Sharing datasets is necessary, because public libraries such as SYS1.LINKLIB or sub-routine libraries should be available

to every other job in the system. **DISP=OLD** locks the file exclusively, so no other job can access the file at the same time.

DISP=MOD

- What if you do not know, whether the file exists or not? **DISP=MOD** works like NEW, and creates the file if it is not there. **DISP=MOD** works like OLD, if the file already exists. However, there is a subtle difference between DISP=MOD and DISP=OLD. **DISP=OLD** over-writes the contents in the existing file, **DISP=MOD** appends to it.

  Existing File



DISP=PASS

DISP=PASS passes temporary datasets, to subsequent job-steps. PASS saves time, because the dataset-location and volume-information is kept in memory. When a PASS'ed dataset is read in a subsequent step, the system does not need to refer to the catalog(VTOC) to locate the dataset.

Q. What is DASD? What is a Cartridge-Tape? What is the SPACE parameter?

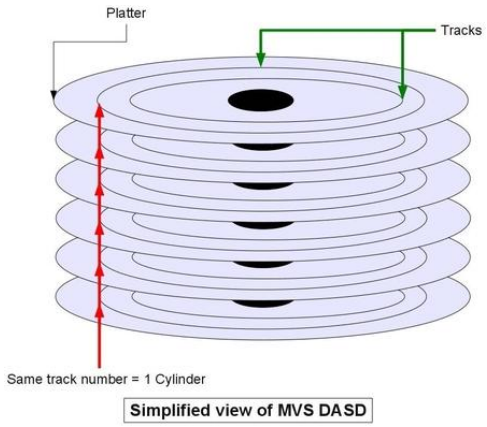
A mainframe computer has several peripheral storage media. Data on a mainframe computer is generally stored in Disk-drives and Tape-cartridges.

- DASD(pronounced as Dazz-dee) – It is the mainframe hard-disk drive, that allows random access to data. A DASD contains fifteen platters. Each platter has Tracks. Concentric tracks that lie on top of one another, form an imaginary cylinder.

  Data is recorded on Tracks, in blocks. Each block contains a bunch of logical-records.

Suppose BLKSIZE=800,LRECL=80,RECFM=FB. Then,
1 Block = 800 Bytes/80 Bytes = 10 records.

Just like a car has different models, IBM DASD Drives have different models – 3380, 3390 etc. Basically, each model offers higher storage capacity, packs more data on each platter. An IBM 3390 DASD device offers a density of 56,664 Bytes/track.



- Tape-Cartridge – It is like a magnetic cassette-tape, that allows sequential access to data.

Whenever you allocate a new dataset, you must specify how many tracks or cylinders of space you need.

SPACE=(TRK,(10,10))

1. First, the system allocate 10 Tracks of Primary Space.
2. If the dataset overflows the primary amount, the system then begins allocating the secondary amount of 10 tracks. The dataset is said to extend itself. This way, upto 15 extents for sequential file and 122 extents for VSAM file are possible. So, the dataset can grow upto 10 primary + 10 x 15 extents of secondary = 160 tracks. If the dataset needs more space, SE37(Space abend) occurs.

Q. How to create a partitioned-dataset(Library)?
Code SPACE=(TRK,(10,10,10)) to create a PDS with 10 directory blocks.

Q. What is RECFM=VB?
Logical-records can be fixed or variable-length. For a variable-length record, Logical Record-length = 4 Bytes(length) + data

Q. What are GDGs? How to create and use them?