JCL Refresher

JCL

Any business application is divided into logical modules and these modules are developed using programming languages. These programs should be executed in a pre-defined sequence to achieve the business functionality.

JCL (Job Control Language) is used to DEFINE and CONTROL the JOB to the operating system.

Definition involves definition of the programs need to be executed, the data for the programs and the sequence of programs. CONTROL involves controlling the execution or bypassing of a program in the sequence based on the result of the prior program execution.

JCL statements should have // in column 1 and 2. STAR (‘*‘) in the third column, indicates that the line is a comment line**.**

NAME is optional field. If coded, it should start at column3 and can have maximum 8 characters. The first character should be an alphabet or national character (@, # or $). Remaining characters can be any alphanumeric or national characters.

OPERATION follows NAME field. There should be at least one space between NAME and OPERATION. If NAME is not coded then OPERATION can start at fourth column itself. Typical OPERATION keywords are JOB, EXEC and DD.

OPERANDS are the parameters for the operation. OPERANDS follow OPERATION and there should be at least one space between them. A comma separates parameters and there should not any space between parameters. If the OPERANDS are more, then they can be continued in the next line. To continue the current line, end the current line before column 72 with ‘,’ and start the next line anywhere between
columns 4-16. Columns 1-3 should be ‘// ‘.

COMMENT FIELD – Comment field optionally follows OPERAND FIELD, preceded by at least one blank.

End of Job is identified by NULL statement. NULL statement has // in column 1 and 2 with no NAME, OPERATION or OPERAND fields. The statements coded after NULL statement will not be processed.

DELIMITER – Some times we pass the data in the JCL itself. This is called in-stream data. The starting of data is identified by '*' in the operand field of DD operation. DELIMITER indicates the end of data. /* in column 1 and 2 is the default delimiter.

Job Entry Subsystem (JES)

Job Entry Subsystem (JES) is the job processor of MVS operating system.
MVS installation can have either JES2 or JES3. The submitted jobs are taken by JES for processing.

JES2	JES3
Decentralized Environment. Every processor processes the incoming jobs individually.	Centralized Environment. There is a global processor that controls all the other processors and assigns the jobs to them.
Datasets are allocated before the step execution.	Datasets are allocated before the job execution.

JCL Statements

JOB. It should be the first statement in the JCL. It indicates accounting information and JOB related information to the system. If the member being submitted contains multiple job cards, then multiple jobs will be submitted. These jobs will run concurrently or one after other based on job name, class and initiator availability.

EXEC. The name of the program or procedure to be executed is coded here. Every EXEC statement in a JOB identifies one step. Maximum of 255 EXEC statements can be coded in a JOB.

DD. Data Descriptor. The dataset details are coded here. Dataset contains the data that need to be processed by the program or data that is produced by the program. Maximum 3273 DD statements can be coded in a step.

Abnormal End (ABEND) & ERROR

Once the work to be done is defined in JCL, it can be submitted to the operating system using SUBMIT command. Usually programmer is expected to issue JEM or JSCAN to check out any possible JCL Errors before submission.

JCL ERROR:

1. Errors before job starts execution: If there are syntax errors, then the whole job is rejected with error message in JES MESSAGES. Typically this needs correction and resubmission of the whole JOB.
2. Errors before step starts execution: If there is any allocation issues in a particular (like dataset not found, duplicate dataset), then also the job will be error out but in this case there might be already n steps got executed. Typically this needs correction and restart in the JOB.

ABEND:

Unlike JCL Errors, ABEND happens during the execution of a program in a step. ABENDS are classified into 2 categories.

System ABEND(Snnn): System abend occurs when the system is not able to execute a statement that is instructed in the program. Divide by ZERO results SOCB system abend. The OS throws it.

User ABEND(Unnnn): When some unexpected condition occurs in the data passed, the program will call an abend routine and abend the step with proper displays. This is thrown by application based on the requirement.

JOB Statement

Sample Syntax:

```
//JOBNAME JOB (ACCOUNTING INFO), (PROGRAMMER NAME),
//   TIME=(MINUTES,SECONDS), CLASS=A,MSGCLASS=A,PRTY=14,ADDR=VIRT,
//   REGION=nK,MSGLEVEL=(A,B),COND=(N,OPERATOR), TYPRUN=SCAN
```

JOBNAME

It identifies name of the job. The job is identified in the JES SPOOL using this name. Naming rules are already mentioned in the coding sheet section.

ACCOUNTING INFO (Mandatory. Installation Dependant.)

1. Resource usage charges are charged to the account mentioned here.
2. If you don't know your installation account, you cannot submit the job. It is like when you don't have account, you cannot withdraw cash in bank.
3. Maximum 142 characters can be coded as accounting information.

PROGRAMMER NAME

Programmer name or program functionality or group can be mentioned. It is used for documentation (Max 20 chars)

CLASS (Installation Defined)

1. CLASS is coded with single alphanumeric character (A-Z, 0-9).
2. During installation, every CLASS and its properties are defined.
3. Definition describes the job characteristics like CPU time usage, number of tape/cart usage and other resource constraints.
4. Every class is assigned to one or more initiators. The jobs run in initiator address space. One initiator can process one job at one time.

PRTY

Syntax: PRTY=N (N can be 0 –15).

1. While selecting the jobs with same class for execution, JES schedules the high priority jobs first. The job coded with PRTY=15 has the highest priority and PRTY=0 has the lowest priority.
2. PRTY works within the JOBCLASS. If there are 2 jobs with CLASS A is submitted and one with PRTY 3 and other with PRTY 4 then PRTY 4 will get into execution queue first.
3. PRTY function is disabled in most of the installations.

MSGLEVEL

Syntax: MSGLEVEL=(X,Y) (X can be 0-2& Y can be 0-1)

1. It is used to control the lists of information appear in the Job log. To get maximum information in the listing, code MSGLEVEL as MSGLEVEL(1,1)
2. The first parameter controls the statements. (0-Only job statement, 1-JCL, JES statements with expanded procedures, 2-Only JCL and JES statement).
3. The second parameter controls the messages. (0- Only Step execution messages, 1-All JCL, JES, operator and allocation messages).

MSGCLASS (Installation Defined)

Syntax: MSGCLASS=X (X can be A-Z, 0-9).

1. MSGCLASS is coded with single alphanumeric character (A-Z, 0-9).
2. Each MSGCLASS is mapped to a device or location, where the messages are routed.

ADDRSPC

It is used to specify whether the job will run in the Real storage or Virtual storage.

Syntax: ADDRSPC={REAL|VIRT}

REAL – Allocation is done in REAL storage and the program is not page-able.

VIRT – Allocation is done in VIRTUAL storage and the program is page-able.

REGION

Syntax: REGION={xK | yM} (x can be 1-2096128 & y can be 1-2047).

1. It is used to specify the amount of central /virtual storage the job requires. It can be requested in the units of kilobytes (xK) or megabytes (yM). If requested in terms of kilobytes, then x should be multiple of 4 or the system will round it to nearest 4K allocates for your job.
2. REGION can be coded in EXEC statement also. REGION parameter coded on JOB card overrides the parameter coded on EXEC card.
3. Maximum virtual memory available is 2GB.
4. Region=0M allocate all the available memory in the address space to this job.
5. Region related ABENDS: When the requested region is not available, the JOB will ABEND with S822. When the requested region is not enough for the program to run, you will get ABEND S80A or S804.

RESTART

RESTART parameter allows restarting from any particular step in the job.

Syntax: RESTART = Step-name in the job

RESTART = * means restart from the beginning.

To restart from any procedure steps, code RESTART=PROCSTEP.STEPNAME

Whereas PROCSTEP=name of the JCL step that invoked the PROC &

STEPNAME=name of the proc step where you want execution to start.

RESTART ignores any condition in the step being restarted and it can also be step

that is in the ELSE part of the IF..ELSE..ENDIF.

TYPRUN

It is used to request special job processing.

1. TYPRUN=SCAN checks the syntax errors without actual execution.
2. TYPRUN=HOLD checks the syntax error and if there is any error, it is notified and if there are no errors, the job is kept in awaiting execution queue and it should be released by user for execution. Release can be done by typing ‘A’ against the job name in SDSF.
3. TYPRUN=JCLHOLD Function is same as HOLD but the syntax check starts only after the release of the job.

TIME

It defines the maximum allowable CPU time for the JOB. The parameter can be coded at EXEC card also. On EXEC, it defines CPU limit of step.

Syntax: TIME = (MINUTES, SECONDS), MINUTES <= 1440 and SECONDS < 60
TIME=NOLIMIT/1440/MAXIMUM means the job can use CPU for unlimited time
TIME=0 will produce unpredictable results.

If TIME is coded on both JOB as well as EXEC, then EXEC Time limit or the time left out in the job Time limit – whichever is smaller will be the time permitted for the step to complete.

If a JOB runs more than allowed time, then it will ABEND with system ABEND code S322. If there is no TIME parameter, then the CPU time limit pre-defined with CLASS Parameter will be effective.

NOTIFY

TSO User-id to whom the job END / ABEND / ERROR status should be notified. NOTIFY=&SYSUID will send the notification to the user who submitted the job.

COND

1. It is used for conditional execution of JOB based on return code of JOB steps.
2. The return code of every step is checked against the condition coded on JOB card. If the condition is found TRUE, then all the steps following it are bypassed.
3. Maximum eight conditions can be coded in the COND parameter. In case of multiple conditions, if ANY of the condition is found TRUE then the JOB stops proceeding further.
   Syntax: COND=(CODE,OPERATOR,STEPNAME)

   STEPNAME is optional. If you code it, then that particular step-name return code is checked against the CODE with the OPERATOR. If omitted, then the return codes of all the steps are checked. On comparison, if the condition found to be true, then all the following steps are bypassed.

   CODE can be 0-4095

   OPERATOR can be GT, LT, GE, LE, EQ

   It can be coded on EXEC statement. STEP level control is popular then JOB level control. On EXEC statement, you may find ONLY, EVEN keywords against COND parameter.

   COND=ONLY allows the step execution only if any prior step is ABENDED.

   COND=EVEN allows the step execution independent of any prior ABENDS.

Consider the COND parameter coded on EXEC statement,

Ex: //STEP2 EXEC PGM=PGM3,COND=((16,GE),(90,LE,STEP1),ONLY)

Step gets executed only if

A preceding step abnormally terminated OR

The return codes from all preceding steps are 17 or greater OR

The return code from STEP1 is 89 or less.

EXEC Statement

It defines the Step and Step level information to the system.

Syntax: //STEPNAME EXEC {PGM=program-name |
PROC=proc-name |
proc-name}

STEPNAME

It is an OPTIONAL field but it is needed if you want to restart a job from this step and for the same reason, it should be unique within the job.

PGM or PROC

Code the Program name or PROC name to be executed. (1-8 characters)

PARM

1. It is used to pass variable information to the processing program, executed by this job step.
2. If there is more than one sub parameter or if there is any special character then enclose them in parentheses/quotes.
3. Maximum 100 characters can be passed in this way. Quotes and brackets are not included in this 100.To pass a quote to the program, indicate that with two quotes one followed by other.
4. The program can receive them using linkage section. Linkage section must be coded with half word binary field as first field. This field is populated with length of the PARM passed from the JCL.

Example:

```
//STEP3 EXEC PGM=WORK,PARM=(DECK,LIST,'LINECNT=80',
//      '12+80',NOMAP)
```

5. To continue the PARM in the second line, start the second line from 16th position.

DPRTY

PRTY assigns priority to a job and DPRTY assigns dispatching priority to job step. Syntax: DPRTY=(value1, value2). Value1 and value2 can be 0-15. D-Priority is calculated using the formula (value1*16 + value)

IF /THEN/ELSE/END-IF

It is used for conditionally executing one or more steps. Nesting is possible up to 15 levels. The meaning is same as programming IF. If the coded condition is true, the following steps till ELSE will be executed. If the condition is false, then the steps coded on ELSE part will be executed.

Syntax:

```
//name IF (relational operation) THEN
//…Steps..
//        ELSE
//…Steps..
//        ENDIF.
```

PROC PEND INCLUDE '/*' '// ' '//*' are executed irrespective of their place.

Don't specify JOBLIB, JCLLIB, JOBCAT, STEPCAT, JOB, SYSCHK within the THEN or ELSE Scope of IF statement.

Relational condition can be also coded as follows:

STEPNAME.ABEND=TRUE, STEPNAME.RUN=TRUE, STEPNAME NOT RUN

STEPNAME.ABENDCC = any-abend-code or

DD Statement

It defines the data requirements of the program. It is coded for every file used in the program. If the employee details are stored in a file and catalogued with the name 'SMSXL86.EMPLOYEE.DETAILS', one of the programs (EMPPGM) in the application reads this file, then JCL card for the DD looks like.

```
//STEP EXEC PGM=EMPPGM,REGION=1M
//EMPFILE DD DSN=SMSXL86.EMPLOYEE.DETAILS,DISP=SHR
```

When the program opens EMPFILE, it would open SMSXL86.EMPLOYEE.DETAILS.

The mapping is done as follows in the program. So in future, if you process the same program with another file SMSXL86.EMPLOYEE.DETAILS2, then change of dataset name in the JCL is enough. Because of logical mapping, program need not be changed.

SELECT file1 ASSIGN TO *EMPFILE*. (COBOL Program)
DECLARE (*EMPFILE*) FILE; (PL/I Program)
DCB DDNAME=*EMPFILE* (ASSEMBLER)
FOPEN(*EMPFILE*,mode) (C)

DSNAME(DSN)

The name of the dataset is coded in the DSN parameter. Dataset name can contain 44 characters including the periods in between qualifier. Each qualifier can have 8 characters and there can be 22 qualifiers. But usually we don't code more than 4 qualifiers.
DSN=XXXX.YYYY.ZZZZ,DISP=SHR

Temporary datasets are indicated by && in the DSN (DSN=&&temp).
If DSN is not specified, then the system assigns the specific name to dataset.
If DSN=NULLFILE or DUMMY is coded, then all the I/O s against this file are bypassed.

DISP

It is used to describe the status of a dataset to the system and instructs the system what to do with that dataset after successful/unsuccessful termination of the step or job.
DISP=(current-status, normal-termination-status, abnormal-termination-status)

Current-status can be:

NEW	Dataset does not exist. It will be created in this step
SHR	Dataset already exist and this step need it without any exclusive access
OLD	Dataset already exist and this step need it with exclusive access
MOD	If the dataset does not exist, it is to be created. If it already exists, records are to be added to the end of the dataset. In both the cases, exclusive is needed.

Normal-termination-status:

CATLG	System is to place an entry, pointing to the dataset in the system/user catalog
DELETE	Dataset is no longer required. Space available for use by another dataset but existing dataset not physically erased until overwritten by another dataset.
PASS	Dataset is passed to subsequent steps in the same job and each step can use the dataset only once.

KEEP	Dataset is to be kept on the volume
UNCATLG	System is to delete the catalog entry corresponding to this dataset and keep this dataset.

Abnormal-termination-status:
PASS is not allowed. Meaning of CATLG, UNCATLG, KEEP, DELETE are same as normal-termination status.

Absence of any parameter should be mentioned with ',' as they are positional parameters. Ex: DISP=(,,CATLG). If the dataset is a new dataset and DISP is not coded, then the default in effect would be *DISP=(NEW,DELETE,DELETE).*

DCB (Data Control Block)
DCB specifies attributes of the records in the dataset.
Syntax DCB=(LRECL=NN,BLKSIZE=YY,RECFM=Z,DSORG=MM,BUFNO=nn)

RECFM

It specifies format of the dataset. It can be Fixed, Variable, Undefined, Fixed Blocked, variable Blocked. (F, V, U, FB, VB). Other special record formats are VBS, FBS, VT, FT, FBA.

LRECL

It specifies logical record length. The length of the record is as in the program for fixed length records, length of the longest record with four more bytes for variable length records.

BLKSIZE

It contains physical record length. That is the length of the record in the storage medium. One block contains one or more logical records. It is suggested to code BLKSIZE as 0 so that the best size is chosen by the system, based on device.

If you explicitly code it, then it should multiple of LRECL for FB datasets and should not be less than length of the longest records with eight more bytes for VB dataset. In the extra eight bytes, four bytes are used for length of the record and four bytes are used for length of the block.

DSORG.

PS (Physical sequential) PO (Partitioned organization)

BUFNO.

The number of buffers to be allocated for the dataset is coded with BUFNO parameter. Maximum of 255 buffers can be coded. The performance of sequential processing will increase if more buffers are allocated. The default buffers are enough for most of the cases.

Source of DCB:

We don't always have to write the DCB parameter for a dataset. Writing DCB parameter is one of the three ways, the information can be supplied. The other two ways are:

1. Coded in the program. In COBOL, RECORD CONTAINS clause specifies the LRECL, BLOCK CONTAINS clause specifies the BLKSIZE, RECORDING MODE clause specifies RECFM and RESERVE clause specifies BUFNO. DSORG can be assumed from the name of the dataset and the directory space allocation of SPACE parameter.

2. Usually for an existing dataset, we don't have to code DCB parameters. It will be available in the dataset label. The dataset label is **STORED** in the VTOC (DASD) or along with dataset (TAPE) during the dataset creation.

LABEL

Syntax: LABEL = (Dataset-sequence-number
,label-type
,PASSWORD | NOPWREAD
,IN | OUT
,RETPD=nnn |EXPDT = (yyddd|yyyy/ddd))

Dataset Sequence number - identifies the relative position of a dataset on a tape/cart volume. Should be 1 through 4 decimal digits. Omit this parameter if access is being made to the first dataset on the tape volume.

Label - indicates the label type of the tape/cart volume.

SL - indicates that a dataset has IBM standard labels. Default value.

NL - indicates that a tape dataset has no labels.

NSL - indicates that a tape dataset has nonstandard labels.

SUL - indicates that a tape dataset has both IBM standard and user labels.
BLP - requests that the system bypass label processing for a tape dataset.

PASSWORD - indicates that a dataset cannot be read, changed, deleted or written to unless the correct password is specified.
NOPWREAD - indicates that a dataset cannot be changed, deleted or written, unless the correct password is specified. No password is necessary for reading the dataset.

IN - indicates that a dataset opened for I/O is to be read only.
OUT - indicates that a dataset opened for I/O is to be written only.

RETPD / EXPDT - indicates the retention period and the expiration date for a dataset.

Ex: LABEL=EXPDT=04121 (Dataset expires on 121st day of 2004)
LABEL=RETPD=200 (Dataset is retained for 200 days)

SPACE

It is used to request space for the new dataset. It is mandatory for all the NEW datasets.

SPACE = ({TRK | CYL | blklgth} (,Primary-qty , Second-qty, Directory)
[,RLSE] [,CONTIG] [,MXIG] [,ROUND])

TRK/CYL - Requests that space be allocated in tracks or cylinders.
Blklgth - Specifies the average block length, in bytes, of data. Specify a decimal number from 1 through 65535. This takes precedence, when specified, together with the BLKSIZE field of DCB parameter.

Primary-qty - Specifies the amount of primary space required in terms of the space unit (tracks/cylinders/number of data blocks). One volume must have enough space for the primary quantity. If a particular volume is requested and it does not have enough space available for the request, the job step is terminated.
Second-qty - Specifies the number of additional tracks, cylinders, blocks to be allocated, if additional space is required.
Directory - Specifies the number of 256-byte records needed in the directory of a PDS. (In every block we can store 5-6 members)

RLSE - requests that space allocated to an output dataset, but not used, is to be released when the dataset is closed. Release occurs only if dataset is open for output and the last operation was a write.

CONTIG - requests that space allocated to the dataset must be contiguous. It affects only primary space allocation.

MIXIG – It is used to specify that space requested should be allocated to the largest contiguous area of space available on the volume. It affects only primary allocation.

ROUND - When the first parameter specifies the average block length, this parameter requests that allocated space must be equal to an integral number of cylinders. Else ignored.

Extents

Extent is contiguous memory location. Only 16 extents are possible for a physical sequential dataset in a volume. In loose terms, only 16 pointers can be stored for a PS dataset in one volume. For a VSAM dataset it can be 123. In addition to this, the primary (first) or secondary (consecutive) space request has to be met within 5 extents.

If any of the above is not met, then there will be space ABEND. So for a PS dataset, even though you request 1600 tracks (using the space parameter (SPACE=TRKS,(100,100)), the system may not allocate you 1600 tracks always.

If all the contiguous available spaces are of size 20 tracks, then 5 extents are used for satisfying every primary or secondary. So 15 extents are used for providing just 300 tracks. If the system could find any 100 tracks for the 16th extent, it would offer it and if not, there will be space ABEND. So in the best case you will get 1600 tracks and in the worst case you will get 400 tracks for the space parameter mentioned.

Space ABENDS:

The most frequent ABEND in any production system is space abend.

SB37: End of volume. If the program tries to write more than the allocated space or if the system could not find the requested primary or secondary space even by joining 5 extents, then the step abnormally ends with SB37.

To solve this abend, increase the primary/secondary reasonably and if the job again comes down, create the dataset as multi-volume by coding VOL=(,,,3). As every volume will offer 16 extents, 48 extents in this case should be more than enough.

SD37: If the primary space is filled and the program tries to write more but no secondary is mentioned in the SPACE parameter, then the step will come down with SD37.To solve this abend, provide secondary allocation in the SPACE parameter.

SE37: End of Volume. This is same as SB37. You will get this ABEND usually for a partitioned dataset. To solve this, compress the PDS by typing ‘Z’ in ISPF 3.4 panel against the dataset or use IEBGENER. If again the job comes down, rename the old one, reallocate the new dataset with more space and copy the old members to here and delete the renamed dataset and restart the job.

UNIT

It is used to request the system to place the dataset on a specific device/a certain type or group of devices or the same device as another dataset.

UNIT=((device-number | device-type | group-name)
(,unit-count | P)
(,DEFER))
OR
UNIT=AFF=ddname

device-number - identifies a particular device by a 3-character hexadecimal number. It should not be used unless absolutely necessary.

device-type - Requests a device by its IBM supplied generic name. (Eg. 3380)

group-name - Requests a group of devices by its symbolic name. Installation must have assigned the name to the device(s) during system initialization.
The group-name is 1 through 8 alphanumeric characters (Eg. TEMPDA)

unit-count - Specifies the number of devices for the dataset. (1-59)

P - asks the system to mount all volumes for a dataset in parallel.

DEFER - Asks the system to assign the dataset to the device but requests that the volume(s) not be mounted until the dataset is opened. DEFER is ignored for a new dataset on direct access.

AFF=ddname - Requests that system to allocate different datasets residing on different removable volumes to the same device during step execution. The ddname is that of an earlier DD statement in the same step. It reduces number of devices used in a job step.

Ex: UNIT=(TAPE,,DEFER) UNIT=AFF=DD1

VOLUME

A reel of TAPE or a disk pack is called as one volume. VOLUME parameter is used to identify the volume(s) on which a dataset resides or will reside.

VOLUME = ((PRIVATE)

(,RETAIN)
(,volume-sequence-number)
(,volume-count))
(SER=serial-number1,
serial number2.....)

PRIVATE - Requests a private volume, that is exclusive use of volume for the dataset specified. Only one job can access it at a time. TAPES are PRIVATE by default.

RETAIN - Requests that volume is not to be demounted or rewound after the dataset is closed or at the end of this step. It is used when a following step is to use the same volume.

volume-sequence-number - identifies the volume of an existing multi volume dataset to be used to begin processing the dataset. (1-255)

volume-count - Specifies the maximum number of volumes that an output dataset requires. (1-255)

SER=serial-number - Identifies by serial number the volume(s) on which the dataset resides or will reside. 1 through 6 alphanumeric or national(@,#,$) characters.
You can code a maximum of 255 volume serials.
Ex : VOLUME=SER=DEV001 VOL=(,,,3,SER=(PAGE01,PAGE02,PAGE03))

SYSOUT=class | *

It is used to identify this dataset as a system output dataset. The SYSOUT dataset is assigned to an output class. The attributes for each class are defined during JES initialization, including device or devices for the output class. '*' refer-backs to MSGCLASS character of JOB CARD.

Positional and Keyword Parameters

All the parameters of JOB, DD and EXEC statements can be broadly classified into two types. They are POSITIONAL and KEYWORD parameters.

Parameter that has its meaning defined by its position is positional parameter. Bypassing of any positional parameter has to be informed to system by ',' Ex: accounting information and programmer name of Job card.

Keyword parameters follow positional parameter and they can be coded in any order. Ex: All the parameters suffixed by '=' are keyword parameters. PGM= and PROC= are exceptions for this rule. They are positional parameters.

In-stream data

The data passed in the JCL stream along with JCL statements is called in-stream data.

Syntax: //SYSIN DD &&&&

&&&&	Meaning
*	The data follows from the next line and ends when any // or /* appears at column 1& 2. So '//' and '/*' cannot be passed to the program. //EMPFILE DD * *2052MUTHU* *1099DEV* /*
DATA	The data follows from the next line and ends when any /* appears at column 1 & 2. So '/*' cannot be passed to the program. //SYSUT1 DD DATA *//STEP1 EXEC PGM=INV1040* *//INVLSTA DD SYSOUT=A* *//INVLSTB DD SYSOUT=A* /*
DATA, DLM=@@	The data follows from the next line and ends when the characters coded in DLM appears at column 1 & 2. //SYSIN DD DATA,DLM=## *//EMPFILE DD ** *2052MUTHU* *1099DEV* /* ##

OTHER Statements

OUTLIM

It limits the number of print lines. The limit ranges from 1 to 16777215. The job is terminated if the limit is reached.

//name DD SYSOUT=*,OUTLIM=3000

If the program tries to write 3001st line, JOB will ABEND with S722.

DEST

The DEST parameter is used in conjunction with the SYSOUT parameter where the output is to be sent. This might be used where a job is run on several MVS systems and the output is directed to a single system for the convenience of the end-user.
Syntax: //name DD SYSOUT=*,DEST=destination-ID

OUTPUT

OUTPUT statement is used to specify SYSOUT parameters for several DD statements with a single JCL statement. It allows printing the output from single DD statement several times, each with different SYSOUT parameters.
COPIES, OUTLIM, CLASS, DEST, FORMS, GROUPID can be coded in OUTPUT.

DEFAULT=Y can be coded on JOB and STEP level. STEP level default overrides JOBLEVEL default.

```
//TEST#10  JOB    ...
//STEP1    EXEC   PGM=ONE
//FORM2    OUTPUT DEFAULT=YES,COPIES=2,DEST=PPP
//SYSPRINT DD     SYSOUT=A    --> Produces 2 copies at PPP
//STEP2    EXEC   PGM=TWO
//FORM3    OUTPUT COPIES=3,DEST=XYZ
//SYSPRINT DD     SYSOUT=Q,OUTPUT=(STEP2.FORM3,STEP1.FORM2)
                                  -->Produces 3 copies at XYZ and 2 copies at PPP.
```

Maximum number of COPIES possible is 254.

FORMS

Specify the type of forms on which the SYSOUT datasets should be printed.
It is 1-8 alphanumeric or national character. SYSOUT DD FORMS parameter overrides OUTPUT PARMS parameter.
//name OUTPUT FORMS=form-name

FREE

The datasets are allocated just before the execution of step and de-allocated after the execution of step. FREE parameter de-allocates the file as soon as the file is closed.
//ddname DD SYSOUT=X,FREE=CLOSE

INCLUDE

The purpose of INCLUDE statement is same as COPY statement of COBOL program. This is used to specify a PDS member that will be copied into the JCL at job submission time. It is used to specify a standard list of DDNAMES, which would otherwise be duplicated in many similar PROCS. This also has the advantage that amendments need only be made in one place. But it makes JCL unnecessarily fragmented or difficult to read/maintain in a live environment.

```
//          INCLUDE MEMBER1
```

MEMBER1 should exist in the procedure library. Procedure libraries are coded using JCLLIB statement. Include must not be used to execute a PROC. It is possible to nest up to 15 levels of INCLUDE statements.

Concatenation Rules

Concatenation allows naming of more than one dataset in a single input file without physically combining them:

```
//STEPLIB DD DSN=PROD.LIBRARY,DISP=SHR
//          DD DSN=TEST.LIBRARY,DISP=SHR
//          DD DSN=USR.LIBRARY,DISP=SHR
```

In this case Prod, Test & User libraries are concatenated.
1.16 PDS or 255 Sequential datasets can be concatenated.
2.LRECL and Record format should be same.
3.If the Block size is different, then largest block size dataset should be first.
4.Datasets may reside on different devices and device types.

REFERBACK

The backward reference or refer back permits you to obtain information from a previous JCL statement in the job stream. STAR (*) is the refer-back operator.
It improves consistency and makes the coding easier.
DCB, DSN, VOL=SER, OUTPUT, PGM can be referred-back.

Refer back can be done using the following ways:

1.Another DD of the same step will be referred.

*.DDNAME

2.DD of another step can be referred

*.STEPNAME.DDNAME (DDNAME of the STEPNAME)

3.DD of another proc step can be referred.

*.STEP-INVOKING-PROC.PROC-STEP-NAME.DDNAME

STAR in the SYSOUT parameter refers back to MSGCLASS of JOB card.

Refer-back example:

```
//STEP1    EXEC PGM=TRANS
//TRANFILE DD   DSNAME=AR.TRANS.FILE,DISP=(NEW,KEEP),
//          UNIT=SYSDA,VOL=SER=MPS800,
//          SPACE=(CYL,(5,1)),
//          DCB=(DSORG=PS,RECFM=FB,LRECL=80)
//TRANERR  DD   DSNAME=AR.TRANS.ERR,DISP=(NEW,KEEP),
//          UNIT=SYSDA,VOL=SER=MPS801,
//          SPACE=(CYL,(2,1)),
//          DCB=*.TRANFILE
//STEP2    EXEC PGM=TRANSEP
//TRANIN   DD   DSNAME=*.STEP1.TRANFILE,DISP=SHR
//TRANOUT  DD   DSNAME=AR.TRANS.A.FILE,DISP=(NEW,KEEP),
//          UNIT=SYSDA,VOL=REF=*.STEP1.TRANFILE,
//          SPACE=(CYL,(5,1)),
//          DCB=*.STEP1.TRANFILE
.
//STEP5    EXEC PGM=*.STEP3.LOADMOD
```

Special DD names

STEPLIB

It follows EXEC statement. Load modules will be checked first in this library and then in the system libraries. If it is not found in both places, then the JOB would ABEND with S806 code.

JOBLIB

It follows the job statement. Load modules of any steps (EXEC) that don't have respective STEPLIB will be looked into this PDS. If not found, it will be checked against system libraries. If it is not found there also, then the JOB would ABEND with S806.

JCLLIB

It follows JOB statement. Catalogued procedures in the JOB are searched in this PDS. If they are not found, they will be checked in system procedure libraries.
If they are not there, then there will be JCLERROR with 'Proc not found' message.
Syntax: //PROCLIB JCLLIB ORDER(PDS1,PDS2)
INCLUDE members are also kept in procedure libraries. (JCLLIB)

ABEND DATASETS

In case of ABEND, one of the following three datasets will be useful. If more than one of the three datasets is coded, then the last coded DD will be effective.

SYSUDUMP

Prints the program area, contents of registers, and gives a trace back of subroutines called. It will be in hexadecimal format.

SYSABEND

Same as SYSUDUMP, but also prints the system nucleus. Don't use unless you need the nucleus. It will be in hexadecimal format.

SYSMDUMP

Same information as SYSABEND, but dump will be in machine language.
Used to store dumps in a data set to be processed by an application program.

JOBCAT and STEPCAT

The datasets used in step are first checked in the STEPCAT (ICF or VSAM Catalog) before checking in system catalog. If no STEPCAT in the step and there is a JOBCAT, then the datasets are first searched in JOBCAT before checking in system catalog.

SYSIN

In-stream data can be coded in SYSIN DD *. Using ACCEPT statement, these records are read into the program. Every accept will read one line into working storage (80 column).

Procedures

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.

If a procedure is defined in the same job stream, then it is called In-stream procedure. They are coded before the first EXEC statement in the job. The definition starts with PROC statement and ends with PEND. Instead procedures can be saved in a PDS and invoked from job and they are called as catalogued procedures. One procedure can call other. This is called nesting and nesting is possible up to 15 levels.

In-stream Procedure	Catalogued Procedure

//JOB1 JOB
//PROC1 EXEC PROC
//STEP1 EXEC PGM=IKJEFT01
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD
DISP=SHR,DSN=MT0012.MSG(TEST)
// PEND
//STEP01 EXEC PROC1
	//JOB1 JOB
	// JCLLIB ORDER=('MT0012.PROC.PDS')
	//STEP01 EXEC PROC1
	
	MT0012.PROC.PDS(PROC1)
	//STEP1 EXEC PGM=IKJEFT01
	//SYSTSPRT DD SYSOUT=*
	//SYSTSIN DD
	DISP=SHR,DSN=MT0012.MSG(TEST)
	/*

Procedure Modification

Procedure should be generic so that it can easily be used by the multiple JOBS by simple overrides. During the invoking of procedures in the JOB, one can do the following.

1. Override: Change the dataset names or parameters that are already coded in the procedure
2. Addition: Add new datasets or parameters in the already existing steps of the procedure.
3. Nullify: Omit the datasets or parameters that are already coded in procedure.

When you override a cataloged procedure, the override applies just to that execution of the job. The cataloged procedure itself isn't changed.

Procedure Modification- EXEC statements

COND, TIME and PARM values of an EXEC statement in the procedure can be added/modified/nullified in the invoking JCL in the following way.

//STEP1 EXEC PROC-NAME, PARAMETER-NAME.STEPNAME-IN-PROC=NEW-VALUE

Ex: PROC COBCLG has a statement

//COB EXEC PGM=IGYCRCTL,REGION=400K

// EXEC COBCLG, REGION.COB=1M => Overrides the value of 400K.

// EXEC COBCLG, TIME.COB=(0,10) => Adds 10 seconds time limit for COB step.

// EXEC COBCLG, REGION.COB= => Nullifies 400K region request. Default region will be allocated now.

Other Rules:

1. Multiple overrides are allowed but they should follow the order. That is first you should override the parameters of step1, then step2 and then step3. Any overrides in the wrong order are IGNORED.
2. If the STEPNAME is not coded during override, then the system applies the override to first step alone.

//EXEC COBCLG,REGION=512K

Procedure Modification- DD Statements

DD statement in the procedure can be modified by

//STEPNAME-IN-PROC.DDNAME-OF-STEP DD parameters of dataset...

1. DD statement overrides should be done in the same order they appear in procedure. Within a DD statement, the order of parameters does not matter.

2.Any additions should follow modifications. In a step, if you want to override the dataset attribute of one existing dataset and add another dataset, you should override the old one before adding the new one.

3. To omit a parameter from a DD statement in a procedure, just name it but don’t pass any value for it.

Procedure Modification Using Symbolic Parameter

A symbolic is a PROC placeholder. The value for the symbolic is supplied when the PROC is invoked. (&symbol=value). If the value is not provided during invoke, then the default value coded in the PROC definition would be used for substitution.

Ex: If you want to override UNIT Parameter value of all the DD statements, define this as symbolic parameter in proc.

Catalog Procedure: PROC1

```
//PROC1 PROC,UNIT=SYSDA            => SETS Default value of UNIT as SYSDA.
//S1      EXEC PGM=TEST1
//DD1 DD  UNIT=&UNIT
//DD2 DD  UNIT=&UNIT
```

//STEP1 EXEC PROC1,UNIT=TEMPDA will set &UNIT as TEMPDA for this run of procedure.

Statements Not Allowed in a Procedure

You can place most statements in a procedure, but there are a few exceptions. Some of these exceptions are:

1. The JOB statement and JES2/JES3 Control statements.
2. The JOBCAT and JOBLIB statement.
3. An instream procedure (an internal PROC/PEND pair)
4. SYSIN DD *, DATA statements

Nested Procedures-Add/Override/Nullification is applicable at only one level. In other words, if PROCA calls PROCB and PROCB calls PROCC, then no statement in PROCC can be overridden from PROCA invocation. Only PROCB can do that.

Procedure Example

SMSXL86.TEST.PROCLIB(EMPPROC)

```
//EMPPROC  PROC CLASS='*',SPACE='1,1'      ← Default values defined for CLASS
```

```
//STEP1A   EXEC PGM=EMPPGM                    and SPACE symbolic parameters.
//SYSOUT    DD   SYSOUT=&CLASS
//EMPMAST   DD   DSN=&HLQ..EMPLOYEE.EDS,DISP=SHR
//             DD   DSN=&HLQ..EMPLOYEE.IMR,DISP=SHR
//             DD   DSN=&HLQ..EMPLOYEE.VZ,DISP=SHR
//EMPOUT    DD   DSN=&&INVSEL,DISP=(NEW,PASS),  ← INVSEL is temporary
//                  UNIT=SYSDA,SPACE=(CYL,(&SPACE))     dataset
//EMPCNTL  DD   DUMMY
//* EMPCNTL is a control card and any in-stream data can be coded during the
//* invoke.
//*
//INV3020  EXEC PGM=EMPRPT
//SYSOUT   DD   SYSOUT=&CLASS
//INVMAST  DD   DSNAME=&&INVSEL,DISP=(OLD,DELETE)
//INVSLST  DD   SYSOUT=&CLASS
```

SMSXL86.TEST.JCLLIB(EMPJCL)

```
//EMPJCLA JOB (1000,200),CLASS=A,MSGCLASS=Q,NOTIFY=&SYSUID
//PROCLIB JCLLIB ORDER=(SMSXL86.TEST.PROCLIB)
//  SET SPACE='1,1'              ← Value is given for symbolic parameter SPACE.
//*STEP1A PARM is added and value for symbolic parameter HLQ is supplied.
//STEP01 EXEC  EMPPROC,PARM.STEP1A='02/11/1979',HLQ=PROD
//STEP1A.EMPMAST DD
//                    DD DSN=PROD.EMPLOYEE.CTS,DISP=SHR
//*Instead of PROD.EMPLOYEE.IMR, PROD.EMPLOYEE.TCS dataset is used whereas
//*other two datasets PROD.EMPLOYEE.EDS and PROD.EMPLOYEE.VZ retains their
//*position in concatenation.
//STEP1A.EMPOUT DD UNIT=TEMPDA
//*UNIT parameter of EMPOUT file is modified
//STEP1A.EMPCNTL DD *
DESIG=SSE
/*
//*EMPCNTL control card value is passed.
//STEP1A.EMPOUT2 DD DSN=PROD.EMPLOYEE.CONCAT,
//                     DISP=(NEW,CATLG,DELETE),UNIT=SYSDA,
//                     SPACE=(CYL,(10,10))
//*EMPOUT2 file is added to the step STEP1A.
```

In the above example, CLASS retains the default value coded on the PROC definition Statement (CLASS='*').

IEBCOPY

It is used to copy one or more members from an existing dataset to a new or existing PDS data set. It can be also used for compressing PDS, Loading PDS to TAPE and unloading from TAPE to disk. This utility needs two work files SYSUT3 and SYSUT4 in addition to SYSIN and SYSPRINT.

FIELD	Meaning
COPY	Function is COPY
SELECT	Specifies the members to be copied/replaced Syntax: (NAME-IN-OUTPUT,NAME-IN-OUTPUT,**R**EPLACE-IF-EXISTS)
EXCLUDE	Specifies the members to be excluded from copy
LIST=YES	Displays the copied members in the SYSPRINT.
INDD	Points to input dataset
OUTDD	Points to output dataset. Should exist on the same line of COPY.

IEBCOPY- CONTROL CARD FOR MERGING TWO LIBRARIES

```
//SYSIN DD *
COPY OUTDD=OUTPUT INDD=(INPUT01,(INPUT02,R),LIST=NO)
/*
```

It says DD statements INPUT01 and INPUT02 are input files. OUTPUT is the output file. Note the 'R' in (INPUT02,R). It instructs to IEBCOPY that like named members are to be replaced. LIST=NO indicates that the names of the members copied need not be listed in the SYSPRINT dataset.

IEBCOPY-CONTROL CARD FOR SELECTIVE COPY/REPLACE

```
COPY OUTDD=OUTPUT,INDD=INPUT01
SELECT MEMBER=((MEM1,NEWNAME,R),(MEM2,,R))
```

MEM1 is copied as NEWMEM in OUTPUT. If already NEWMEM exist, it will be replaced.

IEBCOPY-CONTROL CARD FOR OMITTING SELECTED MEMBERS

```
COPY OUTDD=OUTPUT,INDD=INPUT01
EXCLUDE MEMBER=(MEM1,MEM2)
```

All the members except MEM1 and MEM2 are copied into OUTPUT from INPUT01.

IEBCOPY-Complete step for Compressing PDS

```
//COMPRESS EXEC PGM=IEBCOPY
//SYSPRINT DD   SYSOUT=*
//COMPFILE DD   DSN=MM01.COPYLIB.COB,DISP=OLD
//SYSUT3    DD   UNIT=SYSDA,SPACE=(TRK,(1,1))
//SYSUT4    DD   UNIT=SYSDA,SPACE=(TRK,(1,1))
//SYSIN     DD   *
 COPY OUTDD=COMPFILE,INDD=COMPFILE
/*
```

IEBGENER

In addition to SYSIN and SYSPRINT datasets, it needs SYSUT1 and SYSUT2 datasets. SYSUT1 is coded with input dataset and SYSUT2 is coded with output dataset. If attributes were not given for SYSUT2, then the program would assume SYSUT1 attributes for SYSUT2.

It is primarily used as COPY utility. If you want to copy any TAPE file to DISK or DISK to TAPE, then no SYSIN is needed.

If you want to reformat your input file or if you want to create members out of your PS file, then you need control card (SYSIN) and the first statement should be GENERATE.

FIELD	Meaning
GENERATE	First Statement which sets the values for MAXNAME,MAXGPS, MAXLITS, MAXFLDS
MAXNAME	Maximum MEMBER statements that can follow.(During member generation) Syntax: MAXNAME=3
MAXGPS	Maximum IDENT statement that can follow. (During member generation)
MAXFLD	Maximum FILED statements that can follow. (During reformatting) Syntax: MAXFLDS=10
MAXLITS	Maximum size of literal during reformatting.
MEMBER	It identifies the name of the member to be created. Syntax: MEMBER NAME=MEM1
RECORD IDENT	It usually follows MEMBER statement to identify the last record to be copied from the input dataset.

	RECORD IDENT= (Length,'Literal',Start-Column) Example: RECORD IDENT=(3,'MVS',1), then the last record to be copied into the member from the input dataset, has MVS in column 1-3.
RECORD FIELD	It is used for reformatting the records in the input file. RECORD FIELD=(Length, 'literal' or input column, conversion, output column) Output column says where the field should be placed in the output file. Conversion can be ZP or PZ. PZ means the input packed decimal field is being converted into zoned format and ZP is the reverse.

IEBGENER- SYSIN CARD FOR CREATING THREE MEMBERS FROM INPUT PS FILE

```
//SYSIN DD *
 GENERATE MAXNAME=3,MAXGPS=2
 MEMBER NAME= MEMB1
 RECORD IDENT=(8,'11111111'.1)
 MEMBER NAME=MEMB2
 RECORD IDENT=(8,'22222222',1)
MEMBER NAME=MEMB3
//
```

IEBGENER creates three members. It reads input file writes into memb1 until it finds 11111111 in column 1. In the same way it reads and writes the records into memb2 until it finds 22222222 in column 1. The remaining records in the input dataset are copied into MEMB3.

IEBGENER- SYSIN CARD FOR REFORMATTING DURING COPY

```
//SYSIN     DD    *
 GENERATE MAXFLDS=5,MAXLITS=4
 RECORD FIELD=(5,1,,1),FIELD=(20,21,,6),FIELD=(9,61,ZP,26),          X
            FIELD=(9,70,ZP,31),FIELD=(4,'TEST',,36)
/*
```

Input Column	Any Conversion	Output column
Values in column 1-5		Copied into column 1-5
Values in column 21-40		Copied into column 6-25
Values in column 61-9	Convert the zoned into packed before copying.	Packed value is written in 26-30
Values in 70-9	Convert the zoned into	Packed value is written in

	packed before copying.	31-35
		TEST literal is written in column 36-39

IEHLIST

It is used to list

1. The entries in the catalog. (SYSIN KEYWORD- LISTCTLG)
2. Directory(s) of 1-10 PDS (SYSIN KEYWORD- LISTPDS)
3. The entries in VTOC. (SYSIN KEYWORD-LISTVTOC)

Code SYSIN, SYSPRINT and one more DD that will mount the volume queried in SYSIN.

The following JOB lists the VTOC of INTB01 in a formatted way.

```
//MYJOB    JOB  CLASS=A,MSGCLASS=A,REGION=256K,MSGLEVEL=(1,1)
//LISTVTOC EXEC PGM=IEHLIST
//SYSPRINT  DD  SYSOUT=*
//VOLDD     DD  UNIT=SYSDA,VOL=SER=INTB01,DISP=OLD
//SYSIN     DD  *
  LISTVTOC  FORMAT,VOL=3330=INTB01
/*
```

To list the contents of any PDS:

```
LISTPDS DSNAME=(SYS1.LINKLIB), VOL=SER=INTB01.
```

To list the catalog for a specific DASD volume:

```
LISTCTLG VOL=3350=PUB000
```

IEHMOVE

It is used to move one dataset from one volume to another volume.

```
//STEP01 EXEC PGM=IEHMOVE
//FROMVOL DD VOL=SER=TST001,DISP=OLD,UNIT=SYSDA /*ALLOCAT FROM VOLUME*/
//TOVOL   DD VOL=SER=PRD001,DISP=OLD,UNIT=SYSDA /*ALLOCATE TO VOLUME*/
//SYSPRINT DD SYSOUT=*
//DD01    DD UNIT=SYSDA,VOL=REF=SYS1.SVCLIB,DISP=OLD
//SYSIN   DD *
  MOVE PDS=MUTHU.TEST.PDS,TO=3380=PRD001,FROM=3380=TST001
```

/*

FROM clause in the SYSIN is not needed for catalogued datasets. It is suggested to allocate the Supervisor Call Library.

IEBCOMPR

It is used to compare two PS or PDS datasets. Two PS are same, if the number of records is same and all the records are identical. SYSIN is not needed for PS comparison. If they are not identical, then the following will be listed in the SYSPRINT.

DD statements that define the dataset, Record and Block numbers, the unequal records and maximum of 10 unequal records found.

Two PDS are same, if the corresponding members contain same number of records and all the records are identical. SYSIN should have COMPARE TYPE=PO for PDS.

```
//SYSUT1 INPUT DATASET 1
//SYSUT2 INPUT DATASET 2
//SYSPRINT
//SYSIN DD *
```

IEBBTPCH

IEBEDIT:

One typical interview question is how to run the selected steps. For example, how to execute step4 and step9 of 10 steps JCL. The typical answer is to restart the job from step4 and include a 'ALWAYS TRUE' condition (like COND=(0,LE) or COND=(4096,GT)) in steps 5,6,7,8 and 10. If the interviewer said COND should not used, then only way is IEBEDIT.

```
//M665235C JOB (MVSQuest),'IEBEDIT TEST',
//           CLASS=B,MSGCLASS=X,NOTIFY=V665235,REGION=28M
//*
//SUBMIT   EXEC PGM=IEBEDIT
//SYSUT1   DD DSN=TEST.MUTHU.JCL(JCLINP),DISP=SHR
//SYSUT2   DD SYSOUT=(*,INTRDR)
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
 EDIT START=M665235C,TYPE=INCLUDE,STEPNAME=(STEP0004,STEP0009)
//*
```

In the above JCL, JCLINP is the 10 steps JCL. M665235C is the job-name in the JCL.

If TYPE is exclude, then the mentioned steps will not be copied/submitted.

DFSORT

If you do a global search of your JCL inventory, you will find the program that is used very frequently is SORT. There are two famous SORT products are available in the market. One is DFSORT and the other is SYNCSORT. The basic commands in both the products are same.

ICETOOL provides a lot more than what SORT can offer and it comes with DFSORT product. SYNCTOOL comes with SYNCSORT product. PGM=SORT can point to DFSORT or SYNCSORT. It is actually an alias to SORT product in your installation.

DFSORT is IBM product and it needs the following datasets for its operation. SORTIN (Input dataset), SORTOUT (Output dataset), SYSIN (Control Card) and SYSOUT (Message dataset).

Message dataset can be altered using MSGDDN= parameter of SYSIN.

SORT Card to copy all the records from SORTIN to SORTOUT

SORT FIELDS=COPY.

SORT card to skip first 100 records and then copy 20 records

SORT FIELDS=COPY SKIPREC=100 STOPAFT=20

SORT Card to sort the records based on key fields

SORT FIELDS=(STARTPOS,LENGTH,TYPE,**A**SC|**D**ESC)

Type = CH (Character), BI (Binary), ZD (Zoned Decimal), PD(Packed Decimal),
FS (Signed numbers)

Ex: SORT FIELDS=(1,10,CH,A,15,2,CH,A)

SORTS all the SORTIN records with 1-10th column as major key and 15-16th column
as minor key before writing to SORTOUT.

SORT card to select the records meeting the criteria

INCLUDE COND=(STARTPOS,LENGTH,TYPE,RO,VALUE)

RO-Relational operator can be EQ,NE,LT,GT,LE,GE.

Card to select the records with TRICHY in the column 4-9

INCLUDE COND= (4,6,CH,EQ,C'TRICHY')
Card to select the records which has same values in 2-3 and 5-6
INCLUDE COND= (2,2,CH,EQ,5,2,CH)

SORT card to reject the records meeting the criteria

OMIT COND=(STARTPOS,LENGTH,TYPE,RO,VALUE)
Card to reject the records with TRICHY in the column 4-9
OMIT COND= (4,6,CH,EQ,C'TRICHY')
Card to reject the records which has same values in 2-3 and 5-6
OMIT COND= (2,2,CH,EQ,5,2,CH)

SORT card to change PD to ZD
If input file has a PD field S9(5)V99 Comp-3 and to reformat as PIC S9(5).9(2) then use,
OUTREC FIELDS=(1,5,PD,EDIT(STTTTT.TT),SIGNS=(,-,,))

SORT card to remove the duplicates
SORT FIELDS= (1,5,CH,A),EQUALS
SUM FIELDS=NONE.

SORTIN records are sorted on the key 1-5 and if more than one record is found to have same key, then only one record will be written to SORTOUT. If EQUALS is coded, then the record to be written is the FIRST record else it can be anything.

SORT card to sum the values for same-key records

SORT FIELDS= (1,5,CH,A),EQUALS
SUM FIELDS=(10,5,PD)

SORTIN records are sorted on key 1-5 and if more than one record is found to have same key, then the records are summed on column 10-14 and one record is written with total sum.

SORT card to add sequence number to the output file

OUTREC=(1,20,SEQNUM,4,ZD) ➔ 4 digit zoned decimal sequence number is added with all the records of input file in column 21-24

This will be useful for reading a file from bottom to top. This will be useful for matching logic in JCL. Matching logic in JCL will be explained later.

SORT card to restructure the input file before feeding to sort

INREC FIELDS=(37,2,6,6,40,4,31,2)
The length of the output file is 14.

SORT card to create multiple files from single input file (Maximum 32 files)

OUTFIL FILES=1 INCLUDE=(1,6,CH,EQ,C'MUMBAI')
OUTFIL FILES=2 INCLUDE=(1,6,CH,EQ,C'TRICHY')
Code output files as SORTOF1 and SORTOF2.

SORT card to restructure the sorted file before writing

```
OUTREC FIELDS=(1:1,20,          => FIRST 20 CHAR FROM INPUT FILE
               21:C'MUTHU',     = > FOLLOWED BY STRING 'MUTHU'
               26:10Z,          => FOLLOWED BY 10 BINARY ZEROS
               36:21,10)        =>  21ST to 10 CHARACTERS FROM INPUT FILE.
```

SORT card to change any 'XX' in the column 6-7 to 'YY'

```
OUTREC FIELDS=(1:1,5,
               6:1,2,CHANGE=(2,C'XX',C'YY'),NOMATCH=(6,2),
               8,42)
```

SORT card to merge

MERGE FIELDS=(STARTPOS,LENGTH,TYPE,**A**SC|**D**ESC,STARTPOS,…)
128 such Keys can be given. Datasets to be merged are coded in SORTIN00 to SORTIN99.

SORT CARD to extract all the PROCS executed in a JCL

```
OPTION COPY
INCLUDE FORMAT=SS,COND=(1,81,EQ,C'EXEC',AND,1,81,NE,C'PGM=)
```

ICETOOL

DD statements in ICETOOL:

```
TOOLMSG     FOR   ICETOOL MESSAGES
DFSMSG      FOR   SORT MESSAGES
TOOLIN      FOR   ICETOOL-CONTROL-CARD
XXXXCNTL    FOR   SORT-CONTROL-CARD USED BY ICETOOL
                  XXXX is coded in USING clause of TOOLIN.
```

TOOLIN card to copy

COPY FROM(INDD) TO(OUTDD) (Up-to 10 DD can be given).

TOOLIN card to copy unique/selected duplicate records

SELECT FROM(INDD) TO(OUTDD) ON (STARTPOS,LENGTH,TYPE)
NODUPS/ALLDUPS/LOWER(n)/HIGHER(n)/EQUAL(n)/FIRST/LAST

NODUPS – COPY only the unique records.
ALLDUPS – COPY only the duplicates.
HIGHER(n) – COPY those duplicates that occurs more than n times (n => 1-99)
LOWER(n) – COPY those duplicates that occurs less than n times (n => 1-99)
EQUAL(n) – COPY those duplicates that occurs exactly n times (n => 1-99)
FIRST - Retains the first copy in case of duplicates
LAST - Retains the first copy in case of duplicates

TOOLIN Card to get the statistics of a numeric field

STATS FROM(INDD) ON(START,LENGTH,TYPE)
Print the maximum, average and total for numeric fields. (10 ON possible)

TOOLIN Card to get the number of unique values in a field

UNIQUE FROM(INDD) ON(START,LENGTH,TYPE)
Print count of unique values.

TOOLIN Card to get all the values for a particular field

DISPLAY FROM(INDD) ON(STARTPOS,LENGTH,TYPE) LIST(LISTDD)
Prints values of a field in the input dataset to LISTDD. (20 ON possible)

TOOLIN Card to get all the values for a particular field – With Occurrence constraint

OCCURS FROM(INDD) ON(STARTPOS,LENGTH,TYPE) LIST(LISTDD) OPTION
OPTION = > HIGHER(n) LOWER(n) EQUAL(n) ALLDUPS NODUPS
HIGHER(2) means only the values that are repeated more than 2 times is reported at LISTDD dataset.

TOOLIN Card to get number of records fell into the range mentioned

RANGE FROM(INDD) ON(START,LENGTH,FORMAT) LIST(OUTDD) options

Options are OCCURS HIGHER(n)/LOWER(n)/HIGHER(n1) LOWER(n2)/
EQUAL(n) NOTEQUAL(n)

It prints count of records meeting value criteria and the FORMAT should be numeric.

TOOLIN card to invoke SORT

SORT FROM(INDD) TO(OUTDD) USING(xxxx)

SORT statements should be coded under the DDNAME xxxxCNTL

Matching Logic in JCL

I have two files file1 and file2. I want to generate three reports out of these two files.

1.The first report should have records that exist in both files.

2.The second report should contain records that exist only in first file and not in second file.

3.The third report should contain records that exist only in the second file and not in the first file.

```
//STEP0100 EXEC PGM=ICETOOL
//*
//TOOLMSG   DD SYSOUT=*
//DFSMSG    DD SYSOUT=*
//IN1       DD *
1234567890
3456789012
5678901234
//IN2       DD *
3456789012
7890123456
8901234567
//T1        DD DSN=&T1,SPACE=(CYL,(5,5),RLSE),DISP=(,PASS)
//T2        DD DSN=&T2,SPACE=(CYL,(5,5),RLSE),DISP=(,PASS)
//INT       DD DSN=*.T1,DISP=(OLD,PASS),VOL=REF=*.T1
//          DD DSN=*.T2,DISP=(OLD,PASS),VOL=REF=*.T2
//FILEA     DD SYSOUT=*
//FILEB     DD SYSOUT=*
```

```
//OUT       DD SYSOUT=*
//TOOLIN    DD   *
  SORT FROM(IN1) USING(CTL1)
  SORT FROM(IN2) USING(CTL2)
  SORT FROM(INT) USING(CTL3)
//CTL1CNTL  DD *
  SORT FIELDS=(1,10,CH,A)
  OUTFIL FNAMES=T1,OUTREC=(1,80,C'1')
//CTL2CNTL  DD   *
  SORT FIELDS=(1,10,CH,A)
  OUTFIL FNAMES=T2,OUTREC=(1,80,C'2')
//CTL3CNTL  DD   *
  SORT FIELDS=(1,10,CH,A)
  SUM FIELDS=(81,1,ZD)
  OUTFIL FNAMES=OUT,INCLUDE=(81,1,ZD,EQ,3),OUTREC=(1,80)
  OUTFIL FNAMES=FILEA,INCLUDE=(81,1,CH,EQ,C'1'),OUTREC=(1,80)
  OUTFIL FNAMES=FILEB,INCLUDE=(81,1,CH,EQ,C'2'),OUTREC=(1,80)
/*
```

Explanation:

CTL1 – Add 1 to all the records of the first file at 80th column

CTL2 – Add 2 to all the records of the second file at 80th column

CTL3 – Concatenate both files and sort the file on key if duplicates found, sum on 81st column. So if any record exists in both the file, it will have 3 after summing.

So now extract records with '1' , '2' and '3' into three files. While writing the records, remove the 81st byte added for our temporary purpose.

'1' – Records only in first file

'2' – Records only in second file.

'3' – Records exist in both the files.

<u>IEHPROGM</u>

It is used to

1.Catalog a dataset (CATLG DSNAME=A.B.C, VOL=SER=nnnn)

2.Uncatalog a dataset (UNCATLG DSNAME=A.B.C)

3.Rename a dataset (RENAME DSNAME=A.B.C,VOL=SER=nnnn,NEWNAME=D.E.F)

4.Create an index for GDG (BLDG INDEX=gdg-name, LIMIT=n, [,EMPTY][,DELETE])

5.Deleting the index for GDG (DLTX INDEX=index-name)

The SYSIN cards are given in bracket. The utility needs two work datasets and SYSPRINT for messages. Continuation of control card needs to be indicated by 'X' in 72nd column.

If your shop installed SMS, then uncatalog wont work it out because SMS handles the catalog.

IEHINITT

It is used to initialize a tape. It will write the volume serial number to the tape.

```
//STEP01 EXEC PGM=IEHINITT
//SYSPRINT DD SYSOUT=*
//LABEL DD DCB=DEN=3,UNIT=(TAPE,,DEFER),DISP=(,KEEP),LABEL=(,SL)
//SYSIN DD*
 LABEL INITT SER=DEVSMS,DISP=REWIND
/*
```

Generation Data Group (GDG)

GDG is group of datasets that are related to each other chronologically or functionally. Each of these dataset is called a generation. The generation number distinguishes each generation from others.

If the GDG Base is MM01.PAYROLL.MASTER, then their generations are identified using the generic name "MM01.PAYROLL.MASTER.GnnnnVxx."
nnnn is generation number (01-9999) and xx is version number (00-99).

Referring Generations in JCL

The current generation is referred by GDGBASE(0), the previous generation by GDGBASE(-1) and the next generation by GDGBASE(+1).

GENERATIONS ARE UPDATED ONLY AT THE END OF THE JOB. It means, if the first step creates one generation, code it as GDGBASE(+1) and if the second step creates another generation, then it SHOULD be coded as GDGBASE(+2) as the (+1) version is not yet promoted to current version. Similarly to refer the GDG created in the second step, refer it by GDGBASE(+2).

GDG datasets can be also referenced with their generation number like 'MM01.PAYROLL.MASTER.G001V00'

Advantage of GDG

1. GDG datasets are referred in the JCL using GDG base and relative number. So the same JCL can be used again and again without changing the dataset name and this is the biggest advantage of GDG.

2.GDG Base has pointers to all its generations. When you want to read all the transactions done till today, you can easily do it by reading the GDG base if it is available. Otherwise you have to concatenate all the transaction files before reading.

Creation of GDG

1.GDG Base is created using IDCAMS. The parameters given while creating the GDG are:

Parameter	Purpose
NAME	Base of the GDG is given here.
LIMIT	The maximum number of GDG version that can exist at any point of time. It is a number and should be less than 256.
EMPTY/NOEMPTY	When the LIMIT is exceeded, EMPTY keeps ONLY the most recent generation. NOEMPTY keeps the LIMIT number of newest generation.
SCRATCH/ NOSCRATCH	SCRATCH un-catalogue and deletes the versions that are not kept. NOSCRATCH just does un-cataloguing and it is not physically deleted from the volume.
OWNER	Owner of the GDG.
FOR DAYS (n) / TO (DATE)	Expiry date. Can be coded either in the unit of days or till particular date.

2. Model dataset is defined after or along with the creation of base. Once model DCB is defined, then during the allocation of new versions, we no need to code DCB parameter. Model DCB parameter can be overridden by coding new parameter while creating the GDG version. It is worth to note that two GDG version can exist in two different formats.

A step that defines a GDG and allocates a model DSCB

```
//GDG         EXEC PGM=IDCAMS
//SYSPRINT DD   SYSOUT=*
```

```
//MODEL    DD   DSNAME=MM01.PAYROLL.MASTER,DISP=(,KEEP),
//             UNIT=SYSDA,VOL=SER=MPS800,SPACE=(TRK,(0)),
//             DCB=(DSORG=PS,RECFM=FB,LRECL=400)
//SYSIN    DD   *
 DEFINE GDG ( NAME(MM01.PAYROLL.MASTER)  -
            LIMIT(5)                -
            NOEMPTY                  -
            SCRATCH )
/*
```

How to Solve ABENDS

There are two kinds of abends- USER and SYSTEM.

USER (Unnnn)	SYSTEM (Snnnn)
Prefixed with U	Prefixed with S
Application Driven – The application program issues the user abend by calling installation specific abend routine. IBM Supplied abend routine: ILBOABN0	System Driven – When the system is not able to perform a statement, it abended with respective system abend.
MOVE 999 TO ABEND-CODE CALL 'ILBOABN0' USING ABEND-CODE This code Will abend the program with U0999.	
Ex: In a billing application, the country referred in the bill-to address is missing in the country table which is an serious issue and so the program will be written in such a way to abend in this scenario.	Ex: Trying to do an arithmetic operation on non-numeric data. As system could not perform this, it will abend with S0C7.
Solution: Look for the abend code in the program and study the application logic behind this abend. Then appropriately fix the data/rerun.	Solution: Refer IBM Manuals for system abend and take appropriate action. You may need to analyse program/data to fix based on type of abend.

System abends can be environment (space/region) related or program related(logic/data driven logic). Fixing a program related abend is comparatively complicated than enviroment related system abends or user abends.

The usual process to solve a system abend is:

1.Refer the SYSOUT of the job and get the next sequential instruction to be executed (Offset).
2.Compile the program with LIST option if the compiled one is not with either LIST or OFFSET option.
3.Check for the offset in the compilation list and Get the respective statement number.
4.Identify the statement. This would be a numeric operation on non-numeric data.
5.Identify the source of the non-numeric data and correct it.

Though the process looks simple and this will work 90 percent of the cases, the questions are:

In case offset and the abending module is not displayed in the sysout, how to proceed?

In case of data-exception kind of abend, if the particular statement referred more than one field, how do you conclude which field have problem without display and rerun?

If the source for the field in problem is in file, how do you know which record of this field in the file?

Dump Reading exposure will help you in all these cases. As dump reading knowledge is important for any maintenance / support project, let us study with a simple example:

Simple Program

```
 CBL LIST
IDENTIFICATION DIVISION.
PROGRAM-ID.SANSOFT.
*
ENVIRONMENT DIVISION.
*
DATA DIVISION.
WORKING-STORAGE SECTION.
01 WS-VARIABLES.
   05 WS-EMP-NAME       PIC X(10).
   05 WS-EMP-AGE        PIC 9(02).
   05 WS-EMP-CITY       PIC X(10).
   05 WS-EMP-SAL        PIC S9(08).
   05 WS-EMP-BONUS      PIC S9(08).
```

```
    05 WS-EMP-CTC        PIC S9(08).
*
 PROCEDURE DIVISION.
*
     MOVE 'MUTHU'         TO WS-EMP-NAME
     MOVE 29              TO WS-EMP-AGE
     MOVE 'TRICHY'        TO WS-EMP-CITY
     MOVE  60000          TO WS-EMP-SAL
     COMPUTE WS-EMP-CTC = (WS-EMP-SAL * 12) + WS-EMP-BONUS
     DISPLAY 'SANSOFT  COMPLETED'
     DISPLAY 'EMPLOYEE DETAIL:' WS-EMP-NAME ','
                        WS-EMP-AGE  ','
                        WS-EMP-CITY ','
                        WS-EMP-SAL  ','
                        WS-EMP-CTC
     STOP RUN.
```

The program is compiled with LIST option to get assembler listing.

On successful compilation, the program is submitted with the JCL.

The program is abended and the sysout says:

```
CEE3207S The system detected a data exception (System Completion
Code=0C7).
          From compile unit SANSOFT at entry point SANSOFT at compile unit
offset +0000036A at entry offset +0000036A
      at address
2790117A.
```

The instruction at the Offset 36A is failed. So look into the compilation listing for the statement that is in the offset 36A.

```
000021  MOVE

    000358  D207 2016
A0BD  MVC   22(8,2),189(10)          (BLW=0)+22       PGMLIT AT
+185
 000022  COMPUTE

    00035E  F247 D0F8
201E  PACK  248(5,13),30(8,2)        TS2=0            WS-EMP-
BONUS
  000364  D20F D0E8 A08D  MVC   232(16,13),141(10)     TS1=0        PGMLIT AT
+137
  00036A  FA54 D0F2 D0F8  AP    242(6,13),248(5,13)    TS1=10       TS2=0
    000370  940F
D0F3       NI    243(13),X'0F'            TS1=11

    000374  F844 D0F3
D0F3  ZAP   243(5,13),243(5,13)      TS1=11           TS1=11

    00037A  F374 2026 D0F3  UNPK  38(8,2),243(5,13)       WS-EMP-
CTC       TS1=11
 000023  DISPLAY
```

35E-37F belongs to the COMPUTE statement that is in line 000022.

Now look for line 00022 in compilation listing.

```
   000021        MOVE  60000          TO WS-EMP-
SAL
   000022        COMPUTE WS-EMP-CTC = (WS-EMP-SAL * 12) + WS-EMP-
BONUS
   000023        DISPLAY 'SANSOFT
COMPLETED'
```

So one of this field referred in this statement has junk in it. Just before compute we populated WS-EMP-SAL and so there is a problem with WS-EMP-BONUS. If you go thru the

code, you will find the developer missed to populate/initialize WS-EMP-BONUS and that has caused data exception.

If these fields are from file, we cannot easily confirm like above. So we have to give display for these two fields in the program and rerun the program or look for junks in the source file for these two fields using FILE AID/ INSYNC. The other approach will be look into data division map in the compilation listing.

```
0Source   Hierarchy and                                        Base       Hex-
Displacement  Asmblr Data                  Data Def
 LineID   Data
Name                                        Locator    Blk   Structure   Defi
nition    Data Type      Attributes
      2  PROGRAM-ID SANSOFT-------------------------------------------------
-------*
      8   1  WS-VARIABLES. . . . . . BLW=00000  000              DS
0CL46      Group
      9     2  WS-EMP-NAME . . . . . BLW=00000  000   0 000 000   DS
10C        Display
     10     2  WS-EMP-AGE. . . . . . BLW=00000  00A   0 000 00A   DS
2C         Disp-Num
     11     2  WS-EMP-CITY . . . . . BLW=00000  00C   0 000 00C   DS
10C        Display
     12     2  WS-EMP-SAL. . . . . . BLW=00000  016   0 000 016   DS
8C         Disp-Num
     13     2  WS-EMP-BONUS. . . . . BLW=00000  01E   0 000 01E   DS
8C         Disp-Num
     14     2  WS-EMP-CTC. . . . . . BLW=00000  026   0 000 026   DS
8C         Disp-Num
```

WS-EMP-SAL is in BLW-0 offset 16 whereas WS-EMP-BONUS in offset 01E.

In the dump, look for working-storage dump. BLW-0 maps to 279890B8 address.

```
WORKING-STORAGE for
SANSOFT
```

```
     BLW-0:
279890B8

       +000000 279890B8  D4E4E3C8 E4404040 4040F2F9 E3D9C9C3
                         C8E84040 4040F0F0 F0F6F0F0 F0C00000
                        |MUTHU     29TRICHY    0006000...|
       +000020 279890D8  00000000 00000000 00000000 00000000
                         00000000 00000000 00000000 00000000
                        |................................|
```

BLW-0:
279890B8
+000000 279890B8 D4E4E3C8 E4404040 4040F2F9 E3D9C9C3 C8E84040 4040F0F0 F0F6F0F0 F0C00000 |MUTHU 29TRICHY 0006000...|
+000020 279890D8 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 |.....|

WS-EMP-NAME is 10 bytes starting at 0 - 'D4E4E3C8E44040404040' which is nothing but 'MUTHU '. WS-EMP-AGE is 2 bytes starting at 10th byte which is '29' ('F2F9'). Similarly read the contents of WS-EMP-CITY, WS-EMP-SAL, WE-EMP-BONUS and WS-EMP-CTC.

WS-EMP-BONUS contains low-values. It is a computational field and so it is expected to have sign in the last nibble whereas it doesnot as it is not properly initialised in the program.

In the dump, in every line there will be twenty bytes hexa decimal content will be followed by character content. Due to the column limit, I have the one line dump in three lines.

So if there are million records in a file and during the processing it abended after 'n' number of records, to identify the record caused problem, read the dump for the file section unique variable(s) values and look for the respective record in the file and analyse/correct/delete.

We have taken offset directly from sysout. If it is not available, then refer the PSW. Based on AMODE 24/31, the last 24/31 bits contain the next sequential instruction to be executed. From this value, subtract the entry point of the program being abended and that will give you offset. One instruction above this offset is the one that caused the abend. Entry point of all the programs executed be found in trace back or save trace section of the dump. You can directly found the offset in the traceback also. (last 31 bits of A7901180 is 27901180, 27901180-27900E10 = 370 is the offset, from the compilation listing one instruction just before 370 is 36A.)

PSW..... 078D1000 **A7901180**

```
Traceback:


     DSA Addr   Program Unit   PU Addr    PU Offset   Entry          E
Addr     E   Offset
     0001D120   CEEHDSP        057780F0   +00003C34   CEEHDSP        057780F0   +00
003C34
     0001D018   SANSOFT        27900E10   +0000036A   SANSOFT        27900E10   +00
00036A

      Statement   Load Mod   Service   Status

                  CEEPLPKA   UK00165   Call
                   SANSOFT              Exception
```

Submission of Job from COBOL Program

Write the Job statements to a output file and route the file to INTRDR instead of cataloguing. INTRDR-Internal Reader

```
JOB: //STEP1 EXEC PGM=MAINPGM
     //DD1     DD    DSN=MUTHU.TEST,DISP=SHR
     //JCLDD   DD    SYSOUT=(*,INTRDR)

PROGRAM:MAINPGM
      SELECT JCLFILE ASSIGN TO JCLDD…. (Environment Division)
```

```
FD JCLFILE.
01 JCL-REC PIC X(80).          (File Section)
OPEN OUTPUT JCLFILE.          (Open in output and write JCL statements)
MOVE '//TESTJOB JOB 1111'              TO JCL-REC.
MOVE '//STEP01 EXEC PGM=IEFBR14' TO JCL- REC
CLOSE JCLFILE                  (TESTJOB will be submitted automatically)
```

Submission of Job from CICS Program

JCL can be submitted from CICS by SPOOL OPEN, SPOOL WRITE and SPOOL CLOSE commands. This needs SPOOL=YES entry in SIT table. Refer CICS section for more details.

Storage Management Subsystem

It is optional feature of the operating system that manages user datasets. The SMS parameters apply only to the new datasets and the system ignores for existing datasets, DD* or DD DATA datasets, JOBCAT and JOBLIB DD statements, and SYSOUT datasets.

The following data classes establish various default values for catalogued datasets. An administrator assigns a name to each group of default values, and then you reference this name on your DD statements to use the values.

STORCLAS – Defines UNIT and VOL parameters
DATACLAS - Defines RECFM, LRECL, AVGREC, SPACE, RETPD/EXTPD, DSNTYPE, KEYLEN, RECORG, KEYOFF..etc
MGMTCLAS – Migration, backup, releasing unused space in a dataset.

If you want to override any one of the values of default, you can do that.

```
//PDS DD DSN=BPMAIN.MUTHU.SOURCE,DISP=(NEW,CATLG),
//          STORCLAS=DASDONE,SPACE=(,(,,50)),DATACLAS=COB2
```

➔ Overrides the directory space defined for COB2 data class.

JCL for SAR-IN-BATCH

The production Job logs are usually routed to one of the third party products. SAR is one such product. The other products are CA-VIEW and VIEW-DIRECT.

The following JCL is used to query the SAR in batch.

```
//LOADLOG EXEC PGM=SARBCH
//STEPLIB  DD DSN=SYSP.CAI.CAILIB,DISP=SHR
//SYSPRINT DD SYSOUT=*
//REPORT   DD DSN=CORP.DBSY2KT.JOBRPT,DISP=MOD
//LOADDD   DD DSN=CORP.DBSY2KT. LOADLOG,DISP=OLD
//SYSIN    DD DSN=CORP.DBSY2KT.UTILS(CYCCARDW),DISP=SHR
```

To load the complete log (JOB PRODBKUP of generation 4941) into a dataset named as LOADDD, use the following card:

```
/LOAD DDNAME=LOADDD ID=PRODBKUP GEN=4941
```

To get run-date, run-time, return code and generation of all the prior runs of a job, use the following card. The result will be stored in the dataset named as REPORT.

```
/LIST ID=JOBNAME
```

JCL for CA7-IN-BATCH

CA7 is a scheduler product. We brief about the product in the next section. This JCL is used to query CA7 in batch.

```
//UCC77BTI  PROC  ID=0,POOL='1-8',DSNPFX=,OPT=,PG=SASSBSTR
//BTERM     EXEC  PGM=SAASBSTR,
//            PARM='0,POOL=(1-8) '
//STEPLIB      DD  DISP=SHR,DSN=CAI.CA7.LOADLIB
//UCC7CMDS  DD  DISP=SHR,DSN=CAI.CA7.COMMDS
//SYSPRINT    DD  SYSOUT=*,DCB=BLKSIZE=133
//ERRORS      DD  SYSOUT=*,DCB=BLKSIZE=133
//SYSUDUMP  DD  SYSOUT=*
//SYSIN DD*
 Code the CA7 commands
/*
```

JCL FOR XDC-IN-BATCH

There may be a need to read the run details of a job log inside the REXX. This can be done using the OUTPUT command of TSO. The following JCL captures the complete log information of the job CENNAEFD with ID JOB08079 into T.ISP.MUTHU(TEST1).

```
//RUNTSO EXEC PGM=IKJEFT01
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
OUTPUT CENNAEFD(JOB08079) PRINT('T.ISP.MUTHU(TEST1)')
/*
//*
```

Posted by Sajid Mohammed
Labels: ***MAINFRAMES***