Version 3.1
28th May 2003

TABLE OF CONTENTS

Day-Wise Schedule

Day 1 Theory	Introduction to JCL Syntax of JCL JOB statement EXEC statement COND parameter DD statement DSN, DISP, SPACE, UNIT, VOLSER, DCB
Day 1 Lab	Assignment 1,2,3
Day 2 Theory	DD statement contd…. JOBLIB, STEPLIB, Storage Dump SYSUDUMP SYSMDUMP SYSABEND Procedures… Cataloged, regular overrides
Day 2 Lab	Assignment 4,5,6,7,8
Day 3 Theory	Procedures contd (….) Instream Symbolic parameter substitutions Utilities -IEFBR14 -IEBGENER -SORT
Day 3 Lab	Assignment 9,10,11,12.13

1 Introduction

1.1 What is JCL?

There are three types of work that a user can perform under MVS/XA:

- Time Sharing
- Online
- Batch

The first two appear very much alike to the user. Under both, the user logs on, using a ***userid*** and ***password*** and begins to work interactively. A transaction is entered and a response received, normally in very few seconds. A unit of work for both is called a *session.*

Batch processing is not interactive. It is deferred execution. It can be executed as soon as it is submitted or much later on. Also the execution of batch work can be time consuming as compared to interactive which is usually very quick.

Job Control Language (JCL) must be used in order to submit a batch work.

Job Control Language (JCL) is a means of communicating with the IBM 3090 MVS Operating System. JCL statements provide information that the operating system needs to execute a job. The unit of batch work is a “job”.

A job is something that you want to accomplish with the aid of a mainframe computer, e.g. copy a data set, execute a program, or process multiple job steps. You need to supply the information that the job requires and instruct the computer what to do with this information. You do this with JCL statements. A job can be defined as one or more steps up to 255. A job step consists of statements that control the execution of a program or procedure, request resources, and define input and/or output.

This includes information about:

- The program or procedure to be executed
- Input data
- Output data
- Output Reports

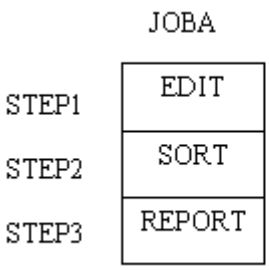


Fig 1.1

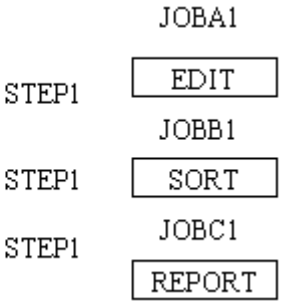


Fig 1.2

How do the two methods differ?

- Steps within a JOB have a predefined sequence
- Steps within a JOB cannot multiprogram with one another
- A Step within a JOB can interrogate the status of a preceding step's execution and determine whether to execute or not

In **Fig 1.1** a job is shown with three steps. STEP1 executes program EDIT, STEP2 executes program SORT and STEP3 executes program REPORT. In **Fig 1.2** the same three steps appear but as part of three different jobs. Both steps achieve the same goal but the two do not work the same way. The steps in Fig 1.1 must be executed in sequence and the execution of steps STEP2 and STEP3 can be controlled based on what happened to preceding steps. The steps in Fig 1.2 are within different jobs and can be executed concurrently (although sequential execution can be forced), and there is no communication between jobs. If these steps require sequential execution and execution control then the setup in Fig 1.1 is better. If not then the one according to Fig 1.2 is better.

1.2 Processing of JCL

Input stream is JCL and optionally, data, which comes together with JCL (80-byte records), known as SYSIN data or input stream data. Disk is the most common device used for submitting jobs.

JES2 or JES3 reads the JCL in the input stream and places it on the spool pack. JCL goes through one or more level of syntax checking. SYSIN data, if any, is also read in and placed on the spool pack by JES2 or JES3. If the JCL is syntactically correct it is queued for execution in Job Queue.

An initiator must be available to execute the job. An initiator is a set of routines whose sole function is to select a job from job queue and execute it under its control. During the job's execution SYSIN data, if any will be read in and print-lines, if any will be placed on spool pack for later printing. There are several initiators available, and each one performs the same function for a different job. The number of initiator varies according to the hardware configuration and the amount of concurrent time-sharing and on-line activity on the same system.

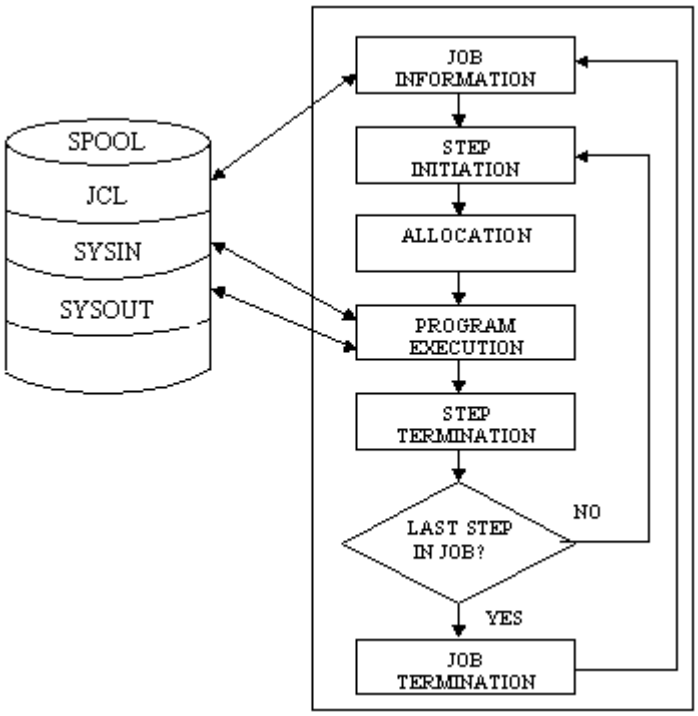


Fig 1.3

The above figure shows the working of an initiator. The initiator goes to the Job Queue and asks for a Job. If an appropriate job exists, it takes its JCL and goes through the following processes:

- **Job initiation:** One of the main functions of this process is to ensure that no data sets needed by this job are reserved by other jobs and, therefore, unavailable. If some data sets are reserved, it places itself in a wait state and informs the operator. If not, the initiator takes the first step of the job and goes to the next process, Step initiation.
- **Step initiation:** This process will check the COND parameter of the Exec as well as the JOB statement.
- **Allocation:** Next comes the allocation process (or Allocation routines), which allocates the required devices for all DD statements in the step.
  If the Allocation routines are successful, Program execution begins.
- **Program execution:** The program to be executed is frequently written by the user (it can also be written by a vendor) and it is normally the most time consuming part of a step's entire process. Sysin records saved during reading process will be read in by the program. Sysout (print-line) records, if any, will be saved on the spool pack for later printing or viewing.
- **Step Termination:** The step termination process is entered when the program execution terminates normally or abnormally (ABEND). This process deallocates all devices allocated in the allocation process. Devices containing "passed datasets" will not be de-allocated.
  Performs disposition processing. It attempts to satisfy what is requested in the normal or abnormal field of the DISP parameter (KEEP,DELETE,CATLG), or pass the dataset).

Verifies that all datasets opened were closed during the program's execution and closes those that were not. If there is another step in the job, the entire Step initiation/Allocation/Program Execution/Step Termination sequence is repeated.

- **Job Termination:** When the last step in the job has been executed, the Job Termination process is given control. This process is responsible for tying up loose ends. For example, when a dataset is passed and not received, Job termination will determine the ultimate status of passed data set.

2 JCL Syntax

2.1 Syntax Rules

All JCL statements begin with two slashes, //, in the first two positions (except /*). All positions of a line, from 1 to 71 (included), can be used for coding a JCL statement. Position 72 is used (rarely) for imbedded comment continuation, and positions 73 through 80 are used for numbering purposes.

With the exception of the null (//), delimiter (/*), comment (//*), all other statements follow the same general format:

//name operation parameter1,parameter2 [comment]

- **Name:** Every JCL statement can or must have a name. The name should not exceed 8 characters
- **Operation:** The operation field follows the name field and specifies the statement's function. Delimiter, comment and null statements do not have an operation field. E.g. JOB, EXEC, DD, PROC, PEND, or OUTPUT. One or more blanks must follow the operation
- **Parameter:** The parameter field consists of one or more parameters, separated by commas. No imbedded blanks between parameters are permitted
  Parameters are broadly classified into 2 categories viz. Positional and Keyword
  A positional parameter is identified by its position relative to other parameters in the operand field

***Rule 1*:** All positional parameters are coded first in the operand field and in their proper sequence.

E.g.: p1,p2,p3
p1,p3,p2 illegal because they are not in sequence.

***Rule 2*:** A keyword parameter is identified by a keyword followed by an equal sign (=) and variable information. A keyword parameter follows positional parameter and can be coded in any order.

E.g. : p1,p2,p3,k1=,k2=,k3=
p1,p2,p2,k3=,k2=,k1=
Both are valid.

***Rule 3*:** The absence of positional parameter is denoted by a comma (,) coded in its place, except when the last or remainder of the positional parameter is not present. The placeholder commas do not need to be coded in this case.

E.g.: p1,,p3,k1=,k2=,k3= place holder comma required.
p1,k1=,k2=,k3= place holder comma not required.

***Rule 4*:** Both positional parameters and variable information for keyword parameters may be composed of subparameters. The subparameters may be either positional or keyword. Subparameters must be coded as a list. The list must be enclosed in parentheses unless only one subparameter is coded. When only one subparameter is coded the parentheses are optional.

e.g. : p1,(sp1,sp2),p3,k1=(sk1,sk2),k2=,k3=
p1,(sp1),p3,k1=(sk1),k2=,k3=
Or p1,sp1,p3,k1=sk1,k2=,k3=

- **Comments**: are separated from the parameters by a blank. The comment field begins in the position after the space that marks the end of parameters field and ends in column 71. MVS ignores what you code here.

Rules for Continuation: The JCL statement can be continued in a simple way. The statement must be interrupted at a comma. This means that the last valid character of the line must be a comma followed by at least one blank. Then the statement can be continued into the next line by coding two slashes at the beginning of the line and continuing the parameter field starting anywhere between positions 4 and 16 (4 and 16 included). Note that, the comma that indicates continuation, is not an extraneous character but part of the statement.

E.g. 2.1 Examples of valid continuation

```
//DD1      DD   DSN=DA0001T.EMPFILE,DISP=(NEW,CATALG,DELETE),
//            UNIT=SYSDA,SPACE=(TRK,(5,1)),
//            DCB=(LRECL=80,RECFM=FB,BLKSIZE=800)
```

3 Job Statement

A JOB statement must be at the beginning of every job submitted to the system for execution. A JOB statement must have a name. The absence of a job name will result in a JCL error. The JOB statement identifies job to the O/S with the job name operand. There are three possible delimiters for a job during a reading process:

- Another job statement in the input stream. It signals the end of (reading) one JOB and the beginning of (reading) another
- A null statement. Following a null statement all JCL statements except a JOB statement will be ignored
- End-of-file on the reading device, meaning there are no more statements to be read in

The format of the JOB statement is:

//jobname JOB parameters

E.g.
//DA0001TA JOB parameters

The job name cannot exceed 8 characters and is usually the userid (loginid). Loginid is generally 7 characters, so you need to add a suffix else the system prompts you to enter the character when a job is submitted to the system for execution.

When a job is submitted to the system, a job number is also assigned so that the job can be further identified. This way jobs with the same name can be uniquely identified. Jobs with the same name cannot execute simultaneously. If several jobs with the same name are submitted they execute sequentially even if additional jobs could be executing. Jobs waiting to run because of this time conflict are shown in hold status.

The rest of the JOB statement contains positional parameters followed by keyword parameter.

3.1 Accounting Information Parameter:

It is a positional parameter. If present (it normally is), it must be the first in the parameter field. It can have a maximum of 142 characters (including parentheses and commas but not apostrophes). It is used to tie the resources used by the job to the appropriate account.

//jobname JOB ([account-number][,additional-accounting-information]),parameters

The **account-number** is an alphanumeric field from 1 to 4 characters long (many installations permit the use of more than 4 characters).
Additional-accounting-information is installation dependent. Many of the fields are not very important and are not often used.

E.g.:
//DA0001TA JOB LA2719,parameters

LA2719 is the account number for the training dept. This varies from one project to another.

Remark: An installation has the option of making the account number mandatory and most installations do. If so, its absence will cause a JCL error.

If the account number is incorrectly specified, in this case its not JCL error. However when job is submitted to the system for execution, we get the message "**JOB NOT RUN"** in the SYSOUT.

3.2 Programmer's Name:

Following the accounting information parameter, another positional parameter, the programmer's name can be coded. The installation determines if this parameter is required or not. If required, it must be coded immediately after the accounting information, and its omission will cause a JCL error. The programmer's name cannot exceed 20 characters. If it contains any special characters other than a hyphen and a period in the middle of the beginning (but not at the end) of the name, the name must be enclosed in apostrophes. The apostrophes are not added to the length of the name.

If a name contains an apostrophe (e.g., D'COSTA), two apostrophes must be coded. They count as one character in the length of the name.

E.g.:
//DA0001TA JOB LA2719,Sheela,parameters

//DA0001TA JOB LA2719,'D''COSTA',parameters

Note: Accounting information and Programmer's name are the only two positional parameters in the JOB statement, what follows after that is keyword parameters.

3.3 MSGLEVEL Parameter:

This parameter specifies whether the submitted JCL and/or JCL-related messages should be shown on the job's output.

General syntax

MSGLEVEL=([jcl][,messages]) keyword parameter
jcl - 0, 1, or 2

0 - Only the JOB statement will be shown
1 - All JCL will be shown
- Instream
- Expanded cataloged procedures
- Symbolic parameter substitutions

2 - All JCL will be shown, but not expanded procedure listing.

messages – 0 or 1
0 - No messages will be shown i.e. information about step completion.
1 – All messages will be shown viz. allocation and termination messages.

The messages subparameter can be thought of as On (1) or Off (0)
The default in majority of the installations is (1,1).

Remark:
1. If the entire parameter or either of the two fields is omitted, an installation-defined default is assumed.
   MSGLEVEL=1 ---------- MSGLEVEL=(1,default)
   MSGLEVEL=(,1) ---------- MSGLEVEL=(default,1)
   Parameter omitted ---------- MSGLEVEL=(default,default)

2. If the job encounters an ABEND failure, the second field always defaults to 1 even if coded as 0.

3.4 The MSGCLASS Parameter:

This parameter assigns a sysout class to the Job log. The job log consists of what is what is known as system or JES datasets:

- JES2 or JES3 log
- JCL and its associated messages
- Allocation and Termination messages

MSGCLASS - indicates the format of output
- Specifies output class for
  Job log (collection of all operations)
  List (collection of all printed output like compiled listing)

General Syntax

MSGCLASS=class keyword parameter

class - A character from A to Z or a number from 0 to 9 (in all 36 classes)

MSGLEVEL parameter indicates whether or not one wishes to print the JCL statements and allocation messages. The MSGLEVEL can save paper. After a job is debugged, there may be no need to print all the JCL and allocation messages each time it runs. To reduce printing to a minimum, one may wish to MSGLEVEL=(0,0).

All datasets to be printed must have a class. This is normally called the ***output*** class, ***sysout*** class or ***message*** class. Sysout datasets created by the executing programs are assigned a class by the SYSOUT DD statement. After the job terminates, the sysout datasets, which are saved on the spool pack, will be selected and printed by a JES2 or JES3 component called a printer. There are several printers available, and each one is assigned one or more sysout classes (from 1 to 36). The sysout class schedules a data

set to a printer in a similar way as a job class schedules a job to an initiator. The sysout class can be thought of as a print-scheduling class.

Remark: If the MSGCLASS parameter is omitted, an installation-defined default will be used.

3.5 The Class Parameter:

This parameter assigns a class to a job.

General Syntax

CLASS=jobclass keyword parameter
jobclass – A letter from A to Z or a number from 0 to 9 (in all 36 classes)

The job class affects job's processing in these ways:

- When job is submitted, it is placed in an input queue where it waits to be executed. Queues can be thought of as waiting lines for jobs. Each job class has its own input queue
- Job waits in the input queue until it is selected by an initiator to be processed. Each initiator is set to a list of job classes that it can select from

Simply put, Jobclass identifies the nature of the job:

- Short running or long running
- Resource utilization

Each installation group jobs that have like characteristics into classes. By segregating jobs with similar characteristics, an installation can maintain a good mix of the jobs running at a given moment. This maintains system throughput and efficient use of resources.

For e.g. suppose the default CLASS is A

This Job statement
//DA0001TA JOB LA2719,PCS,MSGCLASS=A,MSGLEVEL=(1,1)

is equivalent to

//DA0001TA JOB LA2719,PCS,MSGCLASS=A,MSGLEVEL=(1,1),CLASS=A

Remark: Frequently, installations develop a testing class structure that favors short-running jobs with minimal resource requirements and penalize long-running jobs with heavy resource demands. This is achieved by assigning the class used by trivial jobs to many initiators and class used by heavy jobs to a few. To keep people honest, the CLASS parameter in a testing environment is often tied to several other parameters such as TIME, PRTY, REGION, etc. For example, a job coding CLASS=A can be given TIME=(0,5), PRTY=6. Note that the values assigned to these parameters are not shown in the output. However, if any of these parameters were coded in the JOB statement, they would be ignored.
Most installation assigns a default job class if the CLASS parameter is omitted.

3.6 The PRTY Parameter

This parameter determines the scheduling priority of a job in relation to other jobs in the job input queue of the same class.

General Syntax

PRTY=priorty - keyword parameter
priorty – a number from 0 to 15 for JES2 or 0 to 14 for JES3

The **PRTY** parameter is used to define the job's input class selection priority:

- The higher the number, the better (greater) the priority
- The PRTY parameter simply controls the job's position in the input queue. It has no affect on the job's performance
- Jobs with higher priorities will be selected before job's will lower priority
- A job's priority does not affect its performance. Once the job is selected for execution, the priority function is finished
- Two jobs having same job class and same priority will be executed in sequence
- It is meaningless to compare the PRTY parameter of two jobs belonging to different classes

Remark: This parameter is of seldom use in a testing environment. Since high priority would be used by practically all users negating the very purpose the parameter. Therefore, in most installation the PRTY, whether coded or not, will default to an installation-defined value or will be supplied by the CLASS parameter.

3.7 The TIME Parameter

This parameter specifies the total amount of CPU time that all steps in a job can use collectively.

General Syntax

TIME=([minutes][,seconds] | [1440]) keyword parameter

minutes - a number from 1 to 1439
seconds - a number from 1 to 59
1440 - The job will not be timed for CPU. Note that TIME=1440 is rarely used, and most installation disallow its use in a testing environment. TIME=1440 should be used by an on-line system like CICS OR ADS/O.

When the TIME parameter is omitted, an installation-defined default will be used. This default is usually very high and unlikely to cause an S322 ABEND failure unless the program goes into an endless loop.

If the TIME parameter is also coded in the JOB statement, both will be in effect and either can cause a S322 ABEND failure. It is not advisable to use them both.

CPU time is the amount of time that the computer devoted to the job after it was selected for processing. It is not the amount of time it was in the machine.

TIME parameter puts an upper limit on the amount of CPU time that a job may use.

E.g.: TIME=(3,20). All the steps in the job are allowed collectively 3 minutes and 20 seconds of CPU time. If this amount is exceeded, the result will be a S322 ABEND failure.

If the TIME parameter is coded using only minutes, seconds defaults to zero. For example, TIME=5 is the same as TIME=(5,0).

If the TIME Parameter is coded using only seconds, minutes defaults to zero. For example, TIME=(,6) is the same as TIME=(0,6).

The TIME parameter is intended almost exclusively for a testing environment and should be coded to preempt the program going into CPU loop.

The TIME parameter can also be supplied by the CLASS parameter. When the TIME parameter is omitted and the CLASS parameter does not supply it, the job will not be timed for CPU time. However each step will be individually timed (TIME parameter at EXEC statement or its installation-defined default), unless it contains TIME=1440.

Remark: It is possible for a job to get more CPU time than that is specified in the TIME parameter by a maximum 10.5 seconds. This is due to the fact that the system checks for violations every 10.5 seconds.

3.8 The REGION Parameter

This parameter specifies the limit of available storage for each of the steps in the job within the job's address space. i.e., the amount of storage the job is allocated. In other words, it specifies the amount of storage needed by the step (within the job) with the highest storage requirements.

General Syntax

REGION=value{K|M} keyword parameter

value – 1 to 2096128 if K (1024 bytes) is used. It should be an even number. If an odd number is used it will be rounded off to the next higher even number.

value – 1 to 2047 if M (1024K or 1048576 bytes) is used. M is not available to MVS/SP, only to MVS/XA and MVS/ESA.

When a job is selected by an initiator for execution, it is given an address space of 16 MB (minus what MVS/SP uses). In case of MVS/XA, job is given an address space of 2GB. And all of it is available to the job's steps. However a step normally requires only a small fraction of this huge storage, below the 16M line. An ordinary COBOL or any other language program seldom needs more than 1000K. This is normally what the value in the REGION parameter represents in the installations. Few jobs like

CICS, IMS, DB2 need storage above the 16M line. An ordinary batch job seldom has such high requirements and, as a result confined to storage below the 16M line. Storage availability below this line varies in different installations, but is generally around 8MB in MVS/SP and around 9 MB in MVS/XA. Storage above the 16M line can be acquired by coding a value higher than 16M. However, it may be restricted by the installation to only those jobs that need it.

E.g. 1.
Assume REGION=1000K were coded in the JOB statement. All the steps in the job are limited to this value. If more storage is needed, the usual result is S878 or S80A or S804 ABEND failure. If one of these failures occurs, the user must increase the value in the REGION parameter.

E.g. 2.
REGION=10M
When the amount of storage requested in the REGION parameter is higher than the address space can provide, an S822 ABEND failure will result.

Note that for job run under MVS/SP, the entire address space is limited to 16M, of which usually less than 8M is available to the user. In case of MVS/XA, the entire address space is limited to 2G, of which usually around 9M is available to the user.

E.g. 3.
REGION=0K (or 0M) is coded the entire address space except for those areas used by MVS/SP (or MVS/XA) is available.

3.9 The ADDRSPC Parameter

This parameter specifies if the job will use real or virtual storage.

General syntax

ADDRSPC={VIRT|REAL}
VIRT – The REGION will be virtual storage and is the default
REAL – The REGION will be real storage.

Remark: This is the rarely used parameter because of the default. Note that ADDRSPC=REAL is a parameter that is disallowed in practically all installation because it can cause serious performance problems for other jobs.

3.10 The NOTIFY Parameter

This parameter informs a TSO user when his or his job terminates.

General Syntax

NOTIFY=userid keyword parameter
userid – A name from 1 to 7 characters, identifying a valid TSO user.

E.g. NOTIFY=DA0001T

If coded, a message will appear on the user's TSO terminal indicating if the job abended or got a JCL error. If the job terminates while the user was logged off, the message will appear when the user logs on. If the NOTIFY parameter is omitted, no message will appear when the job terminates.

Remark: You can also code NOTIFY=&SYSUID instead of your userid.

3.11 The RESTART Parameter

This parameter requests that a job begin its execution with a step other than the first one.

General Syntax

RESTART={stepname|procexec.stepname| *} keyword parameter

stepname – The name of the step where execution is to begin.

procexec.stepname – The name of the EXEC statement invoking a procedure and the name of the step within the procedure where execution is to begin

Note: Procedures will be covered on day 3.

* - Indicates that execution of the job is to begin with the first step and is the default.

Things to avoid:

- Duplicate names for EXEC statements invoking procedure. If RESTART=procsexec.stepname is used, the first procexec found will be used
- Duplicate stepnames within procedure. If RESTART=procsexec.stepname is used, the first stepname within procexec found will be used
- Duplicate stepnames. If RESTART=stepname is used, the first stepname found will be used
- EXEC statements (invoking procedures or any step) without names. No restart is possible

E.g. If the compile, link and run steps are given in one JCL and subsequently the execution has to begin from the run step we can give

```
000100 //DA0001TC  JOB  LA2719,'SHEELA',NOTIFY=DA0001T,
 000110 //     MSGCLASS=X,TIME=(0,1),RESTART=COBRUN
 000112 //*************************************************************
 000120 //* STEP TO COMPILE A PROGRAM
 000130 //* COMPILER PROGRAM NAME - IKFCBL00
 000140 //* LIBRARY NAME - SYS1.COBCOMP
 000150 //* SYSLIN - OUTPUT FILE NAME
 000160 //* SYSIN  - INPUT FILE NAME (I.E. COBOL PROGRAM NAME)
 000170 //* SYSUT1,2,3, -  TEMPORARY FILES REQUIRED BY COBOL COMPILER
 000180 //*************************************************************
 000200 //COB      EXEC PGM=IKFCBL00,REGION=1024K,
 000210 //       PARM='NOTRUNC,NODYNAM,LIB,SIZE=4096K,BUF=116K,APOST,NORES'
                          000400 //SYSLIB      DD  DSN=SYS1.COBCOMP,DISP=SHR
 000500 //SYSPRINT DD SYSOUT=*
 000600 //SYSLIN      DD DSN=&&TEMP,DISP=(NEW,PASS),
 000700 //               UNIT=SYSALLDA,SPACE=(TRK,(40,40))
 000710 //SYSUT1  DD  UNIT=SYSALLDA,SPACE=(TRK,(6,1))
 000800 //SYSUT2  DD  UNIT=SYSALLDA,SPACE=(CYL,(6,1))
 000900 //SYSUT3  DD  UNIT=SYSALLDA,SPACE=(CYL,(6,1))
 000910 //SYSUT4  DD  UNIT=SYSALLDA,SPACE=(CYL,(6,1))
 001000 //SYSIN   DD  DSN=DA0001T.SHEELA.COBOL(PRG1),DISP=SHR
 001100 //*************************************************
 001110 //* STEP TO LINK THE COBOL PROGRAM
 001120 //* LINKER PROGRAM NAME - IEWL
 001130 //* LIBRARY NAME - SYS1.COBLIB
 001140 //* SYSLMOD - OUTPUT DATASET NAME
 001150 //* SYSLIN - INPUT DATASET NAME
 001160 //*************************************************
 001200 //LKED  EXEC  PGM=IEWL,PARM='LIST,XREF,LET,MAP',
 001300 //  REGION=4096K,COND=(0,LT,COB)
 001400 //SYSLIN  DD  DSN=&&TEMP,DISP=(OLD,DELETE)
 001500 //SYSLIB  DD  DSN=SYS1.COBLIB,DISP=SHR
 001600 //SYSLMOD DD  DSN=DA0001T.SHEELA.LOADLIB(PRG1),
 001610 //  DISP=SHR,UNIT=SYSALLDA
 001800 //SYSUT1  DD  UNIT=SYSALLDA,SPACE=(1024,(50,20))
 001900 //SYSPRINT DD SYSOUT=*
 002000 //*
 003000//*STEP TO RUN COMPILED COBOL PROGRAM
 000500 //COBRUN  EXEC PGM=PRG1
 000600 //STEPLIB DD DSN=DA0001T.SHEELA.LOADLIB,DISP=SHR
 000700 //SYSPRINT DD SYSOUT=*
 000810 //INF1    DD DSN=DA0001T.EMPDATA,DISP=SHR
 000900 //OTF1   DD DSN=DA0001T.L3,DISP=(NEW,CATLG,DELETE),
 001000 //       UNIT=SYSDA,SPACE=(TRK,(1,1)),
 001100 //       DCB=(LRECL=80,RECFM=FB,BLKSIZE=800,DSORG=PS)
 001200 //SYSOUT DD SYSOUT=*
```

3.12 The TYPRUN Parameter

This parameter requests special processing for the job.

General Syntax

TYPRUN={HOLD|JCLHOLD|SCAN|COPY} keyword parameter

HOLD - Job will held (and not executed temporarily) until the operator uses a command to release. A job will be held in the input queue only if syntactically correct

JCLHOLD (JES2 only) – Job will held (and not executed) until the operator uses a command to release it. Note the job will be held in queue even if it is syntactically incorrect. This option is rarely used.

SCAN – Job will be scanned for all syntactical JCL errors but will not execute.

COPY (JES2 only)- Job will be printed. No execution and no syntax checking takes place. This option is also rarely used.

The following JOB statement illustrates the use of the parameters relevant to the JOB statement:

```
//da0001t JOB LA719,pai,MSGCLASS=A,MSGLEVEL=(1,1),PRTY=5,
//        CLASS=B,REGION=0M,TIME=(0,1),NOTIFY=da0001t
```

4 The EXEC Statement

An EXEC statement identifies a step during the reading process when a job is submitted to the system. When an EXEC is found, the system accepts all JCL statements that follow as belonging to the step, until a delimiter is found. There are four possible delimiters for a step during the reading process:

- Another EXEC statement in the input stream. It signals the end of (reading) one step and the beginning of (reading) of another
- A JOB statement
- A null statement i.e. //. All JCL statements will be ignored except for a JOB statement
- End-of-file on the reading device, meaning there are no more statements to read

General Syntax

//[stepname] EXEC parameters
stepname is optional. When the stepname is omitted no reference can be made. A job can contain a maximum of 255 steps.

4.1 The PGM Parameter

The PGM parameter identifies the program to be executed in a step.

General Syntax

PGM=pgmname positional parameter
pgmname - Name of the program to be fetched from the load library and executed.

The program specified in PGM is always a member of library (PDS). This is commonly known as an executable program library or a load library. The EXEC statement can identify only the member. It has no parameter available to identify the library. If necessary, this must be done by using a JOBLIB or a STEPLIB DD statement.

E.g.4.1

```
//DA0001TA   JOB  LA2719,PCS,MSGLEVEL=(1,1),NOTIFY=&SYSUID
//JOBLIB      DD  DSN=DA0001T.LIB.LOADLIB,DISP=SHR
//S1          EXEC   PGM=ASSIGN1

   OR

//DA0001TA    JOB  LA2719,PCS,MSGLEVEL=(1,1),NOTIFY=&SYSUID
//S1          EXEC   PGM=ASSIGN1
//STEPLIB     DD   DSN=DA0001T.LIB.LOADLIB,DISP=SHR
```

If neither JOBLIB nor STEPLIB is coded, the system searches certain predefined libraries. They are the system default libraries. If the specified member is found, it is executed. If not found, the result is S806-04 ABEND failure.

The following keyword parameters can be specified at the EXEC statement. They are REGION, ADDRSPC, TIME, PARM and ACCT.

4.2 The REGION Parameter

This parameter specifies the limit of available storage for the step within the job's address space.

General Syntax

REGION=value{K|M} keyword parameter
value – 1 to 2096128 if K (1024 bytes) is used. It should be an even number; it will be rounded to the next higher even number.

value – 1 to 2047 if M (1024K or 1048576 bytes) is used. M is not available to MVS/SP, only to MVS/XA and MVS/ESA

If the REGION parameter is omitted, the REGION parameter in the EXEC statements within the job will be used. If it is coded in neither the JOB nor the EXEC statement, an installation-defined default will be used. The default value of most installations is between 500K and 1000K.

If the REGION parameter is coded in both the JOB and an EXEC statement within the job, the value in the JOB statement will be used.

The REGION parameter in the JOB statement is used much more often than the one in the EXEC statement. Coding the same value for all steps would have the same effect as the REGION parameter in the JOB statement.

E.g. 4.2

```
//DA0001TA   JOB  LA2719,PCS,MSGLEVEL=(1,1),NOTIFY=&SYSUID
//S1         EXEC   PGM=ASSIGN1,REGION=500K
//STEPLIB    DD  DSN=DA0001T.LIB.LOADLIB,DISP=SHR
```

4.3 The TIME Parameter

This parameter specifies the total amount of CPU time that the step is allowed to use.

General Syntax

TIME=([minutes][,seconds] | [1440]) keyword parameter

minutes - a number from 1 to 1439
seconds – a number from 1 to 59
1440- The step will not be timed for CPU. Note that TIME=1440 is rarely used, and most installation disallow its use in a testing environment. TIME=1440 should be used by an on-line system like CICS OR ADS/O.

When the TIME parameter is omitted, an installation-defined default will be used. This default is usually very high and unlikely to cause an S322 ABEND failure.

If the TIME parameter is also coded in the JOB statement, both will be in effect and either can cause a S322 ABEND failure. It is not advisable to use them both.

Remark: It is possible for a step to get more CPU time than that is specified in the TIME parameter or the default by a maximum 10.5 seconds. This is due to the fact that the system checks for violations every 10.5 seconds.

E.g. 4.3

```
//DA0001TA  JOB  LA2719,PCS,MSGLEVEL=(1,1),NOTIFY=&SYSUID
//S1            EXEC   PGM=ASSIGN1,REGION=500K,TIME=(,3)
//STEPLIB      DD  DSN=DA0001T.LIB.LOADLIB,DISP=SHR
```

4.4 The ADDRSPC Parameter

This parameter specifies if the step will use real or virtual storage.

General syntax

ADDRSPC={VIRT|REAL}
VIRT – The REGION will be virtual storage and is the default
REAL – The REGION will be real storage.

If the ADDRSPC parameter is also coded in the JOB statement, the value in the JOB will be used.

Remark: This is the rarely used parameter because of the default. Note that ADDRSPC=REAL is a parameter that is disallowed in practically all installation because it can cause serious performance problems for other jobs.

4.5 The ACCT Parameter

The parameter specifies accounting information to be used for the step as opposed to the accounting information in the JOB statement.

General Syntax

ACCT=(acctno [,additional-acct-info]) keyword parameter

acctno – The account number to be used for the step
additional-acct-info – same as in the JOB statement.

The **ACCT** parameter is seldom used, and when it is, only the account number normally appears. This is used to charge resource utilization for a step to a different account number other than the one coded in the JOB statement.

If an account number is also coded in the JOB statement, the account number in the EXEC statement will be used.

E.g 4.4

```
//DA0001TA   JOB  LA2719,PCS,MSGLEVEL=(1,1),NOTIFY=&SYSUID
 //S1        EXEC   PGM=IEFBR14,ACCT=('es0013,hr4200,iefbr14')
 //DD1       DD  DSN=DA0001T.SHEELA.EMPFILE,disp=(MOD,DELETE),
 //          SPACE=(TRK,0),UNIT=SYSDA
```

4.6 The PARM Parameter

This parameter provides a way to supply data of limited size to the executing program

General Syntax

PARM=string keyword parameter

string –A string of characters up to 100. If commas are part of the string, the entire field must be enclosed in parentheses (or apostrophes). If any portion of the string contains special characters (other than hyphen), that portion of the entire string must be enclosed in apostrophes. Note that any parentheses used count toward the maximum. Apostrophes do not.

All information after the "=" in the PARM parameter, excluding apostrophes, will be saved by the system within the step's own region. When the program begins execution by using the appropriate instructions, it can find the saved information in storage.

In COBOL, the following must be coded:

```
LINKAGE SECTION.
1     PARM.
      05   PLENGTH            PIC S9(04)    COMP.
      05   INFO               PIC X(05).
PROCEDURE DIVISION USING PARM.
0000-MAIN-PARA.
```

Note that any valid name may be used in place of PARM. The string is stored in INFO and the PLENGTH is set to the length of the string.

E.g. 4.5

```
//DA0001TA  JOB  LA2719,PCS,MSGLEVEL=(1,1),NOTIFY=&SYSUID
//S1          EXEC   PGM=ASSIGN2,PARM='G2  ',TIME=(,1)
//STEPLIB     DD     DSN=DA0001T.LIB.LOADLIB,DISP=SHR
//INFILE      DD     DSN=DA0001T.EMPFILE,DISP=SHR
```

Rules for continuation

E.g. 4.6

```
//DA0001TA  JOB  LA2719,PCS,MSGLEVEL=(1,1),NOTIFY=&SYSUID
//COB         EXEC   PGM=IKFCBL00,REGION=1024K,
//              PARM=('notrunc,nodynam,lib,size=4096k,buf=116k',
//                    'apost,nores,seq')
                      .
                      .
        OR

//DA0001TA    JOB  LA2719,PCS,MSGLEVEL=(1,1),NOTIFY=&SYSUID
//COB           EXEC   PGM=IKFCBL00,REGION=1024K,
//                PARM=(notrunc,nodynam,lib,'size=4096k','buf=116k',
//                      apost,nores,seq)
                .
                .
```

Note that an expression in quotes cannot be continued, we need to enclose the string in parentheses and field containing special characters in apostrophes.

E.g.2 PARM='29/06/00' or ('29/06/00')

E.g. 3 PARM=(A,B,C,D) or 'A,B,C,D'

The two (in Eg 3), however, are not the same. When parentheses are used, the information found by the program is (A,B,C,D). If apostrophes are used, the information found by the program is A,B,C,D.

4.7 The COND Parameter

The COND parameter can be coded in the JOB as well as the EXEC statement. It is mostly used in the EXEC statement. This is the main tool for controlling the execution of steps within a job is the COND parameter.

A Return (or Condition) code
A return code is a number between 0 and 4095, issued by an executing program just before its execution is finished. It is intended to identify an important event found (or not found) during the execution. For example, a program may issue a return code of 21 to indicate that a problematic event (such as a record is out of sequence) was detected during the execution or a return code of 0 to indicate that the execution was trouble free. The return code issued by a program is saved by the system for the duration of the job. Any subsequent step of the same job can interrogate this return code by using the COND parameter either in the JOB or EXEC statement. The result of this interrogation is to permit or bypass the execution of the step. Note that the return code is never available to a job other than the one issued it. In other words, the step that interrogates the return code must be in the same job as, and subsequent to, the step that issued it.

IBM-established conventions.
- Return code of 0 indicates a complete success
- Return code of 4 indicates a warning. The warning is benign, so a return code will normally be treated as acceptable
- Return code of 8 indicates questionable results
- Return code of 12 indicates bad results
- Return code of 16 indicates a terminal condition

4.7.1 The COND Parameter in the JOB statement.

General Syntax

COND=((code,operator) [,(code,operator)]…….) keyword parameter
code - is a number between 0 and 4095
operator – provides a comparison between a return code and the code. There are six operators: LT, LE, NE, EQ, GT, GE

There can be a maximum of eight tests in the COND parameter. Condition is evaluated from left to right and if a test is satisfied, the job stops execution at that point.

An example can best illustrate the mechanism of the COND parameter. Consider a job with five steps. Assume that none will ABEND.

```
//DA0001TA   JOB LA2719,PCS,COND=((12,LT),(8,EQ))
```

STEP1 issues a return code of 0
STEP2, if executed, issues a return code of 4
STEP3, if executed, issues a return code of 16

STEP4, if executed, issues a return code of 0
STEP5, if executed, issues a return code of 4
(Warning: This example does not adhere to conventions.)

STEP 1 is executed by default, since no previous return codes exist and hence, the COND parameter in the JOB statement will be ignored for the first step.

Before STEP2 begins execution, the system interrogates the existing return code (0), using the tests in the COND parameter and reading the test from left to right,

- Is 12 less than 0? The answer is "no". The first test of the COND parameter was not satisfied. The second test is tested.
- Is 8 equal to 0? . The answer is "no". Neither of the two tests was satisfied, and therefore, STEP2 is executed.

Before STEP3 begins execution, the system interrogates the existing return codes (0 and 4), using the tests in the same COND parameter. Since the result for return code 0 is already known, only 4 will be tested:

- Is 12 less than 4? The answer is "no". The first test of the COND parameter was not satisfied. The second test is tested.
- Is 8 equal to 4 . The answer is "no". Neither of the two tests was satisfied, and therefore, STEP3 is executed.

Before STEP4 begins execution, the system interrogates the existing return codes (0 , 4 and 16), using the tests in the same COND parameter. Since the results for return code 0 and 4 are already known, only 16 will be tested:

- Is 12 less than 16? The answer is "yes". The first test of the COND parameter was satisfied. There is no need for the second test. Executions of the job stops. STEP 4 and the remaining steps will not be executed.

A message will appear in the output:
IEF2011 DA0001TA STEP4-JOB TERMINATED BECAUSE OF CONDITION CODES.

A formula can be devised and used to code the COND parameter, if return code conventions are strictly adhered to:

```
COND=(last-good-return-code,LT)
```

Or

```
COND=(first-bad-return-code,LE)
```

Let us apply this formula to this example 0-4 is a good return code:
4 -is the last good return code….COND=(4,LT)
or
5 - is the first bad return code…..COND=(5,LE)
The two COND parameters are logically equivalent to each other, and it makes no difference, which one is used.

Exercise: Code the COND parameter, where 0 is the only good return code.

4.7.2 The COND Parameter in the EXEC statement

The COND parameter can perform a test (or multiple tests) before a step begins execution against the return (condition) codes issued by previous steps. If a test is satisfied (reading from left to right), the step will not be executed.

General Syntax

COND=((code,operator[,stepname])[,(code,operator[,stepname])]…[,EVEN|ONLY])
keyword parameter

code - is a number between 0 and 4095

operator – provides a comparison between a return code and the code. There are six operators: LT, LE, NE, EQ, GT, GE

stepname – Identifies the name of the preceding step whose return code will be interrogated. It can also appear as two names procexec.stepname where "procexec" identifies the name of the EXEC statement invoking a procedure and "stepname" the stepname within the procedure.

EVEN - requests that execution be permitted even though a previous (any previous) step has ABENDed.

ONLY - requests that execution be permitted only if a previous (any previous) step has ABENDed.

There can be a maximum of eight tests in the COND parameter. EVEN or ONLY count toward eight. Condition is evaluated from left to right and if a test is satisfied, only that step is not executed.

Remark:

- EVEN and ONLY cannot make reference to a particular step. They refer to any previous step that has ABENDed
- EVEN and ONLY are mutually exclusive
- EVEN and ONLY have no positional significance. Each can be coded anywhere in the COND parameter in relation to other tests
- Following an ABEND failure, a step cannot be executed unless it contains EVEN or ONLY in the COND parameter of its EXEC statement
- The first step will always be executed unless COND=ONLY appears in the exec statement. COND=ONLY would cause the first step to be bypassed, since no previous ABEND failures could have occurred. Any other COND parameter in the first EXEC statement will be ignored (i.e., COND=(4,LT) or COND=EVEN) or will result in JCL error (i.e., COND=(5,LT,stepname)) – since there are no previous step
- A step that is not executed issues no return code because a program responsible for issuing the return code was not even loaded into the storage. As a result no return code exists. An attempt to interrogate the return code of such a step in the COND parameter of a subsequent step will be ignored

- A step that ABEND's issues no return code because a program always issues a return code (conditionally or by default) if it reaches the end of its execution and intentionally returns control to the system. When an ABEND occurs, the program loses control instantly. And is evicted from execution by the system. As a result when a step ABEND's no return code exists (a completion code exists). An attempt to interrogate the return code of such a step in the COND parameter of a step will be ignored until it contains EVEN or ONLY

An example can best illustrate the mechanism of the COND parameter. Consider a job with five steps. This is a classroom exercise.

```
//DA0001TA    JOB LA2719,PCS,NOTIFY=&SYSUID,MSGLEVEL=(1,1)
//S1           EXEC  PGM=P1                                    (4)
//S2           EXEC  PGM=P2,COND=((0,LT,S1),EVEN)              (12)
//S3           EXEC  PGM=P3,COND=(8,LT,S2)                     (0)
//S4           EXEC   PGM=P4,COND=(4,LT)                       (8)
//S5           EXEC   PGM=P5,COND=((4,LT,S1),(0,LT,S3))        (abends)
//S6           EXEC   PGM=P6,COND=(EVEN,(0,LE,S5),)            (16)*
//S7           EXEC   PGM=P7,COND=((0,LT,S1),EVEN)             (0)
//S8           EXEC   PGM=P8,COND=((0,LT,S1),(12,LT,S3) )      (0)
//S9           EXEC   PGM=P9,COND=EVEN                         (4)
//S10          EXEC   PGM=P10,COND=ONLY                        (0)
```

If the COND parameter is coded neither at the JOB nor at the EXEC statement, the step will be executed regardless of previous return codes. However it will not be if a previous step has ABENDed.

If the COND parameter is coded in both the JOB statement as well as an EXEC statement within the JOB, both will be tested. The COND parameter of the JOB statement is tested first. If none of its tests are satisfied, then the COND parameter of the EXEC statement is tested. If a test is satisfied, none of the steps from that point on will be executed.

5 The DD statement

A DD (Data Definition) statement must appear in a step when the executing program expects to read from or write to a dataset. In other words DD statement describes the dataset.

The maximum number of DD statements in a step is 3273. The DD statement can be coded in any order and always appears after the EXEC statement with the exception of JOBLIB, JOBCAT, PROCLIB DD statement.

5.1 The DSN Parameter

The DSN (or DSNAME) parameter identifies the name of the dataset to be created or retrieved.

General Syntax

DSN=name| NULLFILE | referback keyword parameter

name - It could be a qualified name. This name consists of two or more simple name separated by periods for a maximum of 44 characters.

E.g. 1 DSN=DA0001T.PCS.EMPFILE

E.g. 2 DSN=DA0001T.PCS.COBOL(ASS1) describes a sequential dataset i.e. ASS1 is a member of a PDS/library DA0001T.PCS.COBOL

E.g. .3 DSN=&&name
A simple name preceded by two ampersands identifies a temporary dataset. Temporary because it is not retained beyond job termination.

The system generates a name with the following format:

SYSyyddd.Thhmmss.RV001.jobname.name

yyddd – date as per Julian calendar;

hhmmss – uses 24-hour clock. It is the time of JOB initiation (beginning of JOB execution)

RV001 –system provided information in reference to the reader;

jobname – as it appears in the JOB statement;

name – whatever is coded after &&.

For e.g. DSN=&&temp. The system generates the following name:
SYS03173.T090000.RV001.DA0001TA.TEMP

Remark: If the DSN name is omitted from a DD statement (except DD *, SYSOUT and DUMMY) also indicates a temporary dataset. However the system generates a name with the following format:

SYSyyddd.Thhmmss.RV001.jobname.R0000001

```
//SORTWK1     DD UNIT=SYSDA,SPACE=(TRK,(1,2),RLSE)
```

SYS00173.T100000.RV001.DA0001TA.R0000001

This form is basically used when a step requires a work dataset (a dataset created at the beginning of the step's execution and deleted at the end). Mostly used in utilities.

referback: This can have three formats:

- *.stepname.ddname – Requests that the dataset name be copied from DD statement "ddname " found in a previous step "stepname".

  E.g.: DSN=*.STEP1.DD1

- *.ddname – Requests that the dataset name be copied from a previous DD statement "ddname " found in the same step "stepname".

  E.g.: DSN=*.DD1

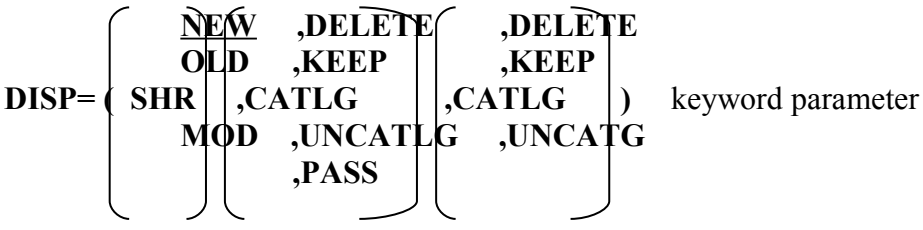
- *.procexec.stepname.ddname – Requests that the dataset name be copied from DD statement "ddname " found in a previous step "stepname" found within procedure "procexec". (name of EXEC statement invoking the procedure)

  E.g.: DSN=*.PD1.STEP1.DD1

5.2 The DISP Parameter

The DISP parameter specifies :
- If the dataset is to be created or retrieved
- How to dispose of the dataset when the step terminates (normally or abnormally)

General Syntax



DISP=(status-field,normal-disp-field,abnormal-disp-field)

The status-field: This field tells the system whether the dataset is to be created or retrieved.

- **NEW** – Indicates that the dataset will be created in this step
- **OLD** - Indicates that an existing dataset will be retrieved and demands exclusive control
- **SHR** - Indicates that an existing dataset will be retrieved. It also indicates that this dataset, if on disk, can be shared with one or more other users
- **MOD** - This sub parameter has two possible meanings:
  - Indicates that an existing dataset will be retrieved. This will be true if
    - The dataset is either cataloged or passed
    - The DD statement contains either VOL=SER or VOL=REF (a VOL VOL=REF referring to a DD statement, which is a nonspecific request for a new dataset, is not included)
  - Indicates that the dataset will be created. This is true if:
    - The DD statement contains neither VOL=SER nor VOL=REF and it describes a dataset which is neither cataloged nor passed
    - The DD statement contains VOL=REF referring to a DD statement, which is nonspecific, request for a new dataset

E.g. 1
```
//DD1   DD  DSN=DA0001T.EMPFILE,DISP=(MOD,CATLG),
//         UNIT=TAPE
```

Explanation:
1. The system assumes DA0001T.EMPFILE to be an existing dataset. Since the DD statement contains neither VOL=SER or VOL=REF, the system searches the catalog and gets volume information from the catalog entry. The volume having been found, the dataset will be treated as existing dataset.
2. Had the dataset been neither cataloged nor passed, the system would have been unable to find the volume information and MOD will default to new.

E.g. 2
```
//DD1   DD  DSN=DA0001T.EMPFILE,DISP=(MOD,CATLG),
//         UNIT=SYSDA,VOL=SER=BS3003,SPACE=(TRK,(1,2))
```

Explanation: Since VOL=SER is specified; the fate of MOD is sealed, whether or not it exists. It will be treated as OLD (with appropriate positioning). If the dataset exists on that volume no problem, however, if it does not exist the result will be S213-04 ABEND failure (i.e. dataset does not exist)

Note: When UNIT and VOL=SER is specified the system does not search the catalog to locate the dataset.

The normal disposition field: This field is used to tell the system how to dispose of the dataset when the step terminates normally (without an ABEND).

- **DELETE**: indicates that the dataset is to be deleted when the step terminates. For an existing dataset, OLD, SHR or MOD (not defaulting to NEW), the dataset will also be uncataloged, if the catalog were used while retrieving the dataset. It will only delete if the catalog were not used during the retrieval. This means that for a cataloged dataset, if you specify UNIT and VOL=SER the system does not search the catalog.

Note:

1. When a tape dataset is deleted, nothing happens. A tape dataset cannot be deleted through the DISP parameter. It is effectively deleted when the dataset is written over.

2. A VSAM cluster cannot be deleted by coding DISP=(OLD,DELETE) as it defaults to DISP=(OLD,KEEP).

3. A member of PDS cannot be deleted, as DISP applies to the entire PDS, and as result it deletes the entire PDS. Use either TSO or IEHPROGM utility.

4. The system always issues a message indicating " DELETED' or "NOT DELETED N" N indicates the reason for failing.

- KEEP – Indicates that the dataset is to be kept when the step terminates. The system takes no action and issues a message indicating the dataset was kept. Again, the system issues a message "KEPT". Note that "NOT KEPT" message does not exist.

Note: KEEP does not imply CATLG. As a result, DISP=(NEW,KEEP) should be rarely used because next time you retrieve the dataset, you need to specify UNIT and VOL=SER.

- **CATLG** – Indicates that the dataset is to be kept and an entry for it placed in the catalog when the step terminates.

- **PASS** – Indicates that an entry for the dataset (containing DSN, VOL=SER and UNIT information) be placed on a table in storage (Passed Dataset Queue). This entry is to be used in a subsequent step to "receive the passed dataset". A message will appear "PASSED".

The abnormal (or conditional) disposition field: This field is used to tell the system how to dispose of the dataset when the step terminates abnormally (ABENDs). It is required only if this disposition is different from the normal disposition.

DELETE,KEEP,CATLG, and UNCATLG have the same meaning they do in the normal disposition. Note that PASS is not permitted in the abnormal disposition field. The best example of using the abnormal disposition field is DISP=(NEW,CATLG,DELETE). If there is ABEND, the dataset is to be deleted. This eliminates future manual intervention to delete and uncatalog the dataset in order to restart.

Defaults: Some defaults in the DISP parameter are fixed and others variable.

1. If the DISP parameter is omitted, the default is always (NEW,DELETE).

```
//SYSUT1      DD  UNIT=SYSDA,SPACE=(TRK,(1,2))
//*     (NEW,DELETE)  IS THE DEFAULT
```

2. If the status is omitted, the default is always NEW
   DISP=(,CATLG) is same as DISP=(NEW,CATLG)

3. If the normal disposition field is omitted
   - If the status field is NEW, the default is DELETE
   - If the status field is OLD or SHR and the dataset name non temporary
     –If the DD statement is not receiving a passed dataset, the default is KEEP.

```
//DD1     DD DSN=DA0001T.EMPFILE,DISP=SHR
```

–If the DD statement is receiving a passed dataset, which was created during the execution of the job and was never given a permanent disposition, the default is DELETE.

```
//S1            EXEC  PGM=PR
//DD1           DD      DISP=(, PASS), DSN=USER1.PDST, UNIT=SYSDA,
//                      DCB=(BLKSIZE=23440, LRECL=80, RECFM=FB),
//                      SPACE=(TRK, (50,10),RLSE)
//S2            EXEC  PGM=PC
//DD2  DD       DISP=OLD, DSN=USER1.PDST
```

In DD2 DISP=OLD defaults to DISP=(OLD, DELETE)

–If the DD statement is receiving a passed dataset, which was created during the execution of the job but was given permanent disposition since being created, the default is KEEP.

```
//S1            EXEC        PGM=PR
//DD1           DD          DISP=(, PASS), DSN=USER1.PDST, UNIT=SYSDA,
//                          DCB=(BLKSIZE=23440, LRECL=80, RECFM=FB),
//                          SPACE=(TRK, (50,10), RLSE)
//S2            EXEC        PGM=PC, COND=(4, LT)
//DD2           DD          DISP=(OLD, CATLG), DSN=USER1.PDST
//S3            EXEC        PGM=PF, COND=(4,LT)
//DD3           DD          DISP=(OLD,PASS), DSN=USER1.PDST
//S4            EXEC        PGM=PK, COND=(4, LT)
//DD4           DD          DISP=OLD, DSN=USER1.PDST
```

In DD4 DISP=OLD defaults to DISP=(OLD,KEEP)

–If the DD statement is receiving a passed dataset, which existed before the job began execution, the default is KEEP.

```
//S1        EXEC    PGM=PR
//DD1       DD      DISP=(SHR,PASS), DSN=USER1.LONE
//S2        EXEC    PGM=PK, COND=(4,LT)
//DD2       DD      DISP=SHR,DSN=USER1.LONE
```

In DD2 DISP=SHR defaults to DISP=(SHR,KEEP)

Despite the several possible defaults for DISP=OLD or DISP=SHR their use is extremely common. When not receiving a passed data set, they always safely default to DISP=(OLD,KEEP) and DISP=(SHR,KEEP), respectively.

- If the status field is OLD or SHR and the dataset name temporary, the default is pass.

  ```
  //DD1     DD DISP=OLD,DSN=&&TEMP
  ```

  DISP=OLD defaults to DISP=(OLD,PASS) and the message will appear in the output –"INVALID DISP FIELD – PASS SUBSTITUTED"

- If the status field is MOD, which defaults to an existing data set, MOD works the same as OLD and SHR
- If the status field is MOD, which defaults to NEW, the default of the second field is DELETE

DISP=MOD can default to (MOD,KEEP), (MOD,DELETE), (MOD,PASS), and (NEW,DELETE). In view of all these possibilities, it is recommended that defaults not be practiced with MOD.

4. If the abnormal disposition field is omitted, the default is the normal disposition field.

Remark: The various fields of the DISP parameter stand for the PDS and not the member.

```
//DD1 DD DSN=USER1.LIB2(Z32),DISP=(OLD,DELETE,DELETE)
```

In the above case the PDS and the member both will be deleted. Both the PDS and the member should exist.

How a member is handled depends on whether or not it exists and on whether the program opens for input or output. A summary of all possibilities is presented below.

```
//S1    EXEC  PGM=P1
//D1    DD    DSN=DA0001T.LIB(M12),DISP=SHR
```

- **M12 EXISTS**
  - ➢In P1 if M12 is opened in I/P mode, for reading, M12 is read.
  - ➢In P1 if M12 is opened in O/P mode for writing, M12 will be replaced (not in place)

- **M12 DOES NOT EXIST**

  - ➢In P1 if M12 is opened in I/P mode for reading, ABEND (S013-18).
  - ➢In P1 if M12 is opened in O/P mode for writing; M12 will be created and written into.

5.3 The UNIT Parameter

The UNIT parameter identifies:

- The device type or device address where the volume is mounted. The volume is the one where the dataset resides (or will reside if DISP=NEW)
- The number of devices to be allocated to the dataset
- When the mount message is to be shown to the operator.

General Syntax

UNIT =(device address / generic device name / generated device name) parameter [,device count] [,DEFER] Keyword

device address – Identifies the exact device address. This notation is almost never used.

generic device name – Identifies the device type using a universal system-supplied name.

E.g.: UNIT=3390 ; UNIT=3400-5 ; UNIT=3480

generated device name – Identifies the device type using an installation-defined name.

E.g.: UNIT=SYSDA ; UNIT=DISK ; UNIT=TAPE

The generated names can be made to mean whatever an installation wishes them to mean. For example, UNIT=SYSDA can mean all 3380 devices of any density, or

single density only, or a subset of double density devices or a combination of 3380 and 3390 device. Their definition can vary from installation to installation.

Of the three, the generated device name is far the most commonly used.

device count – Specifies the number of devices to be allocated for the dataset. The limit is 59 devices. If omitted, default is 1 except when DD statement describes a disk multi-volume dataset. In such case, device count=number of volumes.

E.g. 1 UNIT=(SYSDA,5) , UNIT=(TAPE,2)

E.g. 2 UNIT=SYSDA is same as UNIT=(SYSDA,1) because of default

```
E.g. 3 //DD1     DD  DSN=DA0001T.EMPFILE,DISP=(,CATLG,DELETE),
         //          UNIT=SYSDA,VOL=SER=(BS3001,BS3002,BS3003),
         //          SPACE=(TRK,(1,2)),DCB=(LRECL=80,RECFM=FB,
        //           BLKSIZE=800)
```

In this example UNIT =SYSDA defaults to UNIT=(SYSDA,3)

Note: UNIT=(,2) can also be used if the device is being supplied by the catalog.

Default:
There is no default for device name. If it is not coded in the UNIT parameter and it is also not supplied by the catalog, the Passed dataset Queue, or VOL=REF, the result will be a JCL error . The message is

"IEF210I JOBNAME STEPNAME DDANAME –UNIT FIELD SPECIFIES INCORRECT DEVICE NAME", which is misleading. It means that the device name was needed but not coded.

Sharability Considerations

With the exception of SHR, all other status fields demand exclusive usage of a data set. This means the following:

If a job containing a DD statement (in any step) with OLD, NEW, or MOD begins execution, no other job that contains a DD statement (in any step) with the same data set name can begin execution, regardless of DISP used. It is placed in a wait state (no S522 ABEND can occur) and the operator is informed. When the last step that uses the data set with OLD, NEW, or MOD terminates, then the data set is freed and other jobs can begin execution. Note that if this step were the last one of the job, no other job can begin execution throughout the execution of the first job despite the fact that only the last step uses the data set.

If a job containing a DD statement (in any step) with a DISP of SHR begins execution, no other job that contains a DD statement (in any step) with the same data set name can begin execution, if the DISP is OLD, NEW, or MOD. It is placed in a wait state (no S522 ABEND can occur) and the operator is informed. When the last step that uses the data set terminates, then the other job can begin execution.

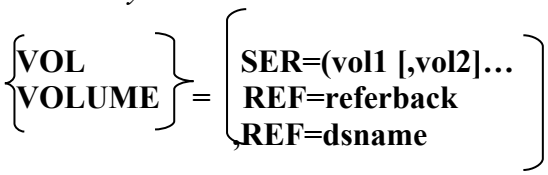
DISP=SHR should be used whenever retrieving and reading a disk data set.
DIPS=OLD should be used whenever retrieving and updating a disk data set.

For tape data sets DISP=OLD is normally used, but DISP=SHR is also acceptable. Tape us a non-sharable device. DISP=SHR can be useful if two jobs contain a DD statement each describing a data set by the same name but on different volumes.

5.4 The VOL Parameter

The main function of the VOL (or VOLUME) is to identify the volume(s) by serial number where an existing dataset resides or where a new dataset will reside.

General Syntax



SER=(vol1,vol2….) – Specifies the serial number(s) of the volume(s) to be used. The maximum number of volumes is 255.

A volume serial is a combination of alphabetic, numeric, and national characters ($ @ #) up to 6. A hyphen is also permitted. In a real (or production) environment, the number of characters is almost never less than 6.

E.g. VOL=SER=BS3001 or VOLUME=SER=BS3001
VOL=SER=(BS3013,BS3014)

The VOL parameter must be coded:

1) When retrieving a dataset which is neither cataloged nor passed.
2) When retrieving a dataset, which is cataloged, but the catalog must not be used.
3) When creating a dataset which must reside on a particular volume.

REF=referback
Referback – This can have three formats:

- *.stepname.ddname - Requests that the volume be the same as for DD statement "ddname" found in the previous step "stepname"

  VOL=REF=*.STEP2.DD1

- *.ddname - Requests that the volume be the same as for previous DD statement "ddname" found in the same step "stepname".

  VOL=REF=*.DD1

- *.procexec.stepname.ddname - Requests that the volume be the same as for DD statement "ddname" found in the previous step "stepname" found within a procedure "procexec" (name of EXEC statement invoking the procedure

VOL=REF=*.PR1.STEP2.DD1

Referbacks are not encouraged. They should be used only when they are necessary. A referback with a "stepname" will cause a JCL error if the referenced step does not execute. Such referbacks must be avoided where restart is required.

REF=dsname – Requests that the volume be the same as the one where dataset "dsname" resides on. The dataset must be cataloged or passed. The dataset does not even have to exist, as long as it is cataloged or passed. The name of the referenced dataset need not appear anywhere else in the job.

E.g: VOL=REF=DA0001T.EMPFILE

Remark: When VOL=REF (referback or dsname) is used, the system supplies the volume as well as the unit information. Therefore, the UNIT parameter is usually unnecessary.

Default:
There is no default for VOL=SER or VOL=REF. However if both are omitted, no JCL error results. Instead the meaning of DD statement changes. For example when retrieving and VOL=SER or VOL=REF is coded, the catalog will not be used. If neither is coded, the catalog will be used.

5.5 The SPACE Parameter

The SPACE parameter must be included in a DD statement when:

- A new disk dataset is created.
- An old dataset needs to alter its entitlement to additional space. i.e., Request additional disk space for an old dataset when available space is exhausted.
- An old disk dataset must free up all unused space.

General Syntax

SPACE={**TRK,** / **CYL,** / **blksize,**} **(prim-alloc [,sec-alloc] [,directory]) [,RLSE])**

TRK – Requests that space be allocated in tracks.
CYL – Requests that space be allocated in cylinders.
blksize – Specifies the average blocksize of the dataset. The system will translate it to tracks.

- **Prim-alloc- Primary allocation or Primary quantity:**

It identifies the number of tracks (if TRK is coded) or cylinders (if CYL is coded) or the number of blocks (if blksize is coded) that must be allocated during the allocation process for a new dataset before the step begins execution. The system will allocate the requested space in one extent. If this is not possible (and CONTIG is not coded),

two extents will be used, then three and so on up to five extents. If as many as five extents still cannot satisfy the request, the result will be an allocation JCL error:

IEF257I jobname stepname ddname –SPACE REQUESTED NOT AVAILABLE.

If the request is nonspecific (no VOL=SER or VOL=REF), needing a storage volume, the JCL error message will be different:

IEF257I jobname stepname ddname –INSUFFICIENT SPACE ON STORAGE VOLUMES.

Remark: The system will always allocate the primary quantity in the least number of extents possible on a single volume. The primary quantity cannot be split over multiple volumes. The primary allocation cannot be omitted (coding 0 is allowed). It is ignored if the dataset is old.

E.g. 1 SPACE=(TRK,3)
E.g. 2 SPACE=(CYL,4)
E.g. 3 SPACE=(23440,100)
E.g. 4 SPACE=(TRK,0)

The primary allocation cannot be omitted (coding 0 is allowed). It is ignored if the dataset is old.

- **sec-alloc - Secondary allocation or secondary quantity**:

It identifies the number of tracks (if TRK is coded) or cylinders (if CYL is coded) or the number of blocks (if blksize is coded) that are to be allocated when all available space is exhausted while writing to a dataset. The system will allocate the secondary quantity in the least number of extents possible, and just like the primary quantity; it can be given in as many as five extents, if necessary

The system will always supply the specified secondary allocation when one is needed unless one of the two events occurs:

- The allocated volume does not have enough space to satisfy the secondary allocation and no other volumes are allocated.

- The needed secondary allocation, if granted, will cause the dataset to exceed 16 extents on the volumes and no other volumes are allocated.

If either of these two conditions arises, the result will be a SB37-04 ABEND failure (normally for a sequential dataset). For a PDS, the ABEND can also be SE37-04.

Please note that a PDS is confined to a single volume, while a sequential dataset can extend into a maximum of 59 volumes. The 16-extent-per-volume limit for a dataset is system-supplied and cannot be altered.

The secondary allocation is optional. If omitted, defaults to 0. When no secondary allocation is coded and the primary allocation is exhausted, the result is an SD37-04 ABEND failure.

Directory – Specifies the number of directory blocks (256 bytes each) to be assigned to the directory of a PDS.
The directory quantity, if not coded, defaults to zero; therefore, the directory quantity must be specified for a new PDS. If it is, not S013-14 ABEND failure will occur if an attempt is made to add the first member to a PDS.

Remark: The directory quantity is taken away from the beginning of the primary allocation if TRK or CYL is coded in the SPACE parameter. When blksize is coded, the system adds the directory blocks to the data blocks and then computes the amount of primary space.

E.g. 1 SPACE=(TRK,(20,5,5)) OR SPACE=(TRK,(20,,5)) if no secondary
E.g. 2 SPACE =(CYL,(20,5,5)) OR SPACE =(CYL,(20,,5)) if no secondary
E.g. 3 SPACE =(23440,(200,50,5)) OR SPACE =(23440,(200,,5)) if no secondary

RLSE –Requests that any unused space be freed when the dataset is closed. This works for both new and old datasets, provided they were opened for output. Space will be released on the boundary used in the SPACE parameter. If tracks (or cylinders) were allocated, unused tracks (or cylinders), will be released.

Remark: Using RLSE is highly recommended for datasets not intended for future expansions. Temporary datasets are ideal candidates. For datasets that expand in future runs, RLSE can result in a larger number of extents, and, possibly, a premature SB37-04 ABEND failure. RLSE will be ignored if the dataset is opened by another user (or shared by another job) or the step ABEND's.

E.g.: SPACE=(TRK,(5,1),RLSE)

5.6 The LABEL Parameter

The LABEL parameter can specify:

- The sequence of a tape dataset on a volume.
- The type of label of the dataset.

General Syntax

LABEL=([seq-no][,type]) keyword parameter

seq-no – Identifies the sequence number of the dataset on a tape volume. 1 to 4 digits. If omitted, it defaults to 1. If 0 is coded, it defaults to 1. Maximum: 9999

E.g. LABEL=3

type – Identifies the type of label for the dataset.

There are many types of labels. To name a few, which are important from project perspective.

SL – Indicates IBM standard label. If the sub parameter is omitted, SL is the default.

NL – Indicates no labels are used. NL is not commonly used. Normally, NL is used for a tape coming from or going to another installation, which has no SL capabilities.

BLP – Bypass Label Processing: Indicates that labels will not be recognized and will be treated as ordinary files. BLP is used as a last resort when neither SL nor NL can accomplish what is required.

Label Verification: When retrieving an SL tape dataset, both the volume serial and the dataset name will be verified. When creating an SL tape dataset with VOL=SER or VOL=REF, only the volume serial will be verified.

When retrieving an NL tape dataset, neither the volume serial nor dataset name can be verified. However, only an NL tape volume can be mounted. An SL volume will be rejected.

Defaults: If omitted, the LABEL parameter defaults to (1,SL). There are four ways to supply the same information.

- Omit the LABEL parameter
- Code LABEL=(,SL) 1 is the default
- Code LABEL=1 SL is the default

SL

VOL	HDR1	HDR2	TM	SL DATA SET # 1	TM	EOF1	EOF2	TM		TM

NL

NL DATASET #1	TM	NL DATASET #2	TM		TM	

TM – Tape Mark

5.7 The DCB Parameter

The DCB parameter specifies values to be used to complete the Data Control Block (DCB) when a dataset is opened. A DCB is constructed by the language processor (compiler or assembler), based on the appropriate instructions of the language being used, and resides inside the code of the program. The compiler collects this information and defaults from various parts of the program (For e.g. In COBOL, RECORD CONTAINS 80 CHARACTERS; BLOCK CONTAINS 10 RECORDS and so on) and constructs the DCB. Note that the DCB exists only for non-VSAM datasets and is checked by the OPEN routines (for input or output). Certain values must be

"hard-coded" in the DCB by the program. Others can be left out, giving the user the option of supplying these values via the DCB parameter (as well as other means).

There are three suppliers of DCB information:

- Values supplied by the program, referred to as hard-coded. When a value is hard-coded, it cannot be changed unless the program is changed

- Values coded in the DCB parameter of the DD statement. These values will be ignored if they are already hard-coded

- Values from the standard label of the dataset. The values supplied by the label are limited to: BLKSIZE,LRECL, RECFM, DSORG etc. Values from the label will not be used if they are hard-coded inside the program or coded in the DCB parameter of the DD statement

General Syntax

DCB=([referback] | [model][,subparameter],…… keyword parameter

referback – This can have three formats:

*.stepname.ddname - Requests that the DCB parameter be copied from the DD statement "ddname" found in the previous step "stepname".
DCB=*.STEP2.DD1

*.ddname - Requests that the DCB parameter be copied from a previous DD statement "ddname" found in the same step "stepname".
DCB=*.DD1

*.procexec.stepname.ddname - Requests that the DCB parameter be copied from DD statement "ddname" found in the previous step "stepname" found within a procedure "procexec" (name of EXEC statement invoking the procedure).
DCB=*.PR1.STEP2.DD1

Remark: The DCB referback copies the DCB parameter as opposed to the DSN and VOL=REF referbacks which acquire the dataset name and the VOL=SER respectively, whether or not the DSN and VOL parameters are present in the referenced DD statement. If the DCB referback refers to a DD statement, which contains no DCB, nothing is copied and no message appears.

Model – specifies the name of the dataset which:

- Must be cataloged. If it is not, the result will be a JCL error: IEF2121 jobname stepname ddname –DATASET NOT FOUND

- Must be on disk (Tapes not allowed)

- Must reside on a volume that is accessible (online)

This dataset is called a model DSCB. The DCB information from the label of the model is extracted and can be used.

E.g. 1. DCB=DA0001T.EMPFILE

E.g. 2. In case you want to override some of the subparameters, the overriding subparameters must follow the DCSB model dataset name.

DCB=(DA0001T.EMPFILE,LRECL=100,BLKSIZE=800)

Models are generally used, during the creations of GDG's and dummying the PDS.

- Subparameters: There is vast number of subparameters, the great majority of which are seldom or never used

  – **BLKSIZE**: Specifies the size of the block (also known as the physical record). For RECFM=FB, the blocksize must be multiple of the logical record length, and it identifies the exact size of the block. For RECFM=VB, the blocksize can be any value up to the limit but atleast 4 bytes larger than the logical record length. For RECFM=U, the blocksize can be any value up to the limit

Remark: There is no default for BLKSIZE. Coding BLKSIZE=0, the system will compute the optimum blocksize based on the device type.

E.g. DCB=BLKSIZE=800

  – **LRECL** – Specifies the size of the logical record. The maximum size is 32,760, and it cannot be larger than blocksize, unless RECFM=VBS is used.

E.g. DCB=(LRECL=80,BLKSIZE=800)

  – **RECFM:** Specifies the record format. There are several values (or combinations of values) that can be coded.

- **F** - All blocks and all logical records are fixed in size.

- **V** - Blocks as well as logical records are of variable size. The first 4 bytes of each block (and logical record) describe its length.

- **B** – One or more logical records reside in each block. B cannot be coded alone. It is used in conjunction with F or V. For example FB or VB.

- **U** – Blocks are of variable size. There are no logical records. Mainly used with Load Library.

- **S**- For fixed-size records, it indicates that no short blocks are permitted anywhere but the end of the data. For variable-size records, it indicates that a logical record can span more than one block. S cannot be coded alone. It must follow F, V, FB or VB.

- **A** – Indicates that the first character of each record is an ANSI control character to be used for printer carriage control. A cannot be coded alone. It must follow F, V, FB, VB or U.

E.g. DCB=(LRECL=80,RECFM=FB,BLKSIZE=800)

If RECFM is not supplied through any means, U is the default.

- **DEN:** Identifies the density of the tape. DEN=3(or 4) indicates 1600 (or 6250) BPI density.

- **BUFNO:** Identifies the number of buffers to be allocated in virtual storage by the OPEN routines, which will contain the blocks to be read in or written out. If omitted

Default is 5. The maximum is 255. Coding for BUFNO a number greater than 5 may require that the REGION parameter be increased. However, default of 5 is more than adequate in most cases of dataset processing.

E.g. //INFILE DD DSN=DA00011.EMPFILE,DISP=SHR,DCB=BUFNO=8

- **DSORG:** Identifies the organization of the datasets

- **PS** – Specifies physical sequential organization. Mostly QSAM and sometimes BSAM

- **PO** - Specifies partitioned organization (or BPAM)

- **DA**- Specifies direct organization (or BDAM)

- **IS** – Specifies indexed sequential organization (or ISAM)

It is important to understand which of these often-used parameters are normally hard-coded and which are not:

- **BLKSIZE** - Seldom hard-coded. The BLKSIZE is unrelated to the logic of the program and hard-coding its value would cause unnecessary changes whenever the BLKSIZE is changed. In COBOL, BLOCK CONTAINS 0 RECORDS must be coded to avoid hard-coding the BLKSIZE. Omitting this clause will cause a default of 1 to be used. The result will be a hard-coded BLKSIZE is equal to LRECL. Many installation standards disallow hard-coding the BLKSIZE for sequential and partitioned datasets

- **LRECL** – Frequently hard-coded. The logic of any ordinary program is dependent on the LRECL and, as a result, the LRECL cannot be changed without changing the logic of the program. Many high-level languages like COBOL always hard-code the LRECL

- **RECFM** – Frequently hard-coded. The logic of any ordinary program is dependent on the RECFM and, as a result, the RECFM cannot be changed without changing the logic of the program. Many high-level languages like COBOL always hard-code the RECFM

5.8 Instream Data

The input stream submitted to the system for execution consists of two possible parts:

- JCL mandatory part of the input stream
- Data mixed in with JCL in the input stream. This data is known as sysin data or input stream data. It is optional part of the input stream and always has a logical record length of 80. Any records encountered in the input stream which are not JCL statements will be treated as sysin data.

Sysin data must be preceded by a DD statement such as:

```
//DDNAME      DD  *
data
/*
```

Sysin data encountered by JES2 or JES3 following a DD * statement will be saved on the SPOOL volume for future use. This is known as input spooling The sysin is delimited (the spooling stops) by:

- A /* (delimiter) statement found
- A valid JCL statement
- An end-of-file condition on reading device

The asterisk (*) is a positional parameter. The DD * is a special statement which is under complete JES2 or JES3 control.

SYSIN is a very common ddname used by many vendor-written programs to pass control information to the utility. E.g. SORT,IEBGENER,IDCAMS utilities.

In user written programs, if you use COBOL ACCEPT statement, then in the run JCL one of the DD statements will be SYSIN DD statement.

```
//SYSIN   DD *
1234
/*
```

Remark: If sysin data is not preceded by DD *, the system will generate a statement and place it in front of the sysin data .

E.g.

```
 //DA0001TA JOB LA2719,…..
//S1         EXEC PGM=ASS1
//STEPLIB    DD …
1234
//DD1        DD …
```

Is equivalent to

```
//DA0001TA JOB LA2719,…..
//S1          EXEC PGM=ASS1
//STEPLIB   DD …
//SYSIN      DD  *    (generated statement)
1234
//DD1         DD …
```

Note: A line with blanks is the most common offender. It is invisible to the user but it will be treated as data by the system this may or may not cause problem. Let us look at the following example.

```
//DA0001TA JOB LA2719,…..
//S1   EXEC PGM=ASS1

//STEPLIB   DD …
//SYSIN  DD *
1234
//DD1  DD …
```

The system will interpret the above JCL in the following way:

```
//DA0001TA JOB LA2719,…..
//S1   EXEC PGM=ASS1
//SYSIN      DD *

//STEPLIB   DD …
//SYSIN  DD *    (generated statement)
1234
//DD1  DD …
```

Conclusion: If there are two or more DD statements by the same name in the same step, this is not an error condition. When the program opens for SYSIN the first of the two be used. The other will be allocated and ignored.

5.9 The SYSOUT Parameter

Print records generated by a program are not normally routed directly to a physical printer (theoretically it is possible, but in practice it is seldom done). Instead, they are written on the SPOOL pack and saved there for later viewing on a terminal or printing (or both). This is called output spooling, and is under the control of JES2 or JES3, which later can use one of their print routines to print the dataset. These print routines must schedule the datasets for printing, and message classes are used for this purpose. All print routines (called printers or writers) are associated with one or more classes (in all 36 classes) and each dataset to be printed must also be assigned classes. The printer routines select datasets for printing in a very similar way as initiators selects jobs for executions. Use S.ST option of ISPF menu to view the output dataset.

The SYSOUT parameter can assign this class, known as sysout or output class, to a dataset. Such datasets are called sysout or output datasets.

General Syntax

SYSOUT=(class| *) keyword parameter

class – Identifies the sysout class of the dataset from A to Z and 0 to 9.

*- Indicates that the same class used in the MSGCLASS parameter of the JOB statement (or the installation-defined default, if MSGCLASS parameter is omitted) is to be used.

E.g. 1 //SYSOUT DD SYSOUT=A

E.g. 2 //SYSPRINT DD SYSOUT=*

This DD statement is used for printing system messages generated by JES2 or JES3. Each step must have SYSPRINT DD statement. Absence will cause "SYSPRINT DD STATEMENT MISSING" message in the sysout.

E.g. 3 //SYSOUT DD SYSOUT=* (or any sysout class may be assigned)

This DD statement is used when you have COBOL DISPLAY clause in your program.

5.10 Concatenation

Concatenating Datasets

At times, program may have to read in sequence several input datasets as if they were one. This can be accomplished without physically putting the data in one datasets. This is done by concatenating the datasets in JCL code with comparable DCB characteristics without programming changes.

Note that only sequential and partitioned datasets can be concatenated. For sequential datasets, the maximum number of concatenations is 255 and for PDS it is 16. Concatenation has meaning only for sequential processing.

E.g. 1. Concatenation of physical sequential files.

```
//DD1    DD  DSN=DA0001T.PCS.GROUP1,DISP=SHR
//         DD  DSN=DA0001T.PCS.GROUP2,DISP=SHR
//         DD  DSN=DA0001T.PCS.GROUP3,DISP=SHR
```

E.g. 2. Concatenation of partitioned datasets.

```
//DD1    DD  DSN=DA0001T.PDS1.GROUP1,DISP=SHR
//         DD  DSN=DA0001T.PDS2.GROUP2,DISP=SHR
//         DD  DSN=DA0001T.PDS3.GROUP3,DISP=SHR
```

There are number of rules and restrictions for concatenations:

1. The first concatenation is the only one with a ddname.
2. The logical record length and the record format of concatenated datasets must be the same. However, the blocksizes need not be.

3. The blocksize of the first concatenation must be greater than or equal to blocksizes of all subsequent concatenation. Violation of this rule results in S001-04 ABEND failure.

E.g. Assume that in the JCL below, the first concatenation has a blocksize of 800, the second a blocksize of 800- and the third a blocksize of 23400.

```
//INFILE  DD  DSN=DA0001T.PCS.GROUP1,DISP=SHR,DCB=23400
//          DD  DSN=DA0001T.PCS.GROUP2,DISP=SHR
//          DD  DSN=DA0001T.PCS.GROUP3,DISP=SHR
```

4. Both sequential datasets and partitioned datasets can be concatenated, but not with each other – sequential with sequential and partitioned with partitioned only. Member of a PDS is treated as sequential dataset and thus can be concatenated with sequential dataset.

E.g.

```
//IN          DD  DSN=DA0001T.EMPFILE,DISP=SHR
//            DD  DSN=DA0001T.PCS.DATA(EMP),DISP=SHR
```

5. Disk as well as tape datasets can be concatenated but not with each other. Only like devices should be concatenated, disk with disk and tape with tape.

5.11 DUMMY Parameter

The DUMMY parameter is a positional parameter. At times, one might want to execute a program but suppress read or write operations in certain jobs, For example. not print a report. At other times, one might want to test a program without actually processing data. At times a DD statement referring to a dataset may be coded in the in a JCL in production region .The DUMMY parameter may be coded in a test region when the same JCL is to be executed.

The DUMMY parameter specifies that:

- No device or external storage be allocated
- No disposition processing is performed
- No input or output operations are performed for sequential access methods

Remark:

1 DCB information is established. Generally used during testing process and in procedures. Instead of using DUMMY, one may use DSN=NULLFILE. It differs from DUMMY by virtue of its position. It is a keyword parameter.

```
e.g.   //DD1     DD  DSN=NULLFILE
```

2 When an attempt to dummy a PDS is made will cause an S013-64 ABEND failure.

3 The DCB parameter may be required while coding DUMMY. Failure to do so may cause an S013-10 ABEND failure.

4 DUMMY provides a safe way to eliminate I/O activity when required.

5.12 The JOBLIB DD Statement

The JOBLIB statement identifies the program library (load library) where the programs to be executed throughout the job resides. It must be placed between the JOB and the first EXEC statement.

E.g.1:

```
//DA0001TA    JOB,LA2719,……
//JOBLIB       DD DSN=DA0001T.LIB.LOADLIB,DISP=SHR
//S1           EXEC  PGM=PROGA
//S2           EXEC  PGM=PROGB
```

Explanation: PROGA (and PROGB) is expected to reside in DA0001T.LIB.LOADLIB as a member of a library and the system searches the directory. If not found will search certain predefined libraries and the S806-04 ABEND failure occurs.

Remark:
A JOBLIB DD statement can have several concatenations (max: 16)

E.g. 2.

```
//DA0001TA  JOB,LA2719,……
//JOBLIB      DD DSN=DA0001T.LIB.LOADLIB,DISP=SHR
//              DD  DSN=DA0001T.LIB1.LOADLIB,DISP=SHR
//              DD DSN=DA0001T.PROD.LOADLIB,DISP=SHR
//S1            EXEC  PGM=PROGA
```

Explanation: All concatenations may be searched to locate a program. If, however, the program is found in a concatenation other than the last one, other concatenations will not be used. Note that, if duplicate names exist in different concatenations, the user can decide which one is to be executed by determining the sequence of the concatenations.

5.13 The STEPLIB STATEMENT

The STEPLIB statement identifies the program library (load library) where the program to be executed for the step resides. It can be placed anywhere after the EXEC statement.

E.g. 1:

```
//DA0001TA    JOB,LA2719,……
//s1            EXEC  PGM=PROGA
//STEPLIB       DD DSN=DA0001T.LIB.LOADLIB,DISP=SHR
//S2            EXEC  PGM=PROGB
//STEPLIB       DD DSN=DA0001T.LIB.LOADLIB1,DISP=SHR
```

Explanation: Program PROGA is expected to reside in DA0001TA.LIB.LOADLIB as a member of the library. If not found, will search certain predefined libraries and the S806-04 ABEND failure occurs.

E.g. 2:

```
//DA0001TA    JOB,LA2719,……
//JOBLIB        DD DSN=DA0001T.LIB.LOADLIB1,DISP=SHR
//S1             EXEC  PGM=PROGA
//S2             EXEC  PGM=PROGB
//STEPLIB       DD DSN=DA0001T.LIB.LOADLIB,DISP=SHR
//S3             EXEC  PGM=PROGC
```

A STEPLIB DD statement has the effect of negating the JOBLIB DD statement for a particular step.

5.14STORAGE DUMP

When a step encounters an ABEND failure, it is often advantageous to request a virtual storage dump, which can then be helpful in determining the cause of an ABEND. To request a storage dump, one of the following three DD statements must be included in the step:

- A SYSUDUMP DD statement
- A SYSMDUMP DD statement
- A SYSABEND DD statement

```
//SYSUDUMP   DD SYSOUT=*
```

All virtual storage allocated to your program i.e. user region of job's address space. It is a formatted dump. SYSUDUMP usually writes to sysout. It can, however, write to a disk dataset, providing a way to preserve the SYSUDUMP information for later viewing and analysis.

```
//SYSUDUMP DD DSN=DA0001T.DUMPFILE,SPACE=(TRK,(0,5),RLSE),
//       DISP=(,DELETE,CATLG),UNIT=SYSDA
```

No DCB is required.

Remark: SYSUDUMP DD statement is more often used.

```
//SYSMUDUMP   DD SYSOUT=*
```

This is same as SYSUDUMP DD statement except for the fact that the dump is unformatted. This type of dump is very difficult to analyze unless it is saved on a disk and then processed by the PRDUMP service aid. SYSMDUMP is seldom used.

```
//SYSABEND   DD SYSOUT=*
```

When a SYSUDUMP DD statement is included in a step which ABEND's, a formatted virtual storage dump will be provided. This dump will also include information about the failed step, as well as most of the MVS storage-resident information, which is of no use to the average user. SYSABEND is intended for system programmer.

Remark:

1 If neither a SYSUDUMP nor a SYSMDUMP nor a SYSABEND statement is coded within a JCL of an ABENDing step, a small amount of information is provided. This information is seldom useful in resolving the problem that caused the ABEND failure.

2 If more than one of the above statements is included in the JCL of a step, only the last one will be used. The previous ones will be ignored.

```
//S1          EXEC PGM=ASS1
//SYSABEND    DD SYSOUT=*
//SYSUDUMP    DD SYSOUT=*
```

Note that SYSUDUMP will be in use.

6 PROCEDURE

- There are two types of procedures:
  - Cataloged Procedures
  - Instream Procedures

- CATALOGED PROCEDURE is a member of a PDS, which is often referred to as procedure library, or just PROCLIB

- INSTREAM PROCEDURES is contained within job's input stream.

Invoking a procedure

```
//pr   exec   abc
          Or
//pr   exec PROC=abc
```

Restrictions

  - Max 255 steps

- The following are not permitted to reside in a procedure:

  - JOB Statement
  - An EXEC statement invoking a procedure
  - JOBLIB
  - JOBCAT
  - DD * or DATA
  - // Null statement
  - A PEND statement (used in Instream procedures)

Common Rules for EXEC & DD Statement to Override for JCL Procedures

- Only the contents of the parameter field of an EXEC, DD or OUTPUT statement can be overridden

- A parameter can be replaced, added or nullified

- When replacing an existing parameter, the overriding parameter must be specified in its complete format. DCB is an exception

- An overriding parameter replaces the same parameter, if it exists. It is added to the statement if it does not exist

- A syntactical JCL error inside a procedure cannot be corrected by overriding the erroneous parameter

6.1 Rules for EXEC statement overriding

- To override an EXEC parameter, " parameter.stepname=value" must be coded when adding or replacing a parameter, and "parameter.stepname=" must be coded when nullifying a parameter
- The PGM parameter cannot be overridden
- All overriding EXEC parameters must be coded in the EXEC statement that invokes the procedure
- All overrides to EXEC parameters for a step must be completed before overriding parameters in a subsequent step. Within a particular step the sequence of overriding parameters is not important
- An EXEC statement can be neither added nor removed by means of overriding

6.2 Rules for DD statement overriding

- To override any parameter in a DD statement an independent DD statement must be supplied in the following format:

  //stepname.ddname DD overriding parameters

- To override any parameters in an concatenation other than the first one, the following must be coded:

```
//stepname.ddname   DD
//                  DD
//                .
//                .
//                  DD overriding parameters
```

- To add an entire DD statement

  // stepname.ddname DD complete parameter field must be coded.

- The sequence of overriding DD statements must be the same as the sequence of the corresponding overridden statements. The sequence of overriding parameters is not important, except for those which are positional
- An additional DD statement must be the last one in a step's overriding statements. When several additional DD statements are supplied their relative sequence is not important, unless referbacks are used
- A DD statement cannot be removed by means of overriding

```
Procedure LAM
//S1          EXEC PGM=ED, PARM=(A,B,C,E),
//              REGION=900K, TIME = (5,30)
//STEPLIB     DD DSN=DEV.LOADLIB,DISP=SHR
//IN1         DD DSN=USER1.FILE2,DISP=SHR
//IN2         DD DSN=USER1.FILEX,DISP=OLD
//              UNIT=TAPE, VOL=SER=000101
//REP         DD SYSOUT =*,
//OUT         DD DSN=USER1.PLA,DISP=(,CTLG,DELETE),
//              UNIT=SYSDA, VOL=SER=BS3003,
//              SPACE=(CYL,(20,5)),DCB=(BLKSIZE=4000,
//              LRECL=80, RECFM=FB)
```

Required in step S1:

a) PARM must be (A,B,C,D) and TIME nullified
b) In IN1, DSN must be USER1.FILE3
c) IN2 must retrieve USER1.FILEX as a cataloged dataset
d) In DD statement OUT, BLKSIZE must be 23440

```
//S2          EXEC PGM=FORM,REGION=900k
//INA         DD     DSN=USER1.PLA,DISP=SHR
//              DD     DSN=USER1.F226,DISP=SHR
//              DD     DSN=USER1.F232,DISP=SHR
//              DD     DSN=USER1.F118,DISP=SHR
//OUTA        DD     DSN=USER.F323,DISP=(,CATLG,DELETE),
//              UNIT=TAPE, VOL=SER=001110,
//              DCB=BLKSIZE=32700, LRECL=100,
//              RECFM=FB)
//PRNT DD SYSOUT=*
```

Required in Step S2:

a) COND = (0,LT) must be coded
b) In INA DSN in the third concatenation must be USER1.F228
c) In DD statement OUTA, UNIT be SYSDA
d) An entire DD statement:
   //STEPLIB DD DSN=DEV.LOADLIB,DISP=SHR
   Must be added.

```
//S3          EXEC PGM=REPO,REGION = 400K, COND=(0,LT)
//IN3         DD     DSN=USER1.F333, DISP=OLD
//OUT3        DD     DSN=USER1.F111,DISP=(,CTLG,,DELETE,
//              UNIT=SYSDA, VOL=SER=DEV012,
//              SPACE=(CYL,(50,15),RLSE),
//              DCB=(BLKSIZE=23440,LRECL=80,RECFM=FB)
//PRNT        DD SYSOUT=*
//
```

Required in Step S3:

a) EVEN must be added to the COND parameter
b) In DD statement OUT3, RLSE must be removed from the SPACE parameter and VOL parameter must be nullified.

The final output would be

```
//ZP          EXEC LAM PARM.S1=(A,B,C,D),TIME.S1=,COND.S2=(0,LT),
//INA         COND.S3=((0,LT),EVEN)
//S1.IN1      DD   DSN=USER1.FILE3
//S1.IN2      DD   VOL=                    ALTERNATIVE: VOL=SER=
//S1.OUT      DD   DCB=BLKSIZE=23440
//S2.INA      DD
//            DD
//            DD   DSN=USER1.F228
//S2.OUTA     DD    UNIT=SYSDA
//S2.STEPLIB  DD    DSN=DEV.LOADLIB,DISP=SHR
//S3.OUT3     DD   SPACE=(CYL,(50,15)),VOL=
```

Some typical examples

Example 1

```
//S1          EXEC PGM=ONE
//OUT1        DD     DSN=U1.S1,
//            DISP=(,CTLG, DELETE),
//            UNIT=TAPE,
//            DCB=(BLKSIZE=32700)
```

Required
OUT1 must be dummied

Override
//S1.OUT1 DD DUMMY

Regardless of the contents, no other parameters are needed.

Example 2:

```
//S1          EXEC PGM=ONE
//IN1         DD     DSN=U1.B1,DISP=SHR
//            DD     DSN=U1.B2,DISP=SHR
//            DD     DSN=U1.B3,DISP=SHR
```

Required
Second concatenation of IN1 must be dummied.

Proposed overriding DD statement.

```
//S1.IN1        DD
//              DD      DUMMY
```

Overriding and dummying the second concatenation will cause the third concatenation to also act as DUMMY. The desired result can be accomplished by the following:

Override

```
//S1.IN1        DD
//              DD      DSN=U1.B3
//              DD      DUMMY
```

Example 3:

```
//S1            EXEC PGM=ONE
//CNTL          DD    DSN=U1.CNTLIB(S1), DISP=SHR
```

Required

DD statement CNTL must be //CNTL DD *

Override

```
//S1.CNTL       DD *
```

Regardless of the contents DD * will override all.

Example 4:

```
//S1            EXEC PGM=ONE
//OUT4          DD      DSN=U1.D1,DISP=NEW
//                      DISP=SYSDA,VOL=SER=TEST26,
//                      SPACE=(TRK,(500,50)),
//                      DCB=(BLKSIZE=23400,RECL=100,RECFM=FB)
```

Required

DCB parameter must be eliminated

Override

```
//S1.OUT4 DD DCB=(BLKSIZE=,,LRECL=,RECFM=)
```

6.3 Symbolic Parameters and Symbolic Overrides

- Symbolic overrides can be used only when symbolic parameters have been coded inside the procedure
- A symbolic parameter is a name preceded by an ampersand (&)
- A symbolic parameter can be coded in place of any parameter, part of a parameter in the parameter field of an EXEC, DD or OUTPUT statement

Symbolic parameter

Example 1:

```
Procedure BLTX
//S1    EXEC PGM=BL,HQ=DA0001T
//IN    DD     DSN=&HQ..INFILE,,DISP=SHR
//OUT   DD    DSN=&HQ..OUTFILE,DISP=(,CATLG,DELETE),
//              UNIT=SYSDA, DCB=BLKSIZE=32700
```

```
//PSK   EXEC  PROC=BLTX,HQ=PROD
```

- First period works as delimiter

Example 2:

```
Procedure BLTX

//S1    EXEC  PGM=BL,HQ='DA0001T.'
//IN    DD    DSN=&HQ.INFILE,DISP=SHR
//OUT   DD    DSN=&HQ.OUTFILE,DISP=(,CATLG,DELETE)
//              UNIT=SYSDA,DCB=(BLKSIZE=32700)
```

```
//PSK   EXEC  BLTX,HQ='PROD.'
```

Symbolic overriding

Rules for Symbolic overriding

- An EXEC statement keyword (TIME, REGION etc.) cannot be used as a symbolic parameter
- A symbolic override in either the EXEC or PROC statement that has no corresponding parameter in the procedure will result in a 'SYMBOL NOT DEFINED' JCL error
- If a symbolic and a regular override conflict, the regular override always prevails
- A symbolic parameter, which is immediately followed by an alphabetic, numeric or national character must have a period at its end
- A symbolic parameter can be coded many times in a procedure. When substitution occurs, all the occurrences will receive the same value
- When nothing must be substituted for a symbolic parameter, "symbolic-override=“ must be coded in the EXEC or PROC statement

```
Procedure SWP
/ABC   PROC    R=800K, Q=AUX, U=TAPE
//S1    EXEC    PGM=P2, REGION=&R
//IN    DD      DSN=&Q..FILEX, DISP=SHR
//OUT   DD      DSN=&Q..FILEY, DISP=(,CATLG,DELETE),
//              UNIT=&U
```

```
//A      EXEC    PROC=SWP,Q=MAX
```

Substitution results in

```
//S1    EXEC  PGM=P2, REGION=800K
//IN    DD DSN=MAX.FILEX,DISP=SHR
//OUT   DD DSN=MAX.FILEY,DISP=(,CATLG,DELETE),
//              UNIT =TAPE
```

6.4 The PROC statement

- The purpose of the PROC statement is to contain symbolic override defaults
- When a procedure is executed, the system will substitute symbolic parameters using symbolic overrides coded in the EXEC statement
- For those symbolic overrides not found in the EXEC statement, the default symbolic overrides in the PROC statement will be used

6.5 In-stream Procedures

- An in-stream procedure is a part of a job's input stream and exists only for the duration of the job
- The PROC statement in an in-stream procedure is mandatory and serves two functions -
  a) It signals the beginning of in-stream procedure
  b) It contains default symbolic overrides.
- The PEND statement must be coded in an in-stream procedure to provide a delimiter

Remark:

1) A PROC statement in a catalogued procedure is optional. The only reason it is required is to contain default symbolic overrides.

Example 1:

```
//DA0001TA   JOB    LA2719,PCS,MSGCLASS=A,,MSGLEVEL=(1,1),
//              N OTIFY=DA0001T
//*       Instream procedure
//PROCBR14  PROC
//S1            EXEC  PGM=IEFBR14
//SYSPRINT  DD      SYSOUT=*
//DD1           DD      DSN=DA0001T.TEMP,DISP=(OLD,DELETE,DELETE)
//              PEND
//*
//STEP1         EXEC  PROC=PROCBR14
//S1.DD1        DD      DSN=DA0001T.TEMP1
//              DISP=(,CATLG,DELETE), UNIT=SYSDA,
//              SPACE=(TRK,(2,1)),
//              DCB=(LRECL=80,RECFM=FB,BLKSIZE=800)
//
```

Example 2:

```
//DA0001TA   JOB    LA2719,PCS,MSGCLASS=A,,MSGLEVEL=(1,1),
//              N OTIFY=DA0001T
//*       Instream procedure with symbolic parameters
//PROCBR14  PROC USRID=DA0001T,DATANAME=TEMP
//S1            EXEC  PGM=IEFBR14
//SYSPRINT  DD      SYSOUT=*
//DD1           DD      DSN=&USRID..&DATANAME,DISP=(OLD,DELETE,DELETE)
//              PEND
//*
//STEP1         EXEC  PROC=PROCBR14,DATANAME=EMPFILE
//S1.DD1        DD      DSN=DA0001T.EMPFILE
//              DISP=(,CATLG,DELETE), UNIT=SYSDA,
//              SPACE=(TRK,(2,1)),
//              DCB=(LRECL=80,RECFM=FB,BLKSIZE=800)
//
```

7 UTILITY

7.1 IEFBR14 UTILITY

This utility is commonly used to delete, allocate and uncatalog a dataset.

Example 1:

```
//DELETE      EXEC  PGM=IEFBR14
//* TO DELETE A FILE
//SYSPRINT    DD SYSOUT=*
//DD1         DD      DSN=DA0001T.EMPLOYEE,
//                    DISP=(MOD,DELETE,DELETE),
//                    UNT=SYSDA, SPACE=(TRK,0)
```

Example 2:

```
//CREATE              EXEC  PGM=IEFBR14
//*TO ALLOCATE A NEW FILE
//SYSPRINT            DD SYSOUT=*
//DD1                 DD      DSN=DA0001T.EMPLOYEE,
//                    DISP=(NEW,CATLG,DELETE),
//                    UNIT=SYSDA,
//                    SPACE=(TRK,(2,1)),
//                    DCB=(BLKSIZE=800,LRECL=80,
//                    RECFM=FB,DSORG=PS)
```

An excerpt of a code illustrating, the deletion of a dataset, before COBRUN1 step is executed.

```
//DA000ITA    JOB     LA2719,PCS,NOTIFY=DA0001T,
//                    MSGCLASS=X, MSGLEVEL=(1,0)
//DELETE      EXEC  PGM=IEFBR14
//LOGFILE     DD      DSN=DA0001T.MYFILE2,DISP=(MOD,DELETE,DELETE),
//                    SPACE=(TRK,0),UNIT=SYSDA
//*
//COBRUN1     EXEC  PGM=ASS2
//STEPLIB     DD      DSN=DA00021T,SHEELA.LOADLIB,DISP=SHR
//INFILE      DD      DSN=DA0001T, EMPLOYEE,DISP=SHR
//OUTFILE     DD      DSN=DA0001T.MYFILE2,
//               DISP=(NEW,CATLG,DELETE),
//               DCB=(LRECL=80, DSORG=PS,BLKSIZE=80, RECFM=FB),
//               VOL=SER=BS3011,
//               SPACE=(TRK, (5,1))
//SYSOUT      DD      SYSOUT=*
//
```

7.2 IEBGENER UTILITY

- This utility is commonly used to copy, concatenate and to empty sequential datasets

```
//DA0001TA    JOB     LA2719, PCS, NOTIFY=DA0001T,
//                    MSGCLASS=X
//*********************************************************************
//* USING THE IEBGENER UTILITY TO MERGE DATASETS
//* SYSUT1 PROVIDING THE INPUT AND SYSUT2 BEING
//* THE OUTPUT
//*********************************************************************
//CPYSTEP     EXEC  PGM=IEBGENER
//SYSUT1      DD    DSN=DA0001T.INDATA1, DISP=SHR
//SYSUT2      DD    DSN=DA0001T.NEW,DISP=MOD
//SYSIN       DD    DUMMY
//SYSPRINT    DD    SYSOUT=*
//
```

```
//DA0001TA    JOB     LA2719,PCS,NOTIFY=DA0001T,
//                    MSGCLASS=X
//*********************************************************************
//*  USING THE IEBGENER UTILITY TO CONCATENATE DATASETS
//*  SYSUT1 PROVIDING THE INPUT AND SYSUT2 BEING
//*  THE OUTPUT
//*********************************************************************
//CPYSTEP     EXEC  PGM=IEBGENER
//SYSUT1      DD    DSN=DA0001T.INDATA1,DISP=SHR
//                  DSN=DA0001T.INDATA3,DISP=SHR
//SYSUT2      DD    DSN=DA0001T.MYOUT,
//                  DISP=(NEW, CATLG, DELETE),
//                  UNIT=SYSALLDA,
//                  SPACE=(TRK,(5,1),RLSE)
//SYSIN       DD    DUMMY
//SYSPRINT    DD    SYSOUT=*
```

```
//DA0001TA  JOB    LA2719,PCS,NOTIFY=DA0001T,
//                 MSGCLASS=X
//  ********************************************************************
//* USING THE IEBGENER UTILITY TO EMPTY EXISTING DATASET
//********************************************************************
//CPYSTEP   EXEC   PGM=IEBGENER
//SYSPRINT  DD     SYSOUT=*
//SYSUT1    DD     DUMMY, DCB=(BLKSIZE=800,
//                 LRECL=80, RECFM=FB)
//SYSUT2    DD     DSN=DA0001T.MYOUT,DISP=SHR
//SYSIN     DD     DUMMY
//
```

7.3 SORT UTILITY

- The utility is commonly used to sort data, copy selective data, remove duplicates, and change data throughout the file
- This is the utility provided by MVS

Usage: It reorders the Physical Sequential dataset as per requirement on given field(s). These fields are called control fields or key fields.

Working: It assumes that all input records are out of sequence and it puts them in sequence you request.

E.g. Employee data is sorted in the sequence of Emp. No., Emp. Name or Salary etc.

Sort utility

Syntax

Sort fields = (position, length, format, sequence) or
Sort fields = (position, length, sequence….), format = format

This syntax is used if all the fields on which the dataset to be sorted are of same type.

position: Location of the 1st byte of the key field, in the input record

length: Length in bytes of the key field. Sum of all key fields (their lengths) should not exceed 4092

Format: Two characters code that identifies the format (type) of the data

Sequence: A - Ascending
D - Descending

Merge Utility

- This assumes that records are in proper sequence but at different locations i.e. in different files. It merges those files into one, in the given sequence.

E.g.: General ledger transactions for different months, in the sequence of a/c no to be merged in one file.

Remark: Many of the examples in this handout refer to EMPMAST data set in Appendix A.

DFSORT:

DFHSORT is a member of IBM's Data Facility family of products. The DFSORT licensed program is a high-performance data arranger developed by IBM for MVS users.

With DFSORT, you can sort, merge, and copy data sets. You can use it to aid complex tasks such as taking inventory or running a billing system. You can also use DFSORT's record-level editing capability to perform data management tasks.

Sorting Data Sets

You can use DFSORT to rearrange the records in your datasets. Sorting is arranging records in either ascending or descending order within a file.

The fields in the records can be in any IBM System/370 format (for example EBCDIC character, decimal, and binary)

You can sort data in several different formats. Figure shows the most common data formats and the codes you use to specify them.

Data Format	Code
EBCIDIC (Character)	CH
Binary (Numeric)	BI
Zoned Decimal (Numeric)	ZD
Packed Decimal (Numeric)	PD

Merging Data Sets

You can also use DFSORT to merge data sets. DFSORT merges data sets by combining two or more files of sorted records to form a single data set of sorted records.

You can merge up to 16 data sets. The data sets you merge must be previously sorted into the same order (ascending or descending order).

- The JCL needed for a merge is the same as that for a SORT, with the following exceptions

➢You do not use the SORTWKnn statement
➢Instead of SORTIN DD statement, you use SORTINnn DD statements to define the input datasets. The SORTINnn DD statements name the input datasets to be merged and tell how many datasets are to be merged. The value nn in SORTINnn is a number from 0 to 16,indicating the number of datasets to be merged.

Copying Data Sets

DFSORT can also copy data sets without any sorting or merging taking place. You copy data sets in much the same way that you sort or merge them.

What else can you do with DFSORT?

While sorting, merging, or copying data sets, you can also:

- Select a subset of records from an input data set. You can include or omit records that meet specified criteria
- Reformat records, add or delete fields, and insert blanks, constants, or binary zeros. For example you can make a report more legible by inserting blank characters to separate fields
- Sum the values in selected records while sorting or merging (but not while copying)
- Alter the collating sequence when sorting or merging records (but not while copying). For example, you can have the lowercase letters collate after the uppercase letters

Creating and Running DFSORT Jobs

Processing data sets with DFSORT involves two steps:

1. Creating a DFSORT job
2. Running a DFSORT job.

You can run a DFSORT job by invoking processing in a number of ways:

- With a JCL EXEC statement using the name of the program or the name of the cataloged procedure
- With interactive panels supported under ISPF and ISMF
- Within programs written in COBOL, PL1, or basic Assembler language

Remarks: JCL-invoked means that the DFSORT program is initiated by JCL EXEC statement. The phrase dynamically invoked means that the DFSORT program is initiated from another program.

The JCL statements you need for most jobs are described below:

//jobname JOB Signals the beginning of a job.

//stepname EXEC Signals the beginning of a job step and tells the operating system what program to run.

//stepname EXEC PGM=SORT

//STEPLIB DD defines the library containing DFSORT program. If your DFSORT program is in system library, you can omit the STEPLIB statement.

//SYSOUT DD defines the output data set for messages.

//SORTIN DD defines the input data set.

//SORTWKnn DD defines a work storage data set for a sort. For most applications, one work storage data set is sufficient. Increasing the number of work storage data sets does not improve performance.

//SORTOUT DD defines the output dataset.

//SYSIN DD Control statements.

All the control information within SYSIN DD can be coded freely between column 2 and column 71.

SORT JCL 1

```
//DA0001TA JOB LA2719,PCS, NOTIFY=DA0001T,MSGCLASS=X
//*****************************************************
//*SORT ON THE EMPLOYEE NAME IN ASCENDING ORDER
//*****************************************************
//SRTSTEP    EXEC PGM = SORT
//SYSIN      DD   *
  SORT FIELDS = (1,5,CH,A)
/*
//SORTIN     DD     DSN=DA0001T.EMPLOYEE,DISP=SHR
//SORTOUT    DD     DSN=DA0001T.OUTSORT,
//                  SPACE=(TRK,(1,1)),UNIT=SYSDA
//SORTWK01   DD     SPACE=(CYL,(1,1)),UNIT=SYSDA
//SYSPRINT   DD     SYSOUT=*
//SYSOUT     DD     SYSOUT=*
//SORTMSG    DD     SYSOUT=*
//
```

Sorting by Multiple fields

You can further sort the records in the data set by specifying multiple control fields. When you specify two or more control fields, you specify them in the order of greater to lesser priority.

SORT JCL 2

```
//DA0001TA JOB LA2719,PCS, NOTIFY=DA0001T,MSGCLASS=X
//*****************************************************
//*SORTS ON ASCENDING DEPTNO & DESCENDING ENAME
//*****************************************************
//SRTSTEP    EXEC  PGM=SORT
//SYSIN      DD *
 SORT FIELDS = (17,2,PD,A,2,6,CH,D)
/*
//SORTIN     DD     DSN=DA0001T.DEPT,DISP=SHR
//SORTOUT    DD     DSN=DA0001T.SORTOUT2,
//                  DISP=(NEW,CATLG,DELETE)
//                   SPACE=(TRK,(3,3)),UNIT = SYSDA
//SORTWK01   DD     SPACE(CYL,(1,1)),UNIT=SYSDA
//SYSPRINT   DD     SYSOUT=*
//SYSOUT     DD     SYSOUT=*
//SORTMSG    DD     SYSOUT=*
//
```

Copying Data Sets

With DFSORT you can copy data sets directly without performing a sort or merge.

You can use COPY with all of the other DFSORT control statements except SUM. DFSORT can select and reformat the specific data sets you want to copy by using the control statements covered later.

You can use SORT FIELDS=COPY or MERGE FIELDS=COPY or OPTION COPY to produce the same results.

SORT JCL 3

```
//DA0001TA JOB LA2719,PCS, NOTIFY=DA0001T,MSGCLASS=X
//*******************************************************
//
//*******************************************************
//SRTSTEP    EXEC  PGM=SORT
//SYSIN      DD *
 SORT  FIELDS = COPY
/*
//SORTIN     DD     DSN=DA0001T.DEPT,DISP=SHR
//SORTOUT    DD     DSN=DA0001T.SORTOUT2,
//                  DISP=(NEW,CATLG,DELETE),
//                  SPACE=(TRK,(3,3)),UNIT = SYSDA
//SYSPRINT   DD     SYSOUT=*
//SYSOUT     DD     SYSOUT=*
//SORTMSG    DD     SYSOUT=*
//
```

The JCL for a copy application is the same as for a sort, except that you do not use the SORTWKnn DD statement.

Tailoring the Input Data Set with INCLUDE and OMIT

Often, you need only a subset of the records in a data set for an application. By tailoring the data set, you can increase the speed of the sort, merge, or copy, The fewer the records, the less time it takes to process them.

You tailor an input data set by:

- Using an INCLUDE control statement to collect wanted records
- Using an OMIT control statement to exclude unwanted records

Your choice of INCLUDE and OMIT depends on which is easier and more efficient to write for a given application. Note that, INCLUDE and OMIT control statements are mutually exclusive.

SORT JCL 4

TO COPY SELECTIVE DATA

a) INCLUDE COND copies data that matches the condition given for e.g. in this case it will copy data where one character in 19th position equals 'M' or 'S'.

```
//DA0001TA JOB LA2719,PCS, NOTIFY=DA0001T,MSGCLASS=X
//*********************************************************************
//* SORTS ON THE INPUT FILE ON JOB AND SELECTS JOB BEGINNING WITH M
//   OR S INTO A DATA SET
//*********************************************************************
//SRTSTEP    EXEC  PGM=SORT
//SYSIN              DD  *
  OPTION EQUALS
  INCLUDE COND = (19,1,CH,EQ,C'M', OR, 19, 1, CH, EQ, C 'S')
  SORT FIELDS = (19,6,A), FORMAT=CH
/*
//SORTIN      DD   DSN=DA0001T.INDATA3,DISP=SHR
//SORTOUT     DD   DSN=DA0001T.SORTOUT3,DISP=(NEW, CATLG,DELETE),
//                 SPACE=(TRK,(3,3)), UNIT = SYSDA,
//                 DCB=(BLKSIZE=800, LRECL=80, RECFM=FB,DSORG=PS)
//SORTWK01  DD   SPACE=(TRK,(10,5)), UNIT=SYSDA
//SYSOUT      DD   SYSOUT=*
//
```

SORT JCL 5 (Sorts on Job and selects Jobs beginning with M and Deptno beginning with 1)

```
//DA0001TA    JOB LA2719,PCS, NOTIFY=DA0001T,MSGCLASS=X
//*************************************************************
//*SORTS ON JOB INCLUDES JOBS BEGINNING WITH M AND DEPTNO
//*BEGINNING WITH 1
//*************************************************************
//SRTSTEP    EXEC  PGM=SORT
//SYSIN        DD *
  OPTION EQUALS
  INCLUDE COND=(19,1,CH,EQ,C'M',AND,51,1,CSF,EQ,1)
  SORT FIELDS = (19,6,A),FORMAT=CH
/*
//SORTIN      DD     DSN=DA0001T.INDATA3,DISP=SHR
//SORTOUT     DD     DSN=DA0021T.SORTOUT4,DISP=NEW,CATLG,DELETE),
//                   SPACE=(TRK,(3,3,)), UNIT = SYSDA,
//                   DCB=(BLKSIZE=800, LRECL=80, RECFM=FB,,DSORG=PS)
//SORTWK01  DD    SPACE=(CYL,(1,1)),UNIT=SYSDA
//SYSOUT      DD     SYSOUT=*
//
```

SORT UTILITY

OMIT COND: INCLUDE and OMIT are mutually exclusive

Records which do not satisfy the condition are sorted and copies into the output dataset

SORT JCL 6(Sorts on Job and omits Jobs beginning with M or S)

```
//DA0001TA     JOB LA2719,PCS, NOTIFY=DA0001T,MSGCLASS=X
//*************************************************************
//*SORTS ON JOB *OMITS JOBS BEGINNING WITH M OR S
//*************************************************************
//SRTSTEP     EXEC  PGM=SORT
//SYSIN         DD  *
 OPTION EQUALS
 OMIT COND = (19,1,CH,EQ,C,'M',OR, 19,1,CH,EQ,C'S')
 SORT FIELDS = (19,6,A),FORMAT=CH
/*
//SORTIN       DD   DSN=DA0001T.INDATA3,DISP=SHR
//SORTOUT     DD   DSN=DA0021T.SORTOUT4,DISP=NEW,CATLG),
//                        SPACE=(TRK,(3,3,)), UNIT = SYSDA,
//                         DCB=(BLKSIZE=800, LRECL=80, RECFM=FB,DSORG=PS)
//SORTWK01  DD  SPACE=(CYL,(1,1)),UNIT=SYSDA
//SYSOUT      DD      SYSOUT=*
//
```

SORT JCL 7

```
//DA0001TA     JOB LA2719,PCS, NOTIFY=DA0001T,MSGCLASS=X
//* Merges fields beginning with column 110 having length 5
//* INDATA1 and INDATA2 are sorted on the control field
//SRTSTEP     EXEC  PGM=SORT
//SORTIN01    DD   DSN=DA0001T.INDATA1,DISP=OLD
//SORTIN02    DD   DSN=DA0001T.INDATA2,DISP=OLD

//SORTOUT     DD   DSN=DA0001T.SORTOUT4,DISP=(NEW,CATLG),
//                        SPACE=(TRK,(3,3,)), UNIT = SYSDA,
//                        DCB=(BLKSIZE=800, LRECL=80, RECFM=FB,DSORG=PS)
//SYSOUT      DD   SYSOUT=*
//SYSIN         DD  *
 MERGE        FIELDS = (110,5,A),FORMAT=CH
/*
```

You can select from the following comparison operators:

C	
EQ	Equal to
NE	Not Equal to
GT	Greater than
GE	Greater than or Equal to
LT	Less than
LE	Less than or equal to

DFSORT uses the following rules for padding and truncation. Padding adds fillers in data, usually zeros or blanks. Truncation deletes or omits a leading or trailing portion of a string.

- In a field-to-field comparison, the shorter field is padded as appropriate (with blanks or zeros)
- In a field-to-constant comparison, the constant is padded or truncated to the length of the field. Decimal constants are padded or truncated on the left. Character and hexadecimal constants are padded or truncated on the right

Allowable Comparisons for INCLUDE and OMIT

The following table shows field-to-field and field-to-constant comparisons.

Field Format	BI	CH	ZD	PD
BI	✔	✔		
CH	✔	✔		
ZD			✔	✔
PD			✔	✔

Field Format	Character String	Hexadecimal String	Decimal String
BI	✔	✔	
CH	✔	✔	
ZD			✔
PD			✔

Writing Constants

The formats for writing character strings, hexadecimal strings and decimal numbers are shown below.

Character Strings

The format for writing a character string is:

C'x…..x'

Where x is an EBCDIC character. For example, C'Sheela'.

If you want to include a single apostrophe in the string, you must specify it as two single apostrophes. For example, O'NEILL must be specified as C'O''NEILL'.

Hexadecimal Strings

The format for writing a hexadecimal string is:

X'yy……yy'

Where yy is a pair of hexadecimal digits. For example X'C1C2' is equivalent to C'AB'

Decimal Strings

The format for writing a decimal number is:
n…..n or +n…..n or –n…n
where n….n is a decimal digit. Examples are 24, +24, and –24.
Decimal number must not contain commas and decimal points.

Summing Records-the SUM statement

Suppose that the TRG dept wants to know the total salary of all the trainers. You can tailor the file to include only records for the TRG department by using the INCLUDE statement, and sum the salaries by using by using the SORT and SUM statements.

On the SUM control statement, you specify one or more numeric fields that are to be summed whenever records have equal control fields (control fields are specified on the SORT statement). The numeric fields can be in binary, packed decimal, or zoned decimal format.

When you sum records, keep in mind that two types of fields are involved:

- *Control fields*, which are specified on the SORT statement
- *Summary fields*, which are specified on the SUM statement

Writing the SUM Statement

SUM FIELDS=(location, length, data-format,...)

The INCLUDE, SORT, and SUM statements are shown below:

```
INCLUDE COND=(26,4,CH,EQ,C'TRG ')
SORT FIELDS=(26,4,CH,A)
SUM FIELDS=(35,5,BI)
```

When the salaries are summed, the final sum appears in the SALARY field of one record and the other records are deleted. The default is for records with equal control fields to appear in the original order. When summing records keeping the original order, DFSORT chooses the first record to contain the original sum.

Remark: Some of the fields in your summation might not be meaningful, such as the employee number field. You could use OMIT statement to omit this field. There are two other ways to leave out fields that are not meaningful.

Suppressing Records with Duplicate Control Fields.

Apart from summing values, you can use SUM to delete records with duplicate control fields. By specifying FIELDS=NONE on the SUM statement, one can eliminate records with duplicate control fields.

E.g.: List all the distinct departments in ascending order.

```
SORT FIELDS=(25,4,CH,A)
SUM FIELDS=NONE
```

Handling Overflow

When a sum becomes larger than the space available for it, overflow occurs. If overflow occurs, the two records involved are left unsummarized. That is, the contents of the records are left undisturbed, neither record is deleted, and the records are still available for summarization. Overflow does not prevent further summary. In some cases, you can correct overflow by padding the summary fields with zeros using the INREC control statement.

Reformatting Records

You can reformat records in your data sets by using the OUTREC and INREC control statements. With OUTREC and INREC, you can:

- Delete fields
- Reorder fields
- Insert separators (blanks, zeros, or constants)

The difference between the DFSORT control statements is that OUTREC reformats records after they are sorted, copied, or merged, whereas INREC reformats records before they are sorted, copied, or merged.

INREC and OUTREC perform the same functions. When deciding which to use, remember their processing order. In general:

- If you are deleting fields, try to use INREC because shorter records take less time to sort, merge, or copy (INREC reformats the records before they are processed)
- If you are going to insert separators, use OUTREC because OUTREC inserts separators into the records after they are processed
- If you are reordering fields, you can use either control statements because reordering fields does not affect the record length

Note: If you use INREC or OUTREC to change the record length, be sure to specify the final record length on the SORTOUT DD statement using the DCB parameter. The final length is either:

- The INREC length if you are using just INREC
- The OUTREC length if you are using just OUTREC or both INREC and OUTREC

Reformatting Records Using the OUTREC Statement

Using the OUTREC statement, you can delete all the fields that are not needed for the application, in other words fields whose contents are not meaningful in a summation records. **Note that on the OUTREC statement you do not specify the data format.**

```
SORT FIELDS=(26,4,CH,A)
SUM FIELDS=(35,5,BI)
OUTREC FIELDS=(26,4,35,5)
```

Because the record length changed, the new length must be specified on the SORTOUT DD statement. For example:

```
//SORTOUT     DD DSN=DA0001T.SORTOUT,DISP=(NEW,CATLG,DELETE),
//                  SPACE=(TRK,(1,1)),UNIT=SYSDA,
//                  DCB=LRECL=9
```

Reordering Fields to Reserve Space

The fields always appear in the order in which you specify them. Therefore, if you want to the salary to appear before the department, just reverse the order in the OUTREC statement.

```
SORT FIELDS=(26,4,CH,A)
SUM FIELDS=(35,5,BI)
OUTREC FIELDS=(35,5,26,4)
```

Inserting Binary Zeros

Assume you want to reformat the records to include a new 4-byte binary field after the salary field (beginning at byte 39). In this case, you can insert binary zeros as placeholders for the new field (to be filled in with data at later date).

To insert the zeros, write 4Z after the last field:

```
SORT FIELDS=(26,4,CH,A)
SUM FIELDS=(35,5,BI)
OUTREC FIELDS=(26,4,35,5,4Z)
```

This time, you must specify on the SORTOUT DD statement the new record length is 13 bytes.

You can insert binary zeros before, between, or after fields. You can use **Z** or **1Z** to specify a single binary zero.

Inserting Blanks

If an output data set contains only character data, you can print it by writing the SORTOUT DD statement as follows:

```
//SORTOUT DD SYSOUT=*
```

You can make the printout more legible by the OUTREC statement to separate the fields with blanks and to create margins. For example, you want to print just the employee number and employee name fields.

```
SORT FIELDS=(1,4,ZD,A)
OUTREC FIELDS=(10X,1,4,,4X,5,20)
```

To insert blanks, specify nX.

You can insert blanks before, between, or after fields. You can use **X** or 1X to specify a single space.

Inserting Constants

In addition to making the printout more legible, OUTREC can also be used to set up a report format by inserting constants. The formats for writing constants are shown below:

Character Strings

The format for writing a character string is:

C'x…..x'

Where x is an EBCDIC character. For example, C'Sheela'

The format for writing a character string repetition is:

nC'x…..x'

Where n can be from 1 to 4095; n repetitions of the character string constant (C'x…x') are inserted into the reformatted input records. If n is omitted, 1 is used instead.

If you want to include a single apostrophe in the string, you must specify it as two single apostrophes. For example, O'NEILL must be specified as C'O''NEILL'.

Hexadecimal Strings

The format for writing a hexadecimal string is:

X'yy……yy'

Where yy is a pair of hexadecimal digits. For example X'C1C2' is equivalent to C'AB'

The format for writing a hexadecimal string repetition is:

nC'yy…..yy'

where n can be from 1 to 4095; n repetitions of the hexadecimal string constant (X'yy…yy') are inserted into the reformatted input records. If n is omitted, 1 is used instead.

Setting Up the Report Format

The statement for setting up the report is shown below:

```
OPTION COPY
OUTREC FIELDS=(11:C'THE EMPLOYEE NUMBER IS ',1,4,
30:C'THE EMPLOYEE NAME IS ',5,20,4X,25,4)
```

Reformatting Records Using the INREC Statement

The INREC statement has the same format as the OUTREC statement.

```
INREC FIELDS=(26,4,35,5)
SORT FIELDS=(1,4,4,CH,A)
SUM FIELDS=(5,5,BI)
```

Because INREC reformats the records before they are sorted, the SORT and SUM statements must refer to the reformatted records, as they will appear in the output data set.

Preventing Overflow Summing Values

In some cases, you can prevent overflow by using INREC to pad summary fields with zeros. However, this method cannot be used for negative fixed-point binary data, because padding with zeros rather than with ones would change the sign.

Padding Summary fields

If the summary fields were overflowing, you could pad each of them on the left with 4 bytes (binary fields must be 2, 4, or 8 bytes long)

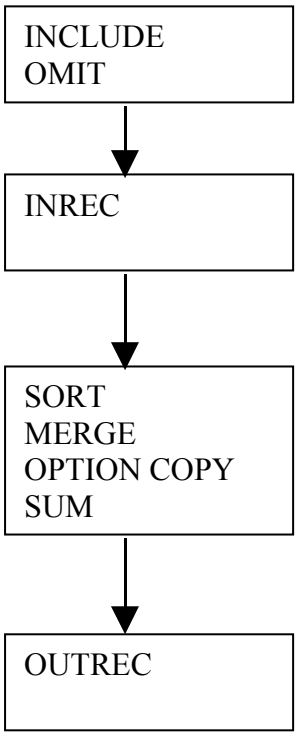
```
INREC FIELDS=(26,4,4Z,35,5)
SORT  FIELDS=(1,4,CH,A)
SUM   FIELDS=(5,10,BI)
```

You cannot use the OUTREC statement to prevent overflow, because it is processed after summarization.

Processing Order Of Control Statements

The flowchart below shows the order in which control statements are processed (SUM is processed at the same time as SORT or MERGE. It is not used with COPY.

Although you can write the statements in any order, DFSORT always processes the statements in the order shown below.



7.4 APPENDIX-A

(Bibliography/references)

Expert MVS/XA JCL -Mani Carathanassis

MVS/JCL -Doug Lowe