JCL - DEFINING DATASETS

A dataset name specifies the name of a file and it is denoted by DSN in JCL. The DSN parameter refers to the physical dataset name of a newly created or existing dataset. The DSN value can be made up of sub-names each of 1 to 8 characters length, separated by periods and of total length of 44 characters *alphanumeric*. Following is the syntax:

```
DSN=&name | *.stepname.ddname
```

Temporary datasets need storage only for the job duration and are deleted at job completion. Such datasets are represented as **DSN=&name** or simply without a DSN specified.

If a temporary dataset created by a job step is to be used in the next job step, then it is referenced as **DSN=*.stepname.ddname**. This is called **Backward Referencing.**

Concatenating Datasets

If there is more than one dataset of the same format, they can be concatenated and passed as an input to the program in a single DD name.

```
//CONCATEX JOB CLASS=6,NOTIFY=&SYSUID
//*
//STEP10    EXEC PGM=SORT
//SORTIN    DD DSN=SAMPLE.INPUT1,DISP=SHR
//          DD DSN=SAMPLE.INPUT2,DISP=SHR
//          DD DSN=SAMPLE.INPUT3,DISP=SHR
//SORTOUT   DD DSN=SAMPLE.OUTPUT,DISP=(,CATLG,DELETE),
//          LRECL=50,RECFM=FB
```

In the above example, three datasets are concatenated and passed as input to the SORT program in the SORTIN DD name. The files are merged, sorted on the specified key fields and then written to a single output file SAMPLE.OUTPUT in the SORTOUT DD name.

Overriding Datasets

In a standardised JCL, the program to be executed and its related datasets are placed within a cataloged procedure, which is called in the JCL. Usually, for testing purposes or for an incident fix, there might be a need to use different datasets other than the ones specified in the cataloged procedure. In that case, the dataset in the procedure can be overridden in the JCL.

```
//SAMPINST JOB 1,CLASS=6,MSGCLASS=Y,NOTIFY=&SYSUID
//*
//JSTEP1    EXEC CATLPROC,PROG=CATPRC1,DSNME=MYDATA.URMI.INPUT
//          DATAC=MYDATA.BASE.LIB1(DATA1)
//STEP1.IN1 DD DSN=MYDATA.OVER.INPUT,DISP=SHR
//*
//* The cataloged procedure is as below:
//*
//CATLPROC PROC PROG=,BASELB=MYCOBOL.BASE.LIB1
//*
//STEP1     EXEC PGM=&PROG
//STEPLIB   DD DSN=&BASELB,DISP=SHR
//IN1       DD DSN=MYDATA.URMI.INPUT,DISP=SHR
//OUT1      DD SYSOUT=*
//SYSOUT    DD SYSOUT=*
//SYSIN     DD MYDATA.BASE.LIB1(DATA1),DISP=SHR
//*
//STEP2     EXEC PGM=SORT
```

In the above example, the dataset IN1 uses the file MYDATA.URMI.INPUT in the PROC, which is overridden in the JCL. Hence, the input file used in execution is MYDATA.OVER.INPUT. Please note that the dataset is referred as STEP1.IN1. If there is only one step in the JCL/PROC, then the dataset

can be referred with just the DD name. Similarly, if there are more than one step in the JCL, then the dataset is to be overridden as JSTEP1.STEP1.IN1.

```
//SAMPINST  JOB 1,CLASS=6,MSGCLASS=Y,NOTIFY=&SYSUID
//*
//STEP      EXEC CATLPROC,PROG=CATPRC1,DSNME=MYDATA.URMI.INPUT
//          DATAC=MYDATA.BASE.LIB1(DATA1)
//STEP1.IN1 DD DSN=MYDATA.OVER.INPUT,DISP=SHR
//          DD DUMMY
//          DD DUMMY
//*
```

In the above example, out of the three datasets concatenated in IN1, the first one is overridden in the JCL and the rest is kept as that present in PROC.

Defining GDGs in a JCL

Generation Data Groups *GDGs* are group of datasets related to each other by a common name. The common name is referred as GDG base and each dataset associated with the base is called a GDG version.

For example, MYDATA.URMI.SAMPLE.GDG is the GDG base name. The datasets are named as MYDATA.URMI.SAMPLE.GDG.G0001V00, MYDATA.URMI.SAMPLE.GDG.G0002V00 and so on. The latest version of the GDG is referred as MYDATA.URMI.SAMPLE.GDG(0), previous versions are referred as (-1), (-2) and so on. The next version to be created in a program is refered as MYDATA.URMI.SAMPLE.GDG(+1) in the JCL.

Create/ Alter GDG in a JCL

The GDG versions can have same or different DCB parameters. An initial model DCB can be defined to be used by all versions, but it can be overridden when creating new versions.

```
//GDGSTEP1 EXEC PGM=IDCAMS
//SYSPRINT DD  SYSOUT=*
//SYSIN    DD  *
           DEFINE GDG(NAME(MYDATA.URMI.SAMPLE.GDG)   -
           LIMIT(7)                                  -
           NOEMPTY                                   -
           SCRATCH)
/*
//GDGSTEP2 EXEC PGM=IEFBR14
//GDGMODLD DD  DSN=MYDATA.URMI.SAMPLE.GDG,
//         DISP=(NEW,CATLG,DELETE),
//         UNIT=SYSDA,
//         SPACE=(CYL,10,20),
//         DCB=(LRECL=50,RECFM=FB)
//
```

In the above example, IDCAMS utility defines the GDG base in GDGSTEP1 with below parameters passed in the SYSIN DD statement:

- **NAME** specifies the physical dataset name of the GDG base.
- **LIMIT** specifies the maximum number of versions that the GDG base can hold.
- **EMPTY** uncataloges all the generations when the LIMIT is reached.
- **NOEMPTY** uncataloges the least recent generation.
- **SCRATCH** physically deletes the generation when it is uncataloged.
- **NOSCRATCH** do not delete the dataset, i.e., it can be referred using the UNIT and VOL parameters.

In GDGSTEP2, IEFBR14 utility specifies model DD parameters to be used by all versions.

IDCAMS can be used to alter the definition parameters of a GDG such as increasing LIMIT, changing EMPTY to NOEMPTY, etc., and its related versions using the SYSIN command is **ALTER MYDATA.URMI.SAMPLE.GDG LIMIT15 EMPTY**.

Delete GDG in a JCL

Using IEFBR14 utility, we can delete a single version of a GDG.

```
//GDGSTEP3   EXEC PGM=IEFBR14
//GDGDEL     DD  DSN=MYDATA.URMI.SAMPLE.GDG(0),
//           DISP=(OLD,DELETE,DELETE)
```

In the above example, the latest version of MYDATA.URMI.SAMPLE.GDG is deleted. Please note that the DISP parameter on normal job completion is coded as DELETE. Hence, the dataset is deleted when the job completes execution.

IDCAMS can be used to delete the GDG and its related versions using the SYSIN command **DELETE *MYDATA. URMI. SAMPLE. GDG* GDG FORCE/PURGE**.

- **FORCE** deletes the GDG versions and the GDG base. If any of the GDG versions are set with an expiration date which is yet to expire, then those are not deleted and hence the GDG base is retained.
- **PURGE** deletes the GDG versions and the GDG base irrespective of the expiration date.

Using GDG in a JCL

In the following example, the latest version of MYDATA.URMI.SAMPLE.GDG is used as input to the program and a new version of MYDATA.URMI.SAMPLE.GDG is created as the output.

```
//CNDSAMP JOB CLASS=6,NOTIFY=&SYSUID
//*
//STP01   EXEC PGM=MYCOBB
//IN1     DD DSN=MYDATA.URMI.SAMPLE.GDG(0),DISP=SHR
//OUT1    DD DSN=MYDATA.URMI.SAMPLE.GDG(+1),DISP=(,CALTG,DELETE)
//        LRECL=100,RECFM=FB
```

Here, if the GDG had been referred by the actual name like MYDATA.URMI.SAMPLE.GDG.G0001V00, then it leads to changing the JCL every time before execution. Using 0 and +1 makes it dynamically substitute the GDG version for execution.