

## UNIT 4

# The DD Statement.

### The DD statement

- Objectives.
- Purpose of the DD statement.
- Syntax.
- Parameters.
  - a. DSN
  - b. DISP
  - c. SPACE
  - d. VOL
  - e. UNIT
  - f. DCB
  - g. SYSOUT
- Temporary datasets
- Refer backs
- Special DD names
- Dataset concatenation

## **Objectives**

- Understand the purpose of the DD statement
- Learn the most important parameters of the DD statement
- Be able to create, use and manipulate datasets
- Code different DD statements depending on the requirement

### Purpose of the DD statement

The DD statement is required for the following reasons

1. To define the input and output datasets that are to be used by a particular Step
  2. To create new datasets if required by a Step
  3. To specify all the dataset attributes required for creating new datasets
  4. To manipulate and specify attributes for existing datasets so that they meet the requirements of the executing Step
  5. To provide input data other than dataset input to programs that are executing in the Step
  6. To direct the output created onto an output device or onto the spool
- Most DD statements appear after at least one EXEC statement. But, some special DD statements, as we shall see, appear before any EXEC statement in the Job.
  - All DD statements related to a particular EXEC statement should be coded before coding the next EXEC statement. The next set of DD statements can now be coded for the new EXEC statement

## Syntax

The DD statement accepts positional and keyword parameters.

```
//DDNAME DD DSN=X.Y.Z,DISP=NEW...
```

### Example:

```
//DDIN DD DSN=MAINUSR.COBOL.INPUT,DISP=OLD  
//DDOUT DD DUMMY
```

- There are many keyword parameters that can be coded along with a DD statement, as we shall see. However, there is only one positional parameter - 'DUMMY'. The explanation for this parameter is discussed later in the unit.
- Usually 'DUMMY' is the only parameter coded on a DD statement, but keyword parameters can follow. However if the keyword parameters are coded as the first parameters then there is no need to code a comma to signify the absence of a positional parameter.

### Parameter – DSN (Data Set Name)

**Type:** Keyword parameter

**Purpose:** To specify the name of the dataset

**Syntax:**

```
//DDNAME DD DSN=MAINUSR.X.Y
```

**Examples:**

```
//DDIN DD DSN=MAINUSR.COBOL.FILE  
//DDOUT DD DSN=MAINUSR.DB2.FILES(DCLGEN)
```

- 
- The DSN parameter is used to specify the name of the dataset, which is used by the Step. The Step will have a DD statement for each dataset used by the Program. The dataset may be used for input or output purposes as required by the program.
- If a program is reading two files and writing into one file then the JCL's EXEC statement will have three DD statements specifying the datasets along with the attributes required.
- DSN can be coded as DSNAME also.
- In the above examples DDIN is specifying a dataset by the name of MAINUSR.COBOL.FILE, which may be a physical sequential file (PS) or partitioned dataset (PDS).
- DDOUT DD statement is specifying a dataset where DCLGEN is a member of the PDS MAINUSR.DB2.FILES

## Parameter – DISP (Disposition)

**Type:** Keyword parameter that accepts positional parameters

**Purpose:** To specify

- ❑ The status of the dataset at the beginning of the step.
- ❑ What action to be performed on the dataset if the step completes normally.
- ❑ What action to be performed on the dataset if the step abends.

**Example:**

```
//JOBNAME      JOB      , ,CLASS=A
//STEP1        EXEC  PGM=SAMPLE1
//DD1          DD      DSN=MAINUSR.COBOL.INPUT,DISP=OLD
//DD2          DD      DSN=MAINUSR.COBOL.IN,DISP=( SHR,UNCATLG,DELETE)
//DD3          DD      DSN=MAINUSR.COBOL.OUT,DISP=(NEW,CATLG,DELETE),
//              VOL=SER=SYSDA,SPACE=(TRK,(1,1))
//STEP2        EXEC  PGM=SAMPLE2
//DDIN         DD      DSN=MAINUSR.VSAM.IN,DISP=SHR
//DDOUT        DD      DSN=MAINUSR.VSAM.OUT,DISP=MOD,
//              VOL=SER=SYSDA,SPACE=(TRK,(1,1))
```

- The above example is a two Step JOB. In the first Step, the program SAMPLE1 requires the use of three datasets specified by the ddnames DD1, DD2, and DD3.
- In the first DD statement (DD1) the DSN parameter names the physical dataset and DISP parameter states that the dataset already exists and that we want exclusive use of this dataset (DISP=OLD).
- In the second DD statement (DD2) the DISP parameter states to use the existing dataset for shared use and that the dataset is to be uncataloged after completion of the program. However if the program terminates abnormally we want to delete this dataset.
- In the third DD statement (DD3) the DISP parameter states that a new dataset is to be created. After completion of the program we want to catalog the dataset. However, if the program terminates abnormally than the dataset should be deleted.
- The Disposition parameter is used to specify if the dataset already exists or if it is going to be a new dataset. It also determines what is to be done to the dataset once the program terminates normally, and if it terminates abnormally for some reason.
- The DISP parameter refers to the entire dataset and not to members of a PDS.
- The default DISP parameter is **DISP=(NEW,DELETE,DELETE)**

### Parameter – DISP (Continued)

<b>Syntax:</b>	Status Before Step Begins	Action on Normal Completion	Action on Abend Completion
	▼	▼	▼
<b>DISP =</b> ( _____ , _____ , _____ )			
	NEW, OLD SHR MOD	CATLG, UNCATLG KEEP DELETE PASS	CATLG UNCATLG KEEP DELETE

- **NEW** – Dataset does not exist at the beginning of the step and is to be created. This parameter requires coding of some additional parameters like Space, Volume and Data Control Block (DCB). Code DISP=NEW when creating a new data set. DISP=NEW allows only one user to access the data set at a time.
- **OLD** – Dataset exists at the beginning of the step and should be opened for exclusive use by the step, i.e. no other process can access the dataset until this step finishes execution. Also, if records are being written to the dataset, the new records will overwrite existing records..
- Code DISP=OLD to reference an existing data set. DISP=OLD allows only one user to access the dataset at a time. The system places other users in a wait state. If DISP=OLD is coded in output, the system starts at the beginning of the data set and writes over existing data. The system writes an end-of-file (EOF) after writing the last record.
- **SHR** – Dataset exists at the beginning of the step and should be opened for this step in shared mode, i.e. any other process can also access the same dataset while the current step is executing but, only in shared mode. This means that no other process can code DISP=OLD for this dataset.
- Code DISP=SHR to reference an existing data set and allow other users to access the data set at the same time.
- **MOD** – If dataset exists, open it in exclusive mode (DISP=OLD) and, if records are being written to it, add the records at the end of the dataset. If the dataset does not exist then the status defaults to NEW.
- Code DISP=MOD to indicate records are to be appended to the logical end of an existing data set. If DISP=MOD is coded and the data set does not exist, the disposition is treated as DISP=NEW.



## Parameter - SPACE

**Type:** Keyword parameter that accepts positional parameters

**Purpose:** To specify the Space related requirements for new datasets

**Syntax:** SPACE=(unit,( p,s,d),RLSE)

Where:

Unit - Specifies the unit type that space will be allocated.

Values are:

TRK – requests allocation of storage for the new dataset in terms of tracks

CYL – requests allocation of storage for the new dataset in terms of cylinders

p - Numerical value that indicates the primary quantity of space to be allocated in terms of the unit specified

s - Numerical value that indicates the secondary quantity to be allocated once the primary quantity is exhausted

d - Numerical value that specifies the number of directory blocks for a PDS (see below)

Directory Blocks      Specifies the Directory Blocks the dataset is to have. Any value greater than zero creates a Partitioned Dataset. Coding a zero in this parameter or skipping it altogether creates a Physical Sequential dataset.

RLSE                      Releases any unused secondary space from the dataset for use by other processes once the step finishes execution.

### Example:

```
//DDOUT DD DSN=MAINUSR.COBOL.OUT,DISP=(NEW,CATLG),  
// SPACE=(TRK,(2,3,1)RLSE),VOL=SER=SYSDA
```

In the above example, MAINUSR.COBOL.OUT is a new partitioned dataset (PDS) to be created since the directory block is a value of one, and the space required is in terms of tracks. The primary quantity is two tracks and secondary is three tracks.

### Parameter - VOLUME

**Type:** Keyword parameter that accepts keyword and positional parameters

**Purpose:** To specify the Volume serial number on which the dataset exists or needs to be created.

**Syntax:**

VOL=SER=volume-serial-number

Volume serial number is the installation-dependent serial number of the volume on which the dataset resides or needs to be created.

or

VOL=REF=cataloged-dataset-name

When you want to specify the name of a cataloged dataset instead of specifying the volume-serial. Here, the dataset will be referenced from or created (depending on the disposition) on the same volume on which the cataloged dataset, mentioned in the parameter, resides.

**Example:**

```
//DDOUT DD DSN=MAINUSR.COBOL.OUT,DISP=(NEW,KEEP),  
//          SPACE=(TRK,(2,3,1)RLSE),VOL=SER=LP2WK1
```

- Volume serial number is an important parameter to be mentioned for datasets that are given the Disposition of KEEP and for those datasets that are uncataloged.
- For new datasets that are going to be cataloged there is no need to mention a volume serial number.
- When you retrieve a cataloged existing dataset there is no need to mention the volume serial number.
- The VOL parameter is a must while creating a dataset or accessing an uncataloged dataset.

## Parameter - UNIT

**Type:** Keyword parameter which accepts positional parameters.

**Purpose:** To specify what type of device the dataset resides on or should be created.

**Syntax:**

**UNIT=Generic-unit-name**  
                  **or**  
**UNIT=Unit-model-number**  
                  **or**  
**UNIT=/Machine-address**

Generic-unit-name:	SYSDA	any Direct Access Device
	TAPE	any Sequential Access Device
	SYSSQ	any Direct or Sequential Access device

Unit-model-number:	9345	} DASD
	3380	
	3390	
	3490	Tape

Machine-address: The installation-specific address of the device

**Example:**

```
//DDOUT DD DSN=MAINUSR.COBOL.OUT,DISP=(NEW,CATLG),  
// SPACE=(TRK,(2,3,1)RLSE),VOL=SER=LP2WK1,UNIT=SYSDA
```

### Parameter – DCB (Data Control Block )

**Type:** Keyword parameter that accepts keyword parameters

**Purpose:** DCB is used to tell the system about the characteristics of the dataset being referenced, such as the structure of the record and the Block size.

**Syntax:** `DCB=(LRECL=nnnn,BLKSIZE=nnnn,RECFM=record-format)`  
n – a numeric value

**LRECL** – The Logical Record Length - Specifies the maximum length of a record in the dataset. Values can range from 1 to 32760 bytes.

**BLKSIZE** – The Block Size - Specifies the maximum length of a block in the dataset. Values can range from 18 to 32760 bytes.

**RECFM** – The Record Format - Specifies the format of the records in the dataset i.e. fixed, variable, blocked, undefined etc.

record-format -	F	Fixed length
	FB	Fixed length Blocked
	V	Variable length
	VB	Variable length Blocked
	U	Undefined

**Example:**

```
//DDOUT DD DSN=MAINUSR.COBOL.OUT,DISP=(NEW,CATLG),  
// SPACE=(TRK,(2,3,1)RLSE),VOL=SER=LP2WK1,  
// DCB=(LRECL=80,RECFM=FB,BLKSIZE=8000)
```

For the RECFM parameter, some additional characters can be added at the end of the RECFM value. They are,

- A If the records contain a carriage-return control character in the first byte.
- M If the records contain machine-specific device control characters other than the carriage-return control character in the first byte.
- S If the records size may exceed the block size.
- T If the records may be bigger than the track capacity of the device.

Also you can code each of these subparameters separately without including them in the DCB parameter. However, if the DCB parameter is coded then all these subparameters, if coded, must appear within the DCB parameter.

## Parameter - SYSOUT

**Type:** Positional keyword parameter

**Purpose:** To send program output to the SYSOUT print spool

**Syntax:**

`SYSOUT=class value`

`or`

`SYSOUT=*` (refers to the MSGCLASS value on the JOB card).

**Examples:**

```
//DDOUT DD SYSOUT=A
//DDOUT1 DD SYSOUT=*
```

If coded, the SYSOUT parameter should appear immediately after the DD statement.

Use the SYSOUT parameter if you want to direct the output meant for any dataset to the print spool maintained by JES.

**Example:**

```
//SYSPRINT DD SYSOUT=*
```

Directs the output meant for dataset SYSPRINT to the systems output spool.

### Temporary Datasets

Temporary datasets are used as scratchpads for storing data only for the run of that step. After the end of the step, the temporary dataset is deleted unless PASSEd to the next step.

The default disposition for temporary datasets is (NEW,DELETE,DELETE)

A temporary data set is one that is created and deleted in the same job. There are three ways to request the creation of a temporary data set:

#### 1. Omit the DSN

```
//A DD UNIT=SYSDA,SPACE=(CYL,1)
```

In this case the system will create temporary data set having a 44-character name. The system builds date, time, and other information into this name to ensure the name is unique.

#### 2. Code a name beginning with && such as DSN=&&WORK

```
//B DD DSN=&&WORK,UNIT=SYSDA,SPACE=(CYL,1)
```

The system will create a temporary data set having a 44-character name. The system builds date, time, and other information into the name to ensure its uniqueness.

#### 3. Code a name beginning with & such as DSN=&TEMP

```
//C DD DSN=&TEMP,UNIT=SYSDA,SPACE=(CYL,1)
```

The system will create a temporary data set with a 44-character name. The system creates a unique name containing date, time and other information. Names prefixed by && are preferable to those prefixed by &. In a name of the form, DSN=&TEMP, the &TEMP can mean a temporary data set or a symbolic parameter (to be covered later).

## Temporary Datasets (Continued)

To avoid ambiguity use the form, DSN=&&TEMP.

If you use two temporary data sets of the forms, DSN=&&WORK and DSN=&WORK, in the same job stream, the system will interpret these to be the same data set.

You can create temporary datasets by omitting the DSN parameter .

To create temporary datasets:

```
DSN=&Tempname  
      or  
DSN=&&Tempname
```

Where Tempname is any name of 8 characters or less.

### Example:

```
DSN=&&TEMPFILE
```

You can reference the dataset by coding &&TEMPFILE but, internally the system will generate a long dataset name of 5 qualifiers which include User ID, Timestamp etc. to maintain uniqueness.

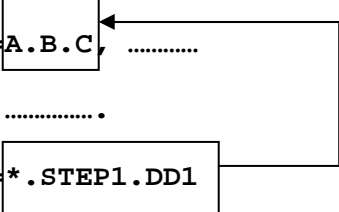
- To retain the dataset even after the current step ends, the temporary dataset has to be passed to the next step by coding PASS in the second sub-option of the DISP parameter.
- It is not advisable to code the name preceded by a single ampersand (&) as the name might be substituted by the value of a symbolic-parameter (covered in later units) of the same name.

### Referback

- You can use refer backs if you don't want to explicitly code the value for parameters.
- The presence of the '\*' indicates that a backward reference is being used.

```
DSN = *.DDNAME
      *.STEPNAME.DDNAME
```

```
//STEP1 EXEC .....
//DD1 DD DSN=A.B.C .....
//STEP2 EXEC .....
//DD2 DD DSN=*.STEP1.DD1 .....
//
```



#### Syntax:

```
Parameter=*.STEPNAME.DDNAME
           ↓
       Referback indicator
```

- Many parameters in JCL statements can use a backward reference to fill in information. A backward reference (or refer back) is a reference to an earlier statement in the job or in a cataloged or instream procedure called by a job step.

A backward reference has the form:

**\*.NAME OR \*.DDNAME**

**\*.STEPNAME.NAME OR \*.STEPNAME.DDNAME**

**\*.STEPNAME.PROCSTEPNAME.NAME OR \*.STEPNAME.PROCSTEPNAME.DDNAME**



## Special DD names

You can use any name for your DD statements but there are certain names that have a special meaning to the system or to the program(s) that are being executed.

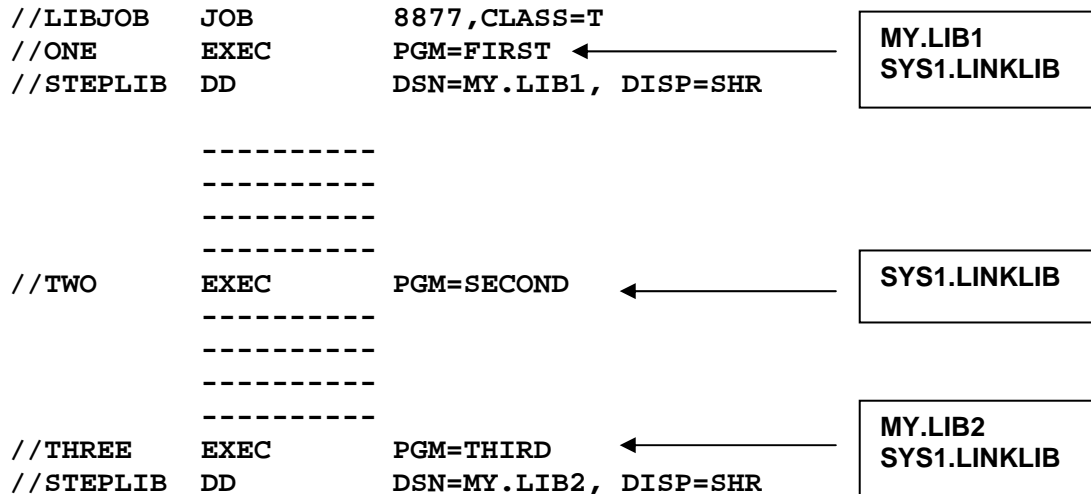
Below are some of the most common DD names that have special meaning on most installations.

- **STEPLIB**     Default search library PDS for the load-module being executed in the step
- **JOBLIB**     Default search library PDS for the load-modules of all the steps in the Job
- **SYSPRINT**   Default output dataset for printable output
- **SYSOUT**     Default output dataset for system messages
- **SYSIN**       Default instream input dataset for most programs
- Note that if a DD name has a special meaning to one program, it may mean nothing at all to another program.

### STEPLIB

The //STEPLIB DD statement alters the normal program search strategy to locate the program being executed.

When included in a step, the library named on the //STEPLIB DD statement is searched before the system searches the default library, SYS1.LINKLIB



- You are permitted to concatenate libraries to a STEPLIB DD statement. If you do so the libraries are searched in their concatenation order.
- STEPLIB overrides JOBLIB in the step, which has the STEPLIB statement.

## JOBLIB

```
//ZEBRA    JOB    66,CLASS=T
//JOBLIB   DD     DSN=MY.LIB,DISP=SHR
//ONE      EXEC   PGM=FIRST

//TWO      EXEC   PGM=SECOND
           -----
           -----

//THREE    EXEC   PGM=THIRD
```

- Code the //JOBLIB DD statement to identify a private library/libraries the system is to search for each program in the job.
- The //JOBLIB DD statement must be coded before the first EXEC statement
- You are permitted to concatenate libraries to a JOBLIB DD statement. If you do so the libraries are searched in their concatenation order.

### Dataset Concatenation

```
//INDATA DD DSN=USERID.INPUT.FILE1,DISP=SHR
//      DD DSN=USERID.INPUT.FILE2,DISP=SHR
//      DD DSN=USERID.INPUT.FILE3,DISP=SHR
//OUTDATA DD DSN=USERID.OUTPUT.DATA1,DISP=OLD
//      DD DSN=USERID.OUTPUT.DATA2,DISP=OLD
//      DD DSN=USERID.OUTPUT.DATA3,
//          DISP=(NEW,CATLG,DELETE),
//          SPACE=(TRK,(1,1),RLSE),LRECL=80,
//          BLKSIZE=6160,RECFM=FB,
//          VOL=SER=LP1WK1,UNIT=SYSDA
```

- Concatenation is the method of combining 2 or more datasets and making them behave as a single file from the point of view of the program.
- In case of input datasets records will be read in from files in the same order as specified in the concatenation i.e. start reading from the first file, once there are no more records in the first file, continue with the second file and so on until the last file is read. Only then will the system encounter the end-of-file condition.
- In case of output datasets, writing will start from the first dataset. Once all the space in the first dataset is exhausted, the next dataset in the concatenation order will be written to and, so on.

- *Unit 4 Exercises*

1. A \_\_\_\_\_ dataset is created when you omit the DSN parameter.
2. DISP= \_\_\_\_\_ is coded for a dataset already existing & you want exclusive use of the dataset.
3. DISP= \_\_\_\_\_ is coded for a dataset already existing & you want to append into it.
4. DISP= \_\_\_\_\_ is coded for a dataset which is not existing, but you want to use it later on without actually knowing its location.
5. Default disposition is DISP= \_\_\_\_\_.
6. The \_\_\_\_\_ parameter is necessary to locate an already existing dataset whose entry is not in any catalog.
7. Code a complete DD statement for a new dataset with the following criteria:
  - a. The dataset name should be userid.JCL.CNTL
  - b. The dataset is new and should also be cataloged
  - c. Place the dataset on SYSDA
  - d. The record length should be fixed at 100, with a blocksize of 1000.

### *Unit 4 Lab Exercises*

*Logon to TSO/ISPF and perform the following exercises. Wherever you see “userid” in lower case, substitute your valid security userid.*

#### **Create a Physical Sequential Dataset**

1. In your PDS called ‘userid.JCL.CNTL’, create a new member called JOBTEST3.
2. Copy the JOB card statement from JOBTEST2 and change the jobname to userid3.
3. Create //STEP1 to execute the IEFBR14 utility program
4. In the DD statement, create a new physical sequential dataset called userid.JCL.SAMPLE1.
  - The allocation for this dataset should be in tracks.
  - Primary quantity to be used is two tracks
  - Secondary quantity should be one track
  - Records should be a fixed length of 80 bytes.
  - The dataset should be cataloged upon successful completion of the step. Otherwise it should be deleted.
5. Run the job and view its results. Fix any errors and resubmit it until it runs successfully.

#### **Create a PDS using IEFBR14**

6. In your PDS called ‘USERID.JCL.CNTL’, create a new member called JOBTEST4.
7. Copy the JOB card statement from JOBTEST3 and change the jobname to userid4.
8. Create //STEP1 to execute the IEFBR14 utility program
9. In the DD statement, create a new PDS called userid.JCL.PDS
  - Allocation for this dataset should be in cylinders
  - Primary allocation should be two & secondary should be ten
  - The dataset should have fixed length records of 80 bytes.
  - The dataset should be cataloged upon successful completion of the step. Otherwise it should be deleted.