UNIT 6

# Procedures

# Procedures

- Objectives
- What is a Procedure?
- Instream and Cataloged Procedures
- The PROC and PEND statements
- Coding an Instream Procedure
- Coding a Cataloged Procedure
- The JCLLIB statement
- Modifying DD statements of a Procedure
- Modifying EXEC statements of a Procedure
- Symbolic parameters
- The SET statement
- The INCLUDE statement

# Objectives

- Understand the concept of a Procedure
- Code instream procedures
- Create your own Procedure-Library with cataloged Procedures
- Use Symbolic parameters to code standard Procedures
- Create copy libraries for frequently used JCL code

# What is a Procedure?

A Procedure is a JCL consisting of multiple executable steps but no JOB statement. This means that a Procedure consists of EXEC statements along with their respective DD statements.

A Procedure would generally be written for a set of programs that you want executed a number of times. Instead of coding these Steps repeatedly, you would make these Steps a part of a Procedure and would then execute the procedure as many times as required.

There are two types of procedures: Instream Procedures and Cataloged Procedures.

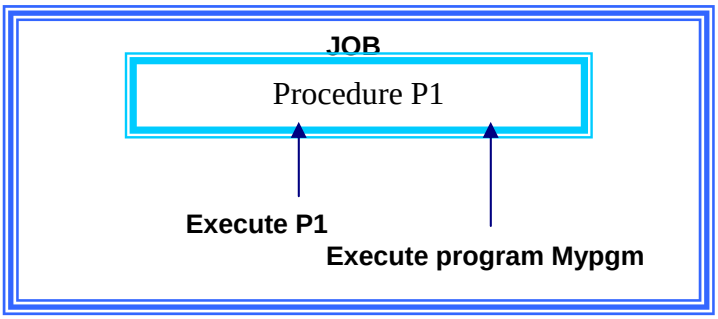



Figure 6-1

Benefits of using procedures:

1. Procedure saves time by reducing the time required to code JCL.
2. They save library storage by eliminating duplicate JCL.
3. They reduce JCL errors by providing access to debugged JCL.

To execute a Procedure, an EXEC statement is required along with the Procedure-name.

**Example:**

```
//JOBNAME  JOB   NOTIFY=&USERID
//STEP     EXEC  PROC=procedure name
```

A Procedure name is a positional parameter of the EXEC statement and therefore can be coded without the ‘PROC=‘ syntax, such as:

```
//JOBNAME  JOB   NOTIFY=&USERID
//STEP     EXEC  procedure name
```

**Note** A Procedure can, in turn, call another procedure. A maximum of 15 such levels of nesting are allowed.

# Cataloged Procedures

A procedure which is part of a library is called a CATALOGED PROCEDURE.

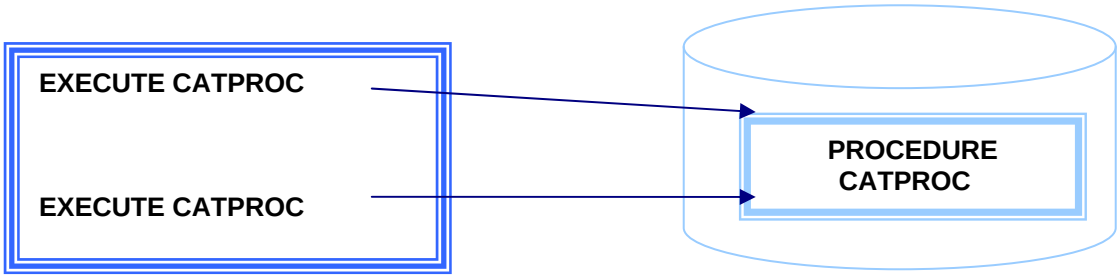



Figure 6-2

Since it a member of a Partitioned dataset, it can be used by multiple Jobs.

It is not mandatory to have PROC statement in a cataloged procedure to signify the beginning of the procedure. However, as we will see, this statement is coded to contain the default values of symbolic parameters.

The PEND statement is also not mandatory in a cataloged procedure.

Procedures cannot contain

- JOB statements
- DD * statements
- DD DATA statements
- Delimiter statements (‘/*’ followed by 78 blanks)
- Null statements (‘//’ followed by 78 blanks)
- Non-JCL statements (for example, JES or utility control statement)

# Instream Procedure

When you code the procedure in the job input stream, it is called an Instream Procedure

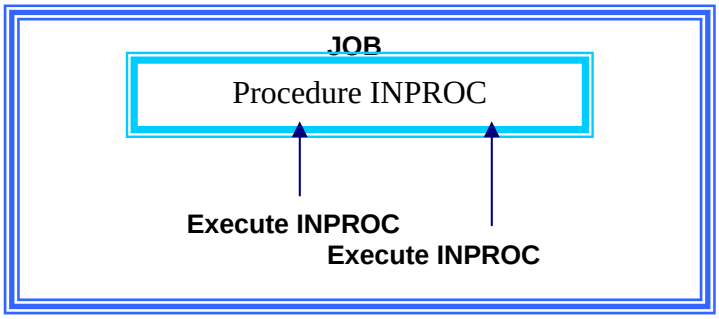



Figure 6-3

An Instream Procedure begins with a PROC statement and ends with a PEND statement.

An Instream Procedure can begin after coding the JOB statement.

Instream Procedures can only be executed by the Jobs in which they are coded.

# The PROC and PEND statements in Instream Procedures

```
//JOBNAME   JOB NOTIFY=USERID
//CREPROC   PROC
//STEP1     EXEC PGM=IEFBR14
//DD1       DD   DSN=MAINUSR.JCL1.CNTL,
//                    DISP=(NEW,CATLG,DELETE),
//                    VOL=SER=LP2WK1,
//                    UNIT=SYSDA,SPACE=(TRK,(1,1,1),RLSE),
//                    DCB=(BLKSIZE=800,LRECL=80,RECFM=FB)
//SYSPRINT  DD SYSOUT=*
//          PEND
//*
//JCLSTEP   EXEC PROC=CREPROC
//SYSIN     DD   DUMMY
//*
```

The PROC statement signifies the beginning of a Procedure.

- An Instream Procedure must begin with a PROC statement.
- The name of the procedure is determined by the LABEL field of the PROC statement (CREPROC above). This is the Procedure name that will be used by the JOB when executing it.

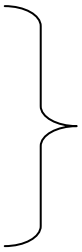
The PEND statement specifies the end of a Procedure definition.

# Coding an Instream Procedure

```
//MYJOB    JOB NOTIFY=USERID
//MYPROC   PROC
//STEP1    EXEC PGM=IEFBR14
//SYSPRINT DD SYSOUT=*
//         PEND
//*
/JOBSTEP EXEC PROC=MYPROC
//SYSIN  DD DUMMY
```

***At Runtime:***

```
//MYJOB    JOB
//*JOBSTEP EXEC PROC=MYPROC
//*
//STEP1    EXEC PGM=IEFBR14
//SYSPRINT DD SYSOUT=*
//SYSIN    DD DUMMY
```



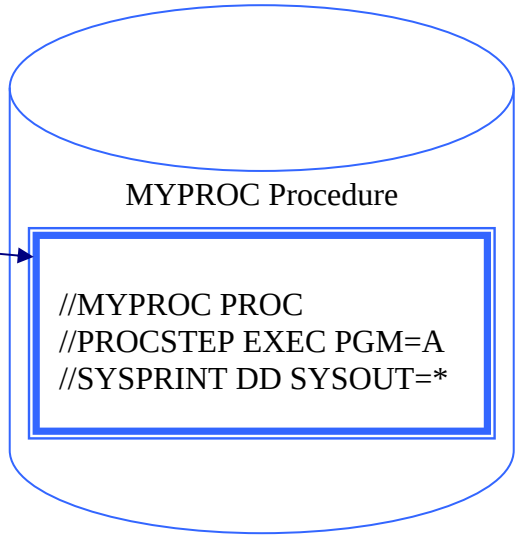
**This is what the system visualizes**

In the example, the instream procedure begins with the PROC statement and ends with the PEND statement.

The procedure name is ‘MYPROC’ which will be used to invoke the procedure in the EXEC statement.

# Coding a Cataloged Procedure

```
//MYJOB   JOB
//STEP1   EXEC MYPROC
//SYSIN   DD DUMMY
```




**During Run-time**

```
//MYJOB    JOB
//PROCSTEP EXEC PGM=A
//SYSPRINT DD SYSOUT=*
//SYSIN    DD DUMMY
```

Figure 6-4

Note that the PEND statement is not necessary for a Cataloged Procedure

The member name in which the cataloged procedure is coded becomes the name of the procedure, which will be used to invoke it in a Job.

A procedure can be cataloged by placing it in one of three types of proclibs:

- SYS1.PROCLIB – IBM-supplied system procedure library.
- System PROCLIBs - defined by an installation.
- A user-defined PROCLIB – OS/390 or MVS/ESA SP V4 or Higher

Use the IEBUPDTE utility or ISPF/PDF to add a procedure to a proclib or modify a procedure. (IEBUPDTE is described in the next topic)

A PROC statement in a cataloged procedure is optional and its label is also optional. Its only function is to contain default symbolic overrides.

# The JCLLIB Statement

The JCLLIB statement identifies a library (PDS) that contains catalogued procedures. Placing the JCLLIB statement in the JCL tells the JCL where to locate the procedures being executed in the job.

Example:

**Procedure Library: `MAINUSR.JCL.CNTL(JCLPROC)`**

```
//JCLPROC  PROC
//STEP1    EXEC   PGM=IEFBR14
//DD1      DD      DSN=MAINUSR.PROC1.PROC,
//                 DISP=(NEW,CATLG,DELETE),VOL=SER=LP2WK1,
//                 UNIT=SYSDA,SPACE=(TRK,(1,1,1),RLSE),
//                 DCB=(BLKSIZE=800,LRECL=80,RECFM=FB)
//SYSPRINT DD     SYSOUT=*
//         PEND
//*
```

**JCL**

```
//CATGC    JOB     A123,'SUSAN JOHN',.....
//DD1      JCLLIB   ORDER=(MAINUSR.JCL.CNTL,..)
//CREAT    EXEC    PROC=JCLPROC
//SYSIN    DD      DUMMY
//*
```

# Rules for overriding a PROC

Parameters on the EXEC and DD statements of a PROC can be added, modified or nullified with the use of ***JCL overrides***.

Procedures are executed when invoked from a Job, in its EXEC statement. The programs coded in the procedure are executed as they are without any changes. However, JCL overrides allow you to change the parameters given in the EXEC statement and the DD statements of the Procedure. These changes are limited for that run of the JCL only.

**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 must be completed before overriding parameters in a subsequent step.

**RULES for DD statement overriding**

- To override any parameters in a DD statement an independent DD statement must be coded in the following format:
- //stepname.ddname DD overriding parameters
- The sequence of overriding DD statements must be the same as the sequence of the corresponding overridden statements.
- An additional DD statement must be the last one coded in a step's overriding statements.

# Overriding EXEC statements of a Procedure

```
//MYJOB  JOB  NOTIFY=USERID
//MYPROC PROC
//STEP1  EXEC PGM=IEBGENER,TIME=NOLIMIT,REGION=4M
//SYSUT1 DD DUMMY
//SYSUT2 DD DUMMY
//       PEND
//STEP2 EXEC MYPROC,TIME.STEP1=10,REGION.STEP1=

AT RUNTIME

//MYJOB  JOB NOTIFY=USERID
//STEP1  EXEC PGM=IEBGENER,TIME=10
//SYSUT1 DD DUMMY
//SYSUT2 DD DUMMY
```

The syntax for overriding Procedure EXEC parameters is

**Parameter.Procedure-stepname=value**

In the example above:

- The value of the TIME parameter has been changed from NOLIMIT to 10
- The REGION parameter has been nullified

# Overriding DD statements of a Procedure

```
//MYJOB  JOB
//MYPROC PROC
//STEP1  EXEC PGM=IEFBR14
//DD1    DD  DSN=MAINUSR.JCL.PDS,DISP=SHR,VOL=SER=LP1WK1,
//           LRECL=80
//       PEND
//STEP2 EXEC MYPROC
//STEP1.DD1 DD VOL=SER=LP2WK1,LRECL=
//STEP1.DD2 DD DSN=MAINUSR.COPYLIB,DISP=SHR

AT RUNTIME

//MYJOB JOB
//STEP1   EXEC PGM=IEFBR14
//DD1     DD   DSN=MAINUSR.JCL,PDS,DISP=SHR,VOL=SER=LP2WK1
//DD2     DD   DSN=MAINUSR.COPYLIB,DISP=SHR
```

The syntax for overriding Procedure DD parameters is

```
//PROC-STEPNAME.DDNAME DD Parameter-modifications
```

In the above example

- In DD1, LP1WK1 was changed to LP2WK1
- The LRECL parameter was nullified (discarded)
- A new DD statement DD2 was added

## Symbolic parameters

A symbolic parameter can be coded in place of any parameter as part of an EXEC statement or DD statement. Symbolic parameters are variables, which hold values that can be changed when the Procedure is called.

A symbolic parameter is a name preceded by an ampersand (&) and can be a maximum of 8 characters long.

The default values for the symbolic parameter can be coded in the PROC statement. To override the default parameter you will have to code the values for the symbolic parameter in the EXEC statement that invokes the procedure.

Symbolic overrides can be used only when symbolic parameters have been coded inside the procedure.

```
//MYPROC PROC A=LP2WK1,B=SYSDA
//STEP1  EXEC PGM=IEFBR14
//DD1    DD DSN=MAINUSR.ABC.INPUT,DISP=SHR,
//          VOL=SER=&A,
//          UNIT=&B
//       PEND
//STEPX     EXEC MYPROC,A=LP1WK1,B=SYSSQ

AT RUNTIME

//STEP1 EXEC PGM=IEFBR14
//DD1     DD DSN=MAINUSR.ABC.INPUT,DISP=SHR,
//              VOL=SER=LP1WK1,
//              UNIT=SYSSQ
```

In the above example, A and B are symbolic parameters. Instead of hard-coding the values for the VOL and UNIT parameters, they have been assigned the values contained in the symbolic parameters.

In case we do not change the value of the Symbolic parameters when the Procedure is called, the default values specified at the Procedure-declaration statement (PROC) are taken.

# The SET statement

The SET statement is another way of assigning values to Symbolic parameters.

```
//MYPROC PROC A=LP2WK1,B=SYSDA
//STEP1  EXEC PGM=IEFBR14
//DD1    DD   DSN=MAINUSR.INPUT.FILE1,DISP=SHR,
//               VOL=SER=&A,
//               UNIT=&B
//       PEND
//SET1   SET A=LP1WK1
//SET2   SET B=SYSSQ
//STEPX  EXEC MYPROC

AT RUNTIME

//STEP1 EXEC PGM=IEFBR14
//DD1 DD DSN= MAINUSR.INPUT.FILE1,DISP=SHR,
//           VOL=SER=LP1WK1,
//           UNIT=SYSSQ
```

The SET statement can appear anywhere in a JCL between the JOB statement and the first point where a SET –statement-assigned symbolic parameter is referenced.

# The INCLUDE statement

The INCLUDE statement allows you to copy statements from any member of the PDS(s) listed in the JCLLIB statement.

Similar to the way PROCs are used, INCLUDE allows you to code a single set of JCL statements that you can use in multiple jobs.

**Example:**

```
//MYJOB JOB
//DD1   JCLLIB  ORDER=COMMOM.PDS
//STEP1 EXEC    PGM=MYPGM
//INDD  DD      DSN=A.B.C,DISP=SHR
//INC1  INCLUDE MEMBER=COMMON

AT RUNTIME

//MYJOB    JOB
//DD1      JCLLIB ORDER=COMMON.PDS
//STEP1    EXEC   PGM=MYPGM
//INDD     DD     DSN=A.B.C,DISP=SHR
//*INC1 INCLUDE MEMBER.COMMON
//SYSPRINT DD     SYSOUT=*
//SYSIN    DD     DUMMY
//SYSOUT   DD     SYSOUT=*
```

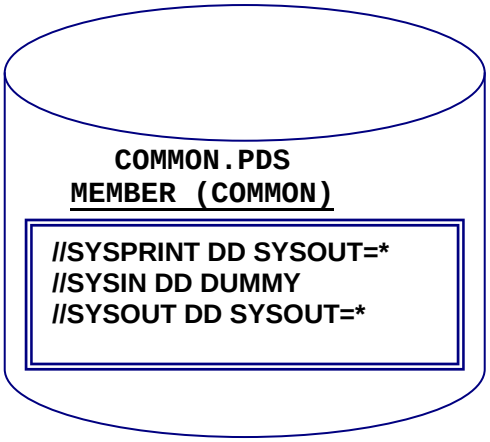



Figure 6-5

## *Unit 6 Exercises*

1. **Two statements that are mandatory for an instream procedure are _____ and _____.**

2. **The name of an instream procedure is specified in the LABEL field of the _____ statement.**

3. **The _____ statement is optional in cataloged procedures but is used to code default values of symbolic parameters.**

4. **To invoke a cataloged procedure you have to code the _____ name of the library that contains it.**

5. **(T/ F) All EXEC statement overrides should be completed before DD statement overrides**
6. **(T/F) A procedure can be a combination of statements as well as of symbolic parameters**

## *Unit 6 Lab Exercises*

*Logon to TSO/ISPF and perform the following exercises. Wherever you see "userid" or "USERID", substitute your valid security userid.*

1. **Instream Procedure.**

- **Create a new PDS called USERID.PROCS.PDS. Copy a member from USERID.JCL.CNTL to this PDS in the first run. In the second run copy the member again with a different name.**

- **You should code two steps to achieve the above and both steps will be included in your Procedure.**
- **Your procedure should consist of symbolic parameters enabling you to change certain parameters during different runs. The changes are made on the EXEC statement calling your Procedure.**

- **In STEP1 you will need to alter the disposition parameter for different executions of the procedure and in STEP 2 the member name will change each time you execute your Procedure.**

- **Code the following in your Procedure.**

- **STEP1 shall use the program IEFBR14 to create the data set USERID.PROCS.PDS. It should reside on SYSDA, which is assigned to SYSDA. The record length is 80 bytes and the block size is 3120 bytes. The record format is FB. Give the data set 1 track Primary, 1 track Secondary and 1 Directory Block.**

- **There should be a SYSPRINT DD statement taking the message output to the same Class as stated in the JOB statement.**

- **STEP 2 shall use the program IEBGENER to copy a member from USERID.JCL.CNTL to USERID.PROCS.PDS.**

- **Catalogued Procedure.**

- **STEP 1**

- **Save the above instream procedure as a member in a PDS and use the same PDS as the library for the catalogue procedure. For example**
- **Execute the Procedure ASMHCL, which resides in X.X.PROCLIB.**

- **Override the following DD statements in the procedure.**

- **All DD statements referring to SYSOUT=A should instead refer to SYSOUT=*.**
- **DD statements having UNIT should be changed to UNIT=SYSDA.**
- **The SYSLMOD DD should refer to X.X.LOAD. This data set is catalogued.**
- **Add a SYSIN DD statement to the first step referring to a catalogued dataset named X.X.ASM (COPYASM). This data set is only to be read.**
- **Add a SYSIN DD statement to second step followed by NAME TEST(R). This will tell the linkage program the name of the module and that the REPLACE option is in use.**
- **STEP 2**
- **Execute a program called TEST (Since the data set where the program TEST is located is not known to the system, you must add a special DD statement in order that this dataset can be found. The program TEST should be in X.X.LOAD).**
- **Add a SYSPRINT DD statement for SYSOUT=*.**