JCL - PROCEDURES

http://www.tutorialspoint.com/jcl/jcl_procedures.htm Copyright © tutorialspoint.com

The **JCL Procedures** are set of statements inside a JCL grouped together to perform a particular function. Usually, the fixed part of the JCL is coded in a procedure. The varying part of the Job is coded within the JCL.

You can use a procedure to achieve parallel execution of a program using multiple input files. A JCL can be created for each input file, and a single procedure can be called simultaneously by passing the input file name as a symbolic parameter.

Syntax

Following is the basic syntax of a JCL procedure definition:

```
//*
//Step-name EXEC procedure name
```

The contents of the procedure are held within the JCL for an instream procedure. The contents are held within a different member of the base library for a cataloged procedure. This chapter is going to explain two types of procedures available in JCL and then finally we will see how we can nest various procedures.

Instream Procedure

When the procedure is coded within the same JCL member, it is called an Instream Procedure. It should start with a PROC statement and end with a PEND statement.

```
//SAMPINST JOB 1,CLASS=6,MSGCLASS=Y,NOTIFY=&SYSUID
//*
//INSTPROC   PROC                //*START OF PROCEDURE
//PROC1   EXEC PGM=SORT
//SORTIN DD DSN=&DSNAME,DISP=SHR
//SORTOUT DD SYSOUT=*MYINCL
//SYSOUT DD SYSOUT=*
//SYSIN  DD DSN=&DATAC LRECL=80
//           PEND               //*END OF PROCEDURE
//*
//STEP1      EXEC INSTPROC,DSNME=MYDATA.URMI.INPUT1,
//           DATAC=MYDATA.BASE.LIB1(DATA1)
//*
//STEP2      EXEC INSTPROC,DSNME=MYDATA.URMI.INPUT2
//           DATAC=MYDATA.BASE.LIB1(DATA1)
//*
```

In the above example, the procedure INSTPROC is called in STEP1 and STEP2 using different input files. The parameters DSNAME and DATAC can be coded with different values while calling the procedure and these are called as **symbolic parameters**. The varying input to the JCL such as file names, datacards, PARM values, etc., are passed as symbolic parameters to procedures.

While coding symbolic parameters, do not use KEYWORDS, PARAMETERS or SUB-PARAMETERS as symbolic names. Example: Do not use TIME=&TIME but yes you can use TIME=&TM and it is assumed as a right way of coding symbolics.

User-defined symbolic parameters are called **JCL Symbols**. There are certain symbols called **system symbols**, which are used for logon job executions. The only system symbol used in batch jobs by normal users is **&SYSUID** and this is used in the NOTIFY parameter in the JOB statement.

Cataloged Procedure

When the procedure is separated out from the JCL and coded in a different data store, it is called a **Cataloged Procedure**. A PROC statement is not mandatory to be coded in a cataloged procedure. Following is an example of JCL where it's calling CATLPROC procedure:

```
//SAMPINST JOB 1,CLASS=6,MSGCLASS=Y,NOTIFY=&SYSUID
//*
//STEP EXEC CATLPROC,PROG=CATPRC1,DSNME=MYDATA.URMI.INPUT
//          DATAC=MYDATA.BASE.LIB1(DATA1)
```

Here, the procedure CATLPROC is cataloged in MYCOBOL.BASE.LIB1. PROG,DATAC and DSNAME are passed as symbolic parameters to the procedure CATLPROC.

```
//CATLPROC PROC PROG=,BASELB=MYCOBOL.BASE.LIB1
//*
//PROC1     EXEC PGM=&PROG
//STEPLIB   DD DSN=&BASELB,DISP=SHR
//IN1       DD DSN=&DSNAME,DISP=SHR
//OUT1      DD SYSOUT=*
//SYSOUT    DD SYSOUT=*
//SYSIN     DD DSN=&DATAC
//*
```

Within the procedure, the symbolic parameters PROG and BASELB are coded. Please note that the PROG parameter within the procedure is overridden by the value in the JCL and hence PGM takes the value CATPRC1 during execution.

Nested Procedures

Calling a procedure from within a procedure is called a **nested procedure**. Procedures can be nested up to 15 levels. The nesting can be completely in-stream or cataloged. We cannot code an instream procedure within a cataloged procedure.

```
//SAMPINST JOB 1,CLASS=6,MSGCLASS=Y,NOTIFY=&SYSUID
//*
//SETNM     SET DSNM1=INPUT1,DSNM2=OUTPUT1
//INSTPRC1  PROC               //* START OF PROCEDURE 1
//STEP1        EXEC PGM=SORT,DISP=SHR
//SORTIN       DD DSN=&DSNM1,DISP=SHR
//SORTOUT      DD DSN=&DSNM2,DISP=(,PASS)
//SYSOUT       DD SYSOUT=*
//SYSIN        DD DSN=&DATAC
//*
//STEP2        EXEC PROC=INSTPRC2,DSNM2=MYDATA.URMI.OUTPUT2
//          PEND               //* END OF PROCEDURE 1
//*
//INSTPRC2  PROC               //* START OF PROCEDURE 2
//STEP1        EXEC PGM=SORT
//SORTIN       DD DSN=*.INSTPRC1.STEP1.SORTOUT
//SORTOUT      DD DSN=&DSNM2,DISP=OLD
//SYSOUT       DD SYSOUT=*
//SYSIN        DD DSN=&DATAC
//          PEND               //* END OF PROCEDURE 2
//*
//JSTEP1    EXEC INSTPRC1,DSNM1=MYDATA.URMI.INPUT1,
//          DATAC=MYDATA.BASE.LIB1(DATA1)
//*
```

In the above example, the JCL calls the procedure INSTPRC1 in JSTEP1 and procedure INSTPRC2 is being called within the procedure INSTPRC1. Here, the output of INSTPRC1 *SORTOUT* is passed as input *SORTIN* to INSTPRC2.

> *A **SET statement** is used to define commonly used symbolics across job steps or procedures. It initializes the previous values in the symbolic names. It has to be defined before the first use of the symbolic names in the JCL.*

Let's have a look at the below description to understand a little more about the above program:

- SET parameter initializes DSNM1=INPUT1 and DSNM2=OUTPUT1.
- When INSTPRC1 is called in JSTEP1 of JCL, DSNM1=MYDATA.URMI.INPUT1 and DSNM2=OUTPUT1., i.e., the value initialized in SET statement is reset with the value set in any of the job step/procedures.
- When INSTPRC2 is called in STEP2 of INSTPRC1, DSNM1=MYDATA.URMI.INPUT1 and DSNM2=MYDATA.URMI.OUTPUT2.

Loading [MathJax]/jax/output/HTML-CSS/fonts/TeX/fontdata.js