JCL - CONDITIONAL PROCESSING

The Job Entry System uses two approaches to perform conditional processing in a JCL. When a job completes, a return code is set based on the status of execution. The return code can be a number between 0 *successfulexecution* to 4095 *non − zeroshowserrorcondition*. The most common conventional values are:

- 0 = Normal - all OK
- 4 = Warning - minor errors or problems.
- 8 = Error - significant errors or problems.
- 12 = Severe error - major errors or problems, the results should not be trusted.
- 16 = Terminal error - very serious problems, do not use the results.

A job step execution can be controlled based on the return code of the previous step*s* using the **COND** parameter and **IF-THEN-ELSE** construct, which has been explained in this tutorial.

COND parameter

A **COND** parameter can be coded in the JOB or EXEC statement of JCL. It is a test on the return code of the preceding job steps. If the test is evaluated to be true, the current job step execution is bypassed. Bypassing is just omission of the job step and not an abnormal termination. There can be at most eight conditions combined in a single test.

Syntax

Following is the basic syntax of a JCL COND Parameter:

```
COND=(rc,logical-operator)
or
COND=(rc,logical-operator,stepname)
or
COND=EVEN
or
COND=ONLY
```

Here is the description of parameters used:

- **rc** : This is the return code
- **logical-operator** : This can be GT *GreaterThan*, GE *GreaterthanorEqualto*, EQ *Equalto*, LT *LesserThan*, LE *LesserthanorEqualto* or NE *NotEqualto*.
- **stepname** : This is the job step whose return code is used in the test.

Last two conditions *a* COND=EVEN and *b* COND=ONLY, have been explained below in this tutorial.

The COND can be coded either inside JOB statement or EXEC statement, and in both the cases, it behaves differently as explained below:

COND inside JOB statement

When COND is coded in JOB statement, the condition is tested for every job step. When the condition is true at any particular job step, it is bypassed along with the job steps following it. Following is an example:

```
//CNDSAMP JOB CLASS=6,NOTIFY=&SYSUID,COND=(5,LE)
//*
//STEP10 EXEC PGM=FIRSTP
//* STEP10 executes without any test being performed.
```

```
//STEP20 EXEC PGM=SECONDP
//* STEP20 is bypassed, if RC of STEP10 is 5 or above.
//* Say STEP10 ends with RC4 and hence test is false.
//* So STEP20 executes and lets say it ends with RC16.

//STEP30 EXEC PGM=SORT
//* STEP30 is bypassed since 5 <= 16.
```

COND inside EXEC statement

When COND is coded in EXEC statement of a job step and found to be true, only that job step is bypassed, and execution is continued from next job step.

```
//CNDSAMP JOB CLASS=6,NOTIFY=&SYSUID
//*
//STP01 EXEC PGM=SORT
//* Assuming STP01 ends with RC0.

//STP02 EXEC PGM=MYCOBB,COND=(0,EQ,STP01)
//* In STP02, condition evaluates to TRUE and step bypassed.

//STP03 EXEC PGM=IEBGENER,COND=((10,LT,STP01),(10,GT,STP02))
//* In STP03, first condition fails and hence STP03 executes.
//* Since STP02 is bypassed, the condition (10,GT,STP02) in
//* STP03 is not tested.
```

COND=EVEN

When COND=EVEN is coded, the current job step is executed, even if any of the previous steps abnormally terminate. If any other RC condition is coded along with COND=EVEN, then the job step executes if none of the RC condition is true.

```
//CNDSAMP JOB CLASS=6,NOTIFY=&SYSUID
//*
//STP01 EXEC PGM=SORT
//* Assuming STP01 ends with RC0.

//STP02 EXEC PGM=MYCOBB,COND=(0,EQ,STP01)
//* In STP02, condition evaluates to TRUE and step bypassed.

//STP03 EXEC PGM=IEBGENER,COND=((10,LT,STP01),EVEN)
//* In STP03, condition (10,LT,STP01) evaluates to true,
//* hence the step is bypassed.
```

COND=ONLY

When COND=ONLY is coded, the current job step is executed, only when any of the previous steps abnormally terminate. If any other RC condition is coded along with COND=ONLY, then the job step executes if none of the RC condition is true and any of the previous job steps fail abnormally.

```
//CNDSAMP JOB CLASS=6,NOTIFY=&SYSUID
//*
//STP01 EXEC PGM=SORT
//* Assuming STP01 ends with RC0.

//STP02 EXEC PGM=MYCOBB,COND=(4,EQ,STP01)
//* In STP02, condition evaluates to FALSE, step is executed
//* and assume the step abends.

//STP03 EXEC PGM=IEBGENER,COND=((0,EQ,STP01),ONLY)
//* In STP03, though the STP02 abends, the condition
//* (0,EQ,STP01) is met. Hence STP03 is bypassed.
```

IF-THEN-ELSE Construct

Another approach to control the job processing is by using IF-THEN-ELSE constructs. This gives more flexibility and user-friendly way of conditional processing.

Syntax

Following is the basic syntax of a JCL IF-THEN-ELSE Construct:

```
//name IF condition THEN
list of statements //* action to be taken when condition is true
//name ELSE
list of statements //* action to be taken when condition is false
//name ENDIF
```

Following is the description of the used terms in the above IF-THEN-ELSE Construct:

- **name** : This is optional and a name can have 1 to 8 alphanumeric characters starting with alphabet, #,$ or @.
- **Condition** : A condition will have a format: **KEYWORD OPERATOR VALUE**, where **KEYWORDS** can be RC *ReturnCode*, ABENDCC *Systemorusercompletioncode*, ABEND, RUN *stepstartedexecution*. An **OPERATOR** can be logical operator AND (&, OR |) or relational operator < , <= , > , >= , <> .

Example

Following is a simple example showing the usage of IF-THEN-ELSE:

```
//CNDSAMP JOB CLASS=6,NOTIFY=&SYSUID
//*
//PRC1   PROC
//PST1    EXEC PGM=SORT
//PST2    EXEC PGM=IEBGENER
//       PEND
//STP01  EXEC PGM=SORT
//IF1    IF STP01.RC = 0 THEN
//STP02     EXEC PGM=MYCOBB1,PARM=123
//       ENDIF
//IF2    IF STP01.RUN THEN
//STP03a    EXEC PGM=IEBGENER
//STP03b    EXEC PGM=SORT
//       ENDIF
//IF3    IF STP03b.!ABEND THEN
//STP04     EXEC PGM=MYCOBB1,PARM=456
//       ELSE
//       ENDIF
//IF4    IF (STP01.RC = 0 & STP02.RC <= 4) THEN
//STP05     EXEC PROC=PRC1
//       ENDIF
//IF5    IF STP05.PRC1.PST1.ABEND THEN
//STP06     EXEC PGM=MYABD
//       ELSE
//STP07     EXEC PGM=SORT
//       ENDIF
```

Let's try to look into the above program to understand it in little more detail:

- The return code of STP01 is tested in IF1. If it is 0, then STP02 is executed. Else, the processing goes to the next IF statement *IF2*.
- In IF2, If STP01 has started execution, then STP03a and STP03b are executed.
- In IF3, If STP03b does not ABEND, then STP04 is executed. In ELSE, there are no statements. It is called a NULL ELSE statement.
- In IF4, if STP01.RC = 0 and STP02.RC <=4 are TRUE, then STP05 is executed.

- In IF5, if the proc-step PST1 in PROC PRC1 in jobstep STP05 ABEND, then STP06 is executed. Else STP07 is executed.
- If IF4 evaluates to false, then STP05 is not executed. In that case, IF5 are not tested and the steps STP06, STP07 are not executed.

The IF-THEN-ELSE will not be executed in the case of abnormal termination of the job such as user cancelling the job, job time expiry or a dataset is backward referenced to a step that is bypassed.

Setting Checkpoints

You can set checkpoint dataset inside your JCL program using **SYSCKEOV,** which is a DD statement.

A **CHKPT** is the parameter coded for multi-volume QSAM datasets in a DD statement. When a CHKPT is coded as CHKPT=EOV, a checkpoint is written to the dataset specified in the SYSCKEOV statement at the end of each volume of the input/output multi-volume dataset.

```
//CHKSAMP JOB CLASS=6,NOTIFY=&SYSUID
//*
//STP01     EXEC PGM=MYCOBB
//SYSCKEOV  DD DSNAME=SAMPLE.CHK,DISP=MOD
//IN1       DD DSN=SAMPLE.IN,DISP=SHR
//OUT1      DD DSN=SAMPLE.OUT,DISP=(,CATLG,CATLG)
//           CHKPT=EOV,LRECL=80,RECFM=FB
```

In the above example, a checkpoint is written in dataset SAMPLE.CHK at the end of each volume of the output dataset SAMPLE.OUT.

Restart Processing

You can restart processing ether using automated way using the **RD parameter** or manual using the **RESTART parameter**.

RD parameter is coded in the JOB or EXEC statement and it helps in automated JOB/STEP restart and can hold one of the four values: R, RNC, NR or NC.

- **RD=R** allows automated restarts and considers the checkpoint coded in the CHKPT parameter of the DD statement.
- **RD=RNC** allows automated restarts, but overrides *ignores* the CHKPT parameter.
- **RD=NR** specifies that the job/step cannot be automatically restarted. But when it is manually restarted using the RESTART parameter, CHKPT parameter *ifany* will be considered.
- **RD=NC** disallows automated restart and checkpoint processing.

If there is a requirement to do automated restart for specific abend codes only, then it can be specified in the **SCHEDxx** member of the IBM system parmlib library.

RESTART parameter is coded in the JOB or EXEC statement and it helps in manual restart of the JOB/STEP after the job failure. RESTART can be accompanied with a checkid, which is the checkpoint written in the dataset coded in the SYSCKEOV DD statement. When a checkid is coded, the SYSCHK DD statement should be coded to reference the checkpoint dataset after the JOBLIB statement *ifany*, else after the JOB statement.

```
//CHKSAMP JOB CLASS=6,NOTIFY=&SYSUID,RESTART=(STP01,chk5)
//*
//SYSCHK    DD DSN=SAMPLE.CHK,DISP=OLD
//STP01     EXEC PGM=MYCOBB
//*SYSCKEOV DD DSNAME=SAMPLE.CHK,DISP=MOD
//IN1       DD DSN=SAMPLE.IN,DISP=SHR
//OUT1      DD DSN=SAMPLE.OUT,DISP=(,CATLG,CATLG)
//           CHKPT=EOV,LRECL=80,RECFM=FB
```

In the above example, chk5 is the checkid, i.e., STP01 is restarted at checkpoint5. Please note that

a SYSCHK statement is added and SYSCKEOV statement is commented out in the previous program explained in Setting Checkpoint section.