

(Extract)



# JCL

**MVS - OS/390 - z/OS**

Michel Castelein - Arcis Services

14 May 2009

[arcis@advalvas.be](mailto:arcis@advalvas.be)

<http://www.arcis-services.net>

# (Extract)

## JOB COND parameter

```
//jobname JOB CLASS=n,MSGCLASS=n,
//          COND={rc-test | (rc-test[,rc-test]...)}
```

Every time a job step's execution is completed, the system performs the **JOB COND** parameter against the return code (**RC**) produced by that step.

On the JCL **COND** parameter you can specify up to eight *return code tests*.

Every return code test has the following syntax: **(code,operator)**

- **code** is a decimal number from 0 through 4095.
- **operator** is one of the following two-character operators:  
**GT** (greater than, i.e. '>'), **GE** (greater than or equal to, i.e. '≥'), **EQ** (equal to, i.e. '='), **LT** (less than, i.e. '<'), **LE** (less than or equal to, i.e. '≤'), or **NE** (not equal to, i.e. '≠').
- Do not forget the parentheses!

e.g. `COND=(0,NE)`  
`COND=(7,EQ),(16,LE)`

- Mind the semantics of the return code test!

e.g. `(16,LE)` This test is not "RC ≤ 16" ? !  
 It is "16 ≤ RC" ? !



When multiple return code tests are specified, they form an **OR expression**.

When the expression is true (i.e. if at least one of the return code tests is satisfied), the system bypasses ("flushes") all remaining job steps and terminates the job, i.e. the system uses a negative logic.



e.g. `//jobname JOB COND=(10,EQ)`

`//STEP1 EXEC PGM=program`

⋮

`10 = RC ?`

The return code (RC) from this step is 9.

No! Therefore, the job is not terminated.

`//STEP2 EXEC PGM=program`

⋮

`10 = RC ?`

The return code (RC) from this step is 10.

Yes! Therefore, the job is terminated.

e.g. `//jobname JOB COND=((15,GE),(20,LT))`

`//STEP1 EXEC PGM=program`

⋮

`(15 ≥ RC) OR (20 < RC)?`

The return code (RC) from this step is 22.

Yes! Therefore, the job is terminated.

# (Extract)

## EXEC COND parameter

```
//[stepname] EXEC {PGM=pgm|[PROC=]proc},
//  COND[.procstepname]=([rc-test][,rc-test)]...,[ONLY|EVEN])
```

Recall:

- If the EXEC statement calls a (cataloged or in-stream) procedure, the EXEC statement's keyword parameters affect all steps of the procedure.
- If a keyword parameter is to affect only one EXEC statement in a multi-step procedure, code `.procstepname` immediately following the keyword.

On the EXEC **COND** parameter you can specify up to eight test.

Those tests can be:

1. zero, one, or multiple *return code tests* with the format<sup>1</sup> `(code,operator)` to test the return codes from all previous steps that have completed processing.
2. and/or zero, one, or multiple *return code tests* with the format<sup>1</sup> `(code,operator,stepname[.procstepname])` to test the return code from a specific earlier job step.
  - `stepname` identifies the job step.
  - `stepname.procstepname` identifies a step in a (cataloged or in-stream) procedure called by an earlier job step. The step identified by `procstepname` must contain a PGM keyword, rather than invoke a procedure.
3. and/or one *ABEND test*, i.e. **ONLY** OR **EVEN**.

The system performs each “*non-specific*” *return code test* `(code,operator)` against the return code (**RC**) from every previous step that executed.

If the RC issued by any of those previous steps causes the test condition to be satisfied, the system evaluates the test as true.

A “*specific*” *return code test* `(code,operator,stepname[.procstepname])` refers to one of the previous steps that executed.

If the return code (**RC**) of that particular step causes the test condition to be satisfied, the system evaluates the “specific” return code test as true.



If a return code test specifies a previous step that was bypassed (“flushed”) or ended abnormally, the system evaluates the test as false.

When multiple (“non-specific” and/or “specific”) return code tests are specified, they form an **OR expression**.

<sup>1</sup> Cf. the JOB **COND** parameter for the syntax and semantics.

# (Extract)

When the expression is true, i.e. if at least one of the return code tests is satisfied, the system bypasses (“flushes”) the job step, i.e. the system uses a negative logic.



Notice:

- Bypassing (“flushing”) a step because of a return code test is not the same as abnormally terminating a step.
- The system always evaluates a **COND** parameter in the first EXEC statement in a job as false.

e.g. `//STEP4 EXEC PGM=PROG4,COND=(20,GE)`

If  $(20 \geq RC_{STEP1})$  **OR**  $(20 \geq RC_{STEP2})$  **OR**  $(20 \geq RC_{STEP3})$   
then bypass STEP4.

e.g. `//STEP8 EXEC PGM=PROG8,COND=((8,LT,STEP1),(7,GT,STEP2))`

If  $(8 < RC_{STEP1})$  **OR**  $(7 > RC_{STEP2})$  then bypass STEP8.

e.g. `//STEP9 EXEC PGM=PROG9,COND=(7,EQ,STEP3.PROCSTP2)`

STEP3 called an in-stream or cataloged procedure.

PROCSTP2 is a step in that procedure.

If  $(7 = RC_{STEP3.PROCSTP2})$  then bypass STEP9.

e.g. `//STEP11 EXEC PROC=PROC2,  
// COND.PROCSTP1=(4,LE),  
// COND.PROCSTP3=(0,EQ,STEP9)`

If  $(4 \leq RC_{\text{of-any-previous-step}})$  then bypass PROCSTP1 of PROC2.

If  $(0 = RC_{STEP9})$  then bypass PROCSTP3 of PROC2.

e.g. `//STEP13 EXEC PROC=PROC5,  
// COND.PROCSTP2=((12,LE),(8,EQ),(3,NE,STEP12.PROCSTP6))`

If  $(12 \leq RC_{\text{of-any-previous-step}})$  **OR**  $(8 = RC_{\text{of-any-previous-step}})$  **OR**

$(3 \neq RC_{STEP12.PROCSTP6})$  then bypass PROCSTP2 of PROC5.

Notice that, according to Augustus De Morgan<sup>1</sup>, the following three statements are equivalent:

- If  $(12 \leq RC_{\text{of-any-previous-step}})$  **OR**  $(8 = RC_{\text{of-any-previous-step}})$  **OR**  $(3 \neq RC_{STEP12.PROCSTP6})$  then bypass PROCSTP2 of PROC5.
- If  $(12 \text{ NOT } \leq RC_{\text{of-any-previous-step}})$  **AND**  $(8 \text{ NOT } = RC_{\text{of-any-previous-step}})$  **AND**  $(3 \text{ NOT } \neq RC_{STEP12.PROCSTP6})$  then execute PROCSTP2 of PROC5.
- If  $(12 > RC_{\text{of-any-previous-step}})$  **AND**  $(8 \neq RC_{\text{of-any-previous-step}})$  **AND**  $(3 = RC_{STEP12.PROCSTP6})$  then execute PROCSTP2 of PROC5.

<sup>1</sup> English mathematician (1806-1871), known for his two laws:  $A \cap B = \overline{\overline{A} \cup \overline{B}}$  and  $A \cup B = \overline{\overline{A} \cap \overline{B}}$

$\cap$ ,  $\cup$ , and the overbar denote respectively intersection (logical AND), union (logical OR), and negation.

# (Extract)

The following example shows how to write the EXEC **COND** parameter:

- Let's consider a job consisting of three job steps:
  - The first step calls a compiler to process a source program.
  - The second step invokes the linkage editor (or binder) to convert the compiler's output (i.e. an object module) into an executable program (i.e. a load module or program object).
  - The third step executes the new program.  
The third step should be executed only if the first step's return code is less than or equal to 8, and the second step's return code is zero.
- Take the following approach:
  1. If  $(RC_{STEP1} \leq 8)$  AND  $(RC_{STEP2} = 0)$  then execute STEP3
  2. Apply the first law of De Morgan:  
If  $(RC_{STEP1}$  NOT  $\leq 8)$  OR  $(RC_{STEP2}$  NOT  $= 0)$  then **bypass** STEP3
  3. Rewrite the comparison operators:  
If  $(RC_{STEP1} > 8)$  OR  $(RC_{STEP2} \neq 0)$  then **bypass** STEP3
  4. Invert the sequence of the terms:  
If  $(8 < RC_{STEP1})$  OR  $(0 \neq RC_{STEP2})$  then **bypass** STEP3
  5. Write the EXEC **COND** parameter:  
`//STEP3 EXEC PGM=pgm,COND=((8,LT,STEP1),(0,NE,STEP2))`

To perform an *ABEND test*, code **ONLY** or **EVEN**:

```
//[stepname] EXEC {PGM=pgm|[PROC=]proc},  
// COND[.procstepname]=([(rc-test)[,(rc-test)]...],[ONLY|EVEN])1
```



When performing an *ABEND test*, the system uses a positive logic:

- **ONLY** specifies that this job step is to be executed only if a preceding job step resulted in an ABEND, i.e. the step is a recovery step.
- **EVEN** specifies that this job step is to be executed even if a preceding job step resulted in an ABEND, i.e. an unconditional execution is required.

Recall that every job step consists of a sequence of three actions.

For the system to act on the **EVEN** or **ONLY** option, i.e. to trap an ABEND condition, the step must abnormally terminate while the program has control.

If a step abnormally terminates during scheduling, due to failures such as JCL errors, the system bypasses the remaining steps, no matter what the EXEC **COND** parameter requests.



If the EXEC **COND** parameter specifies both one or more return code tests and an ABEND test:

- If at least one of the return code tests is satisfied, the step is bypassed, no matter what the ABEND test specifies.

<sup>1</sup> You can omit the outer parentheses if you code only one return code test or only **EVEN** or **ONLY**.

# (Extract)

- If none of the return code tests is satisfied, the ABEND test determines whether or not the step is to be executed.

i.e.

EXEC COND parameter?	Any RC-test satisfied?	EVEN?	ONLY?	Any preceding ABEND?	Result
NO	n/a	n/a	n/a	NO	Execute this step
				YES	Bypass this step
YES	YES <sup>1</sup>	-	-	-	Bypass this step
	NO <sup>2</sup>	NO	NO	NO	Execute this step
				YES	Bypass this step
		YES	NO	NO	Execute this step
				YES	Execute this step
		NO	YES	NO	Bypass this step
				YES	Execute this step

e.g. //STEP10 EXEC PROC=PROC5, COND=ONLY

STEP10 (i.e. PROC5) is to be executed only if a previously executed program had an ABEND.

e.g. //STEP11 EXEC PROC=PROC6, COND.PROCSTP2=EVEN

PROCSTP2 of PROC6 is to be executed even if a previously executed program had an ABEND.

e.g. //STEP12 EXEC PROC7, COND=((10,LT),EVEN)

If  $(10 < RC_{\text{of-any-previous-step}})$  then bypass STEP12 (i.e. PROC7).

If the return code test is not satisfied, STEP12 must be executed even if there has been an ABEND.

e.g. //STEP13 EXEC PROC8,

// COND.PROCSTP3=((10,LT),(4,NE),EVEN),

// COND.PROCSTP5=((5,EQ,STEP12.PROCSTP2),ONLY)

If  $(10 < RC_{\text{of-any-previous-step}})$  OR  $(4 \neq RC_{\text{of-any-previous-step}})$  then bypass PROCSTP3 of PROC8.

Otherwise, execute PROCST3 of PROC8 even if there has been an ABEND.

If  $(5 = RC_{\text{STEP12.PROCSTP2}})$  then bypass PROCSTP5 of PROC8.

Otherwise, execute PROCSTP5 of PROC8 only if there has been an ABEND.

<sup>1</sup> One or more RC-tests are coded and at least one of them is satisfied.

<sup>2</sup> RC-tests are omitted, or none of the specified RC-tests is satisfied.

# (Extract)

## Conditional JCL

The JOB **COND** parameter has a higher priority than the EXEC **COND** parameter:

- If at least one return code test on the JOB statement is satisfied, the job terminates. Any EXEC **COND** parameter is ignored.
- If none of the return code tests on the JOB statement is satisfied, the tests on the EXEC statement are performed.

You cannot restrict the scope of a return code test on the JOB **COND** parameter to the return code of a specific step.

The JOB **COND** parameter cannot be used to trap an ABEND condition.

Beginning with MVS/SP4.1, you can use the **IF-THEN/ELSE/ENDIF** statement construct instead of coding EXEC **COND** parameters.

The **IF-THEN/ELSE/ENDIF** statement construct is easier to use and has more functions than the EXEC **COND** parameter.

Use the **IF-THEN/ELSE/ENDIF** statement construct to conditionally execute job steps within a job:

```

//[name] IF [( )relational-expression( )] THEN [comments]
...      (action when relational-expression is true)
[ // [name] ELSE                               [comments]
...      (action when relational-expression is false)
// [name] ENDIF                               [comments] ]

```

- **name** is optional. If specified, it must be unique within a job, and has the same requirements as a DD name.
- **relational-expression** consists of one or more tests.  
The result of evaluating a relational-expression is either true or false.
- **action** must/may consist of:
  - at least one EXEC statement;
  - nested IF-THEN/ELSE/ENDIF constructs ( up to 15 levels);
  - DD statements;
  - OUTPUT JCL statements;
  - CNTL and ENDCNTL statements<sup>1</sup>.

---

<sup>1</sup> The **CNTL** and **ENDCNTL** statements are used to mark the beginning and end of an instream enumeration of program control statements for a subsystem such as the Print Services Facility (**PSF**) .

On a DD statement, use the **CNTL** parameter to reference a **CNTL/ENDCNTL** statement construct that appears earlier in the job.

See the “**JCL Reference**” in the OS/390 or z/OS **MVS** bookshelf.

# (Extract)

## IF-THEN/ELSE/ENDIF statement construct

A relational-expression consists of *operators* and *relational-expression keywords*.

You can also use parentheses within a relational-expression.

The operators that you can use in the relational-expression and their processing priority are:

(1)	<b>NOT</b> or $\neg$	Logical NOT
	<b>GT</b> or $>$	Greater than
	<b>LT</b> or $<$	Less than
	<b>NG</b> or $\neg>$	Not greater than
(2)	<b>NL</b> or $\neg<$	Not less than
	<b>EQ</b> or $=$	Equal to
	<b>NE</b> or $\neg=$	Not equal to
	<b>GE</b> or $\geq$	Greater than or equal to
(3)	<b>LE</b> or $\leq$	Less than or equal to
	<b>AND</b> or $\&$	Logical AND
	<b>OR</b> or $ $	Logical OR

- You can use  $\neg$  (logical not),  $\backslash$  (backslash), and  $\wedge$  (hat) interchangeably in operators according to availability and personal preference.
- The system expects EBCDIC 4F<sub>16</sub> for the vertical bar (|). Users in countries using code pages other than EBCDIC U.S. Code Page (Cp037) may have to enter a different character. With Cp1148 for instance, you must enter ! instead of the vertical bar.



**RC** refers to the highest return code<sup>1</sup> that occurred during job processing prior to the time of evaluation.

e.g. 

```
// IF RC GE 12 THEN
// IF RC >= 12 THEN
// IF RC>=12 THEN
```

**stepname[.procstepname].RC** refers to the return code<sup>1</sup> of a specific step among the previous steps that executed<sup>2</sup>.

e.g. 

```
// IF RC >= 12 OR STEP1.RC ^= 0 THEN
// IF RC >= 12 ! NOT (STEP1.RC = 0) THEN
// IF RC >= 12 ! NOT(STEP1.RC = 0) THEN
```

<sup>1</sup> The return code must be within the range of 0 – 4095.

<sup>2</sup> If the step that you are evaluating was bypassed (i.e. “flushed”) or ended abnormally, the result of the return code test evaluation is false.

# (Extract)

Specify **ABEND** to test for an ABEND condition that occurred on any of the previous steps that executed.

```
e.g. // IF ABEND          THEN
      // IF ABEND = TRUE THEN
      // IF ABEND=TRUE   THEN
```

```
e.g. // IF NOT ABEND     THEN
      // IF ^ABEND      THEN
      // IF ABEND = FALSE THEN
      // IF ABEND=FALSE  THEN
```



Use **stepname[.procstepname].ABEND** to test for an ABEND condition that occurred on a specific step among the previous steps that executed<sup>1</sup>.

```
e.g. // IF (STEP01.ABEND OR STEP03.ABEND = TRUE) AND STEP05.ABEND
      //      = FALSE THEN /* MULTI-LINE IF-THEN STATEMENT */
```

To continue a relational-expression, break it where a blank is valid on the current line, code `//` in columns 1 and 2 of the following line, and continue the expression beginning in column 4 through 16 of that line.

Do not put comments on the line you are continuing; comments can be coded after you have completed the statement.



Specify **[stepname[.procstepname].]ABENDCC** and **{sxxx|udddd}** to test the ABEND completion code: code **s** with 3 hexadecimal digits for a system completion code, or **u** with 4 decimal digits for a user completion code<sup>1</sup>.

```
e.g. // IF STEP05.ABENDCC >= U0100 THEN
```



If **stepname[.procstepname]** is omitted, the **ABENDCC** test refers to the most recent ABEND code that occurred during the execution of the job prior to the time of evaluation.

For the system to trap an ABEND condition, the step must abnormally terminate while the program has control.



**stepname[.procstepname].RUN** indicates that the system is to test whether a specific step started execution or was bypassed (“flushed”).

```
e.g. // IF NOT STEP02.RUN          THEN
      // IF STEP02.RUN=FALSE      THEN
      // IF NOT (STEP02.RUN = TRUE) THEN
```

You must always specify a step name when using the **RUN** keyword.

The **IF-THEN/ELSE/ENDIF** statement construct can appear anywhere in the job after the first EXEC statement.

Do not place a JOB, JOBCAT DD, JOBLIB DD, or JCLLIB statement in a **THEN** or **ELSE** clause.

<sup>1</sup> If the step that you are evaluating was bypassed, the result of the ABEND test evaluation is false.