Understanding and Coding JCL COND= Parameters

-Ian! D. Allen - idallen@idallen.ca - www.idallen.com

JCL COND codes are used on EXEC statements to skip steps if any of the previous steps have certain return codes. (In other words, you execute the current step only if *every* previous step does *not* have the specified return code.)

The logic in COND is reversed from the way you'd expect it to work. If the given condition line evaluates as TRUE for *any* of the previous steps, the current step is *skipped*, not executed. Many people find this confusing. (I do.)

If you don't specify a step to test in the COND= parameter, then *all* previous steps are tested. Any TRUE value causes the current step to be skipped (not executed). The current step will execute only if the COND test fails (is FALSE) for *every* previous step.

Here are some examples, with some steps you can follow to turn the English of the job specification into the JCL COND parameter:

A) If the return code (RC) of every previous step is equal to zero then execute this step.

```
Step 1: execute IF RC = 0   (rephrase and shorten the above statement)
Step 2: execute IF 0 = RC   (put the number on the left and the RC right)
Step 3: skip IF 0 != RC     (complement both sides to make into "skip")
Step 4: COND=(0,NE)         (write it as JCL)
```

Note that Step 2 simply reverses the order of the operands to get the number on the left and the RC on the right; it does not change the logical relationship between them. RC = 0 is the same as 0 = RC.

B) If the return code (RC) of every previous step is less than four then execute this step.

```
Step 1: execute IF RC < 4   (rephrase and shorten)
Step 2: execute IF 4 > RC   (put number on left)
Step 3: skip IF 4 <= RC     (complement both sides to make "skip")
Step 4: COND=(4,LE)
```

Note that Step 2 simply reverses the order of the operands; it does not change the logical relationship between them. RC < 4 is the same as 4 > RC. (Saying 3 < 4 is the same as saying 4 > 3.)

C) Execute this step unless the return code of any previous step is bigger than 3.

This is tricky English. The word "unless" means the same as "if not". We could code this using "unless", and work it through:

```
Step 1: execute UNLESS RC > 3   (rephrase and shorten)
Step 2: execute IF NOT RC > 3   (translate UNLESS to IF NOT)
Step 3: execute IF RC <= 3      (logic: "NOT >" means "<=")
Step 4: execute IF 3 >= RC      (number goes on the left)
Step 5: skip IF 3 < RC          (complement to make into a skip)
Step 6: COND=(3,LT)             (code it)
```

```
    Note that Step 4 simply reverses the order of the operands.
    RC <= 3 is the same as 3 >= RC.

    We could also observe that "execute unless" really means "skip".  So the
    English means: "skip this step if the return code is bigger than 3.

    Step 1: skip IF RC > 3       (rephrase and shorten)
    Step 2: skip IF 3 < RC       (number goes on the left)
    Step 3: COND=(3,LT)          (code it)

    Note that Step 2 simply reverses the order of the operands.
    RC > 3 is the same as 3 < RC.

D)  Don't execute this step unless every previous step returns a
    return code (RC) bigger than 5.

    This is tricky English.  "Don't execute" means "skip"; "unless"
    means "if not".  Work it through:

    Step 1: skip IF NOT RC > 5    (rephrase and shorten)
    Step 2: skip IF RC <= 5       (logic: "NOT >" means "<=")
    Step 3: skip IF 5 >= RC       (number goes on the left)
    Step 4: COND=(5,GE)

    Note that Step 3 simply reverses the order of the operands.
    RC <= 5 is the same as 5 >= RC.

    We could also reword to remove "Don't" and "unless", which cancel
    each other out: Execute this step if every previous step has a return
    code bigger than 5.

    Step 1: execute IF RC > 5
    Step 2: execute IF 5 < RC
    Step 3: skip IF 5 >= RC
    Step 4: COND=(5,GE)

    Note that Step 2 simply reverses the order of the operands.
    RC > 5 is the same as 5 < RC.

    Why would you have a step that executes when other steps fail?
    Perhaps to clean up the mess left behind by the failure!

    You can also derive the simpler English from the above steps:

        skip IF 5 >= RC
        skip IF RC <= 5
        execute IF RC > 5

    This translates to: "Execute this step if the return code of every
    previous step is greater than five."

E)  Unless every previous step returns a code of 2 or bigger, skip this step.

    This is tricky English.  We could code this using "skip" and "unless",
    and work it through (recoding "unless" as "if not"):  Skip this step
    unless ("if not") a previous step returns a code of 2 or bigger.

    Method A:

    Step 1: skip IF NOT RC >= 2   (reword and shorten)
    Step 2: skip IF RC < 2        (logic: "NOT >=" means "<")
    Step 3: skip IF 2 > RC        (number goes on the left)
    Step 4: COND=(2,GT)

    What does this mean?  In better English:

        skip IF RC < 2
        execute IF RC >= 2        (complement "skip" into "execute")

    It means: "Execute this step if the return code of every previous
    step is greater than or equal to two."

    Method B:

    We could reword to remove "unless" and "skip", which cancel each
    other out: Execute this step if every previous step has a return
    code greater than or equal to 2.

    Step 1: execute IF RC >= 2    (reword and shorten)
    Step 2: execute IF 2 <= RC    (number goes on the left)
    Step 3: skip IF 2 > RC        (complement to make into "skip")
    Step 4: COND=(2,GT)

    Note that Step 2 simply reverses the order of the operands.
    RC >= 2 is the same as 2 <= RC.

F)  If the return code (RC) of every previous step is equal to zero then
    execute this step.

    Step 1: execute IF RC = 0
    Step 2: execute IF 0 = RC
    Step 3: skip IF 0 != RC
    Step 4: COND=(0,NE)

J)  Don't execute this step if any previous step returns a code bigger
    than 5.

    Step 1: skip IF RC > 5
    Step 2: skip IF 5 < RC
    Step 3: COND=(5,LT)

    What does this mean?  In better English:

        skip if RC > 5
        execute IF RC <= 5

    It means: "Execute this step if every previous step return code is
    less than or equal to five.

K)  Execute this step unless the return code of any previous step is less
    than four.

    Step 1: execute IF NOT RC < 4    (UNLESS is the same as "IF NOT")
    Step 2: execute IF RC >= 4       (logic: recode "NOT <" as ">=")
    Step 3: execute IF 4 <= RC       (number goes on the left)
    Step 4: skip IF 4 > RC           (complement to make into "skip")
    Step 5: COND=(4,GT)

    In better English:
        execute IF RC >= 4

    "Execute this step if the return code of every previous step is
    greater than or equal to 4."

L)  Unless the return code of every previous step is zero, skip this step.

    Step 1: skip IF NOT RC = 0      (UNLESS is the same as "IF NOT")
    Step 2: skip IF RC != 0         (logic: recode "NOT =" as "!=")
    Step 3: skip IF 0 != RC         (number goes on the left)
    Step 4: COND=(0,NE)

    In better English:

        skip IF NOT RC = 0
        execute IF RC = 0

    The "skip unless" double-negative translates to "execute if":
    "Execute this step if the return code of every previous step is
    equal to zero."

M)  Execute this step unless the return code of any previous step is
    less than 10.

    Step 1: execute IF NOT RC < 10   (UNLESS is the same as "IF NOT")
    Step 2: execute IF RC >= 10      (logic: recode "NOT <" as ">=")
    Step 3: execute IF 10 <= RC      (swap: number goes on the left)
    Step 4: skip IF 10 > RC          (complement to make into "skip")
    Step 5: COND=(10,GT)

    "Execute this step if the return code of all previous steps is greater
    than or equal to 10."

N)  Execute this step as long as no previous step has set a condition
    code of 4 or more.

    "Execute as long as no..." is the same as "Execute IF NOT...".

    Step 1: execute IF NOT RC >= 4   ("as long as no" is the same as "IF NOT")
    Step 2: execute IF RC < 4        (logic: recode "NOT >=" as "<")
    Step 3: execute IF 4 > RC        (swap: number goes on the left)
    Step 4: skip IF 4 <= RC          (complement to make into "skip")
    Step 5: COND=(4,LE)

    "Execute as long as no" is the same as "Execute IF NOT" is the same
    as "Skip...".

    Step 1: skip IF RC >=  4
    Step 2: skip IF  4 <= RC
    Step 3: COND=(4,LE)

-----------------------------
Notes on "any" versus "every"
-----------------------------

The COND parameter on the EXEC statement will skip this step if the
condition is TRUE for *any* of the previous steps.  In other words,
to execute the current step, the condition must be FALSE for *every*
one of (all) the previous steps.  For example, these are equivalent
for "skip if zero is not equal to any previous return code":
    COND=(0,NE)
    skip    IF          0 != (any) RC   (skip if true for *any* previous step)
    skip    IF   (any) RC != 0          (swap the clauses)
    execute IF (every) RC  = 0          (logical complement of both sides)
```

In other words: "Execute this step if every RC in all previous steps is zero."
The logical complement turns "skip if any" into "execute if every".

Limitations

You can't use COND to handle conditions that depend on skipping a step
if *every* (all) return codes have some value; nor, can you use COND to
handle conditions that depend on executing a step if *any* return codes
have some value. You cannot code "skip if every" or "execute if any"
using a simple COND. You can only code "skip if any" and "execute
if every".

For example, you can't translate this into a single COND:

```
   "Skip this step if *every* previous return code is zero." (IMPOSSIBLE)
```

You can only code this (which has a different meaning):

```
   "Skip this step if *any* previous return code is zero."  (WRONG MEANING)
```

You can only code "any" for a "skip" and, logically, "every" for an "execute".

In the Real World, you can combine multiple COND parameters to check
the values on all your previous steps. In this course, you will only
be asked to code "execute if every" and "skip if any" COND parameters.

```
--
| Ian! D. Allen  -  idallen@idallen.ca  -  Ottawa, Ontario, Canada
| Home Page: http://idallen.com/   Contact Improv: http://contactimprov.ca/
| College professor (Free/Libre GNU+Linux) at: http://teaching.idallen.com/
| Defend digital freedom:  http://eff.org/  and have fun:  http://fools.ca/
```