Windows Batch Scripting: Return Codes

Today we'll cover return codes as the right way to communicate the outcome of your script's execution to the world. Sadly, even skilled Windows programmers overlook the importance of return codes.

Return Code Conventions

By convention, command line execution should return zero when execution succeeds and non-zero when execution fails. Warning messages typically don't effect the return code. What matters is did the script work or not?

Checking Return Codes In Your Script Commands

The environmental variable `%ERRORLEVEL%` contains the return code of the last executed program or script. A very helpful feature is the built-in DOS commands like `ECHO`, `IF`, and `SET` will preserve the existing value of `%ERRORLEVEL%`.

The conventional technique to check for a non-zero return code using the `NEQ` (Not-Equal-To) operator of the `IF` command:

```
IF %ERRORLEVEL% NEQ 0 (
  REM do something here to address the error
)
```

Another common technique is:

```
IF ERRORLEVEL 1 (
  REM do something here to address the error
)
```

The `ERRORLEVEL 1` statement is true when the return code is any number equal to or greater than 1. However, I don't use this technique because programs can return negative numbers as well as positive numbers. Most programs rarely document every possible return code, so I'd rather explicity check for non-zero with the `NEQ 0` style than assuming return codes will be 1 or greater on error.

You may also want to check for specific error codes. For example, you can test that an executable program or script is in your PATH by simply calling the program and checking for return code 9009.

```
SomeFile.exe
IF %ERRORLEVEL% EQU 9009 (
  ECHO error - SomeFile.exe not found in your PATH
)
```

It's hard to know this stuff upfront – I generally just use trial and error to figure out the best way to check the return code of the program or script I'm calling. Remember, this is duct tape programming. It isn't always pretty, but, it gets the job done.

Conditional Execution Using the Return Code

There's a super cool shorthand you can use to execute a second command based on the success or failure of a command. The first program/script must conform to the convention of returning 0 on success and non-0 on failure for this to work.

To execute a follow-on command after sucess, we use the `&&` operator:

```
SomeCommand.exe && ECHO SomeCommand.exe succeeded!
```

To execute a follow-on command after failure, we use the `||` operator:

```
SomeCommand.exe || ECHO SomeCommand.exe failed with return code %ERRORLEVEL%
```

I use this technique heavily to halt a script when any error is encountered. By default, the command processor will continue executing when an error is raised. You have to code for halting on error.

A very simple way to halt on error is to use the `EXIT` command with the `/B` switch (to exit the current batch script context, and not the command prompt process). We also pass a specific non-zero return code from the failed command to inform the caller of our script about the failure.

```
SomeCommand.exe || EXIT /B 1
```

A simliar technique uses the implicit GOTO label called `:EOF` (End-Of-File). Jumping to EOF in this way will exit your current script with the return code of 1.

```
SomeCommand.exe || GOTO :EOF
```

Tips and Tricks for Return Codes

I recommend sticking to zero for success and return codes that are positive values for DOS batch files. The positive values are a good idea because other callers may use the `IF ERRORLEVEL 1` syntax to check your script.

I also recommend documenting your possible return codes with easy to read `SET` statements at the top of your script file, like this:

```
SET /A ERROR_HELP_SCREEN=1
SET /A ERROR_FILE_NOT_FOUND=2
```

Note that I break my own convention here and use uppercase variable names – I do this to denote that the variable is constant and should not be modified elsewhere. Too bad DOS doesn't support constant values like Unix/Linux shells.

Some Final Polish

One small piece of polish I like is using return codes that are a power of 2.

```
SET /A ERROR_HELP_SCREEN=1
SET /A ERROR_FILE_NOT_FOUND=2
SET /A ERROR_FILE_READ_ONLY=4
SET /A ERROR_UNKNOWN=8
```

This gives me the flexibility to bitwise OR multiple error numbers together if I want to record numerous problems in one error code. This is rare for scripts intended for interactive use, but, it can be super helpful when writing scripts you support but you don't have access to the target systems.

```
@ECHO OFF
SETLOCAL ENABLEEXTENSIONS

SET /A errno=0
SET /A ERROR_HELP_SCREEN=1
SET /A ERROR_SOMECOMMAND_NOT_FOUND=2
SET /A ERROR_OTHERCOMMAND_FAILED=4
```

```
SomeCommand.exe
IF %ERRORLEVEL% NEQ 0 SET /A errno^|=%ERROR_SOMECOMMAND_NOT_FOUND%

OtherCommand.exe
IF %ERRORLEVEL% NEQ 0 (
    SET /A errno^|=%ERROR_OTHERCOMMAND_FAILED%
)

EXIT /B %errno%
```

If both SomeCommand.exe and OtherCommand.exe fail, the return code will be the bitwise combination of 0x1 and 0x2, or decimal 3. This return code tells me that both errors were raised. Even better, I can repeatedly call the bitwise OR with the same error code and still interpret which errors were raised.