Wrapping some other scripting language inside a batch file

devblogs.microsoft.com/oldnewthing/20170804-00

August 4, 2017

Raymond Chen

Nobody actually enjoys batch programming, but sometimes you can get away with writing in a language you like better while retaining the `.cmd` extension. Still, that leaves you having to get the extension for that language registered on your target machines, which can be tricky for xcopy-style deployment scenarios. The solution then is to use a polyglot header that is valid both as a batch file and in your target language. The header re-invokes the target language interpreter with the batch file itself as input.

Note: That this trick isn't necessary if you can associate the file extension with the scripting engine. So you don't need to do this polyglot nonsense with, say, PowerShell scripts, because the `.ps1` extension is already associated with `powershell.exe` (where available).

The general shape of a polyglot header is

```
@rem prefix stuff
@‹interpreter›.exe ‹interpreter options› "%~f0" %*
@goto :eof
‹suffix stuff›
‹the script itself›
‹trail stuff›
```

Prefixing each line with an at-sign prevents it from being echoed. The first line is a comment, which lets you stick arbitrary goop in front, in order to swallow up the `@rem` and make the rest of the header invisible to the interpreter.

The `"%~f0" %*` sequence looks like line noise, but it's actually a batch file idiom for "A quoted, fully-qualified path to the batch file, followed by the original arguments." The `%~f0` part uses the tilde operator to build up a full path to the `%0` (which is the batch file itself). And `%*` is a batch variable that expands to the arguments passed to the batch file.

Anything after the `@goto :eof` is ignored by the batch interpreter, so you can add language-specific suffix stuff to finish up the "start ignoring this" goop you set up on the first line.

Finally, in rare cases, you might need to add trail stuff at the end of the script to balance out anything you set up in the header, like closing an open set of braces. This is rare because you usually close them up in the ⟨suffix stuff⟩ part.

Okay, now that we see the general shape of a polyglot header, let's look at some examples.

Perl

```
@rem --*-Perl-*--
@perl.exe -x "%~f0" %*
goto :eof
#!perl
⟨perl script⟩
```

This isn't a proper polyglot because we're running perl in a special mode which is not the default (`-x`). But hey, we're trying to get things done, not solve some theoretical puzzle, so running perl in a special mode is just fine if it gets the job done.

Note that if you want other special command line options to be passed to perl, you can sneak them in with the `-x` . For example, you might ask for `-Sx` to get poor-man's command line switch auto-parsing.

The leading comment `--*-Perl-*--` is not used by either perl or the command processor. It's there by tradition, so that when emacs users load the script into the editor, it will be detected as a perl script, and perl-specific editing commands will be enabled.

JavaScript

```
@if (1 == 0) @end /*
@cscript.exe /E:jscript /nologo "%~f0" %*
goto :eof
*/
⟨JScript script⟩
```

Instead of using `@rem` , the JScript polyglot header uses an `@if` conditional that is never true. This was chosen so that the opening syntax of the file matches that of JScript conditional compilation, and the entire header gets gobbled up as a false conditional followed by a big comment. Note that JScript conditional compilation is a Microsoft extension, but since `cscript` runs the Microsoft JScript engine when you specify `/e:jscript` , it's okay to use it anyway.

Bonus chatter: Sometimes I miss the `EXTPROC` directive from OS/2's command interpreter, then I realize that it really only solves half of the problem (getting the command interpreter to hand control to another scripting engine), and doesn't solve the other half (getting the scripting engine to ignore the start of the batch file). The additional restriction that `EXTPROC` appear on the first line of the batch file makes it harder to work the first line into valid code in your target language.

Bonus chatter 2: JScript is probably the most convenient alternative scripting language because, while it may be the world's most misunderstood programming language, it's nevertheless immeasurably better than batch. And it has come preinstalled since Windows 2000, so your script will work on pretty much any Windows computer of modern interest. The downside is that the version of JScript used by `cscript` is ancient.

PowerShell is very nice, but it wasn't standard-issue until Windows 7. With the retirement of Windows Vista, we are finally in a situation where all supported versions of Windows come with PowerShell. It took eight years, but we made it. (Note that you can't run PowerShell scripts by default. You have to go in and change an administrative setting first.)

So maybe, if you're lucky, you may be able to declare the end of the era of suffering with batch files. I can more confidently say that the suffering of Batch File Week is now over, at least for now.

Bonus content: Here'a Web page which demonstrates various batch file string manipulation operations.

Raymond Chen

Follow