What are these strange =C: environment variables?

devblogs.microsoft.com/oldnewthing/20100506-00

May 6, 2010

Raymond Chen

You won't see them when you execute a `SET` command, but if you write a program that manually enumerates all the environment variables and prints them out, and if you launch it from a command prompt, then you'll see weird variables with names like =C: and whose values correspond to directories on that drive. What are these things?

These variables are part of the private bookkeeping of the command processor cmd.exe. That's why I added *if you launch it from a command prompt* to the steps above, because if you run the program from Explorer's *Run* dialog, you won't see them. If a cmd.exe is not in the chain of custody of your environment block, then you won't see the weird cmd.exe bookkeeping variables.

Okay, so the command processor sets these things, but what are they? They are a leftover from the command processor's attempt to mimic the old MS-DOS way that drives and directories were handled. Whereas in Win32, there is only one current directory, in MS-DOS, there was one current directory for each drive. Consider the following sequence of commands:

```
A> CD \SUBDIR
// current directory for drive A is A:\SUBDIR
A> B:
B> CD \TWO
// current directory for drive B is B:\TWO
B> A:
A> DIR
// shows a directory listing for A:\SUBDIR
```

During this sequence of commands, we start with A: as the current drive and set its current directory to A:\SUBDIR. Next, we set the current drive to B: and set B:\TWO as its current directory. Finally, we set the current drive back to A:, and when we ask for a listing, we get the contents of A:\SUBDIR because that is the current directory on the current drive.

Win32 does not have the concept of a separate current directory for each drive, but the command processor wanted to preserve the old MS-DOS behavior because people were accustomed to it (and batch files relied upon it). The solution was to store this "per-drive

current directory" in the environment, using a weird-o environment variable name so it wouldn't conflict with normal environment variables.

If you repeated the above commands in cmd.exe, the output is the same, but it is accomplished in a very different way.

```
A> CD \SUBDIR
// Environment variable =A: set to A:\SUBDIR
// Current Win32 directory set to A:\SUBDIR
A> B:
B> CD \TWO
// Environment variable =B: set to B:\TWO
// current Win32 directory set to B:\TWO
B> A:
// Current Win32 directory set to A:\SUBDIR
A> DIR
// shows a directory listing for A:\SUBDIR
```

When we switch back to drive A:, the command processor says, "Hey, what was the current directory on drive A: the last time I was there?" It looks into its environment and finds the =A: variable, which tells it, "Oh, it was A:\SUBDIR". And that's the Win32 directory that it sets as current.

But why put these internal variables in the environment? Can't they just be regular variables inside the cmd.exe process?

The variables are exported into the environment because you want these "fake per-drive current directories" to be inherited by child processes. For example, consider that you are sitting at your command prompt, you run emacs, then from emacs, you shell out to another command prompt. You would expect that the nested command prompt will have the same "per-drive current directories" that you set back in the outer command prompt.

```
C:\SUBDIR> D:
D:\> emacs
M-x shell
D:\> C:
C:\SUBDIR>
// the "current directory on drive C" was inherited as expected
```

What should you do about these variables, then?

Nothing. Just let them be and do their jobs. I'm just mentioning them here so you won't freak out when you see them.

Raymond Chen

Follow