Going for the facts: Who is displaying this message box?

devblogs.microsoft.com/oldnewthing/20140320-00

March 20, 2014

Raymond Chen

A customer wanted to know whether `CreateProcess` had a problem with Unicode.

A problem with it? Quite the contrary. `CreateProcess` *loves* Unicode! In fact, if you call the ANSI version, it converts the strings to Unicode and then finishes the work in Unicode.

Okay, here's the customer's problem.

> We have a custom application written in managed code. When we launch the process from unmanaged code via `CreateProcess`, we sometimes get a bogus error message:
>
> > WARNING! The specified path, file name, or both are too long. The fully qualified file name must be less than 260 characters, and the directory name must be less than 248 characters.
>
> The filename is well under the 260-character limit and the directory name is well under the 248-character limit. We have isolated the problem to be related to whether we put Unicode characters in the command line arguments. If the command line arguments are all ASCII, then no message appears.
>
> In case it matters, here's our code to launch the custom application.
>
> ```
> STARTUP_INFO si = { sizeof(si) };
> PROCESS_INFORMATION pi;
> if (CreateProcess(NULL, commandLine, 0, 0, FALSE,
>                   NORMAL_PRIORITY_CLASS, 0, 0,
>                   &si, &pi)  ...
>  // (in our case, the call succeeds)
> ```
>
> What do we have to do to get `CreateProcess` to accept non-ASCII characters on the command line without display an error message?

(Note that Unicode is a superset of ASCII. All ASCII characters are also Unicode characters. The customer is making the common mistake of saying Unicode when they mean Unicode-not-ASCII.)

Actually, that error message is not coming from `CreateProcess`. It's coming from the custom application.

> We have the source code for our custom application and it does not display this message. The custom application actually receives the command line just fine (be it Unicode or not), but if there is Unicode in the command line, we get the message above.

The message box may not be coming from your code, but it's still coming from your application. Why not hook up a debugger when the message box is up, then take a stack trace to see whose idea it was to display the message box.

The customer connected a debugger and determined that the message was coming from a third-party library that their custom application uses. Now they know whom to talk to in order to solve the problem.

Raymond Chen

Follow