Why does FindExecutable behave erratically for files with extensions longer than three characters? (And what can you do about it?)

devblogs.microsoft.com/oldnewthing/20160408-00

April 8, 2016

Raymond Chen

The `FindExecutable` function looks up the executable responsible for launching a particular file. This is a dubious undertaking, because it assumes that the thing that launches a file is an executable. There are other things capable of launching a file, such as a DDE command, a context menu shell extension, or a custom drop target. What should `FindExecutable` return in those cases?

Okay, so if `FindExecutable` is based upon a flawed assumption, why does it even exist?

Because at the time it was originally introduced, the assumption was valid.

The `FindExecutable` function comes from 16-bit Windows, and back in those days, there were no context menu shell extensions or custom drop targets. (There was DDE, but that's okay, because programs still have to register an executable to be used in the fallback case when nobody responds to the DDE message.)

In the port to 32-bit Windows, the `FindExecutable` function remains, but it works only in the case where files were registered in the 16-bit way; that is, with a command line executable. It so happens that most file types are still registered that way, so the `FindExecutable` function basically still works.

Since the `FindExecutable` function is basically a throwback to 16-bit Windows, there is another attempt to accommodate the 16-bit world that is not as obvious: The `FindExecutable` function takes the thing you pass and converts it into a short file name before trying to look up the handler.

The effect of the conversion to a short file name depends on a bunch of things.

If the volume does not have short file name autogeneration enabled, then the conversion to a short file name has no effect. But if the volume does have short file name autogeneration enabled, then the net effect is that the extension gets truncated to three characters.

`foo.abcde` becomes `foo~1.abc`. And then `FindExecutable` looks up and returns the handler for the `.abc` extension instead of the `.abcde` extension.

Back in the days before long file names, all file extensions were truncated to 3 characters. if you asked for `foo.abcde`, you got `foo.abc`. The `FindExecutable` function tries to maintain this compatibility with older applications. Newer applications shouldn't be using `FindExecutable` anyway, seeing as the handler for a file type may not even be an executable.

I accept that the concept of finding the executable associated with a file is flawed in the face of handlers that do not take the form of an executable, but I still want to get the executable associated with a file, if possible, with the understanding that the answer may be incorrect.

You can use the `AssocQueryString` function to get the executable associated with the default verb of a file extension, if one exists.

```
HRESULT FindExecutableAssociatedWithFileExtension(
    _In_ PCWSTR extension,
    _Out_ PWSTR resultBuffer,
    _In_ DWORD bufferLength)
{
 return AssocQueryString(ASSOCF_INIT_INGORENUNKNOWN,
                         ASSOCSTR_EXECUTABLE,
                         fullPath,
                         nullptr,
                         resultBuffer,
                         &bufferLength);
}
```

The `ASSOCF_INIT_UNKNOWN` flag says that if the file extension has no handler, don't return the "Open unknown file" handler.

This is not exactly the same as `FindExecutable` because that function has special-case code for when you pass in, for example, `excel.exe`. In those cases, the `FindExecutable` function just returns the file itself, since executables are their own handlers.

The `ASSOCF_INIT_UNKNOWN` flag was added in Windows 7. What do you do for older versions of Windows? Well, you're in luck. Older versions of Windows didn't have the "Open unknown file" handler, so if there is no registered handler, the call will simply fail. (Indeed, the introduction of the "Open unknown file" handler is what most likely prompted the creation of the `ASSOCF_INIT_UNKNOWN` flag in the first place.) As a second mark of good fortune, the flag is ignored by older versions of Windows, so you can go ahead and pass the flag unconditionally: On versions of Windows that support it, it does what you want. And on versions of Windows that don't support it, they already behave the way you want by default.

Raymond Chen

Follow