

Add a shortcut to the right click menu (C#)

Question

I'd like to add a program I've made to the Windows right click menu. However, My Googling for how to do this with the Shell have turned up little thus far.

Hopefully you guys can point me in a helpful direction?

Thanks in advance.

Answer

Don't.

You're not supposed to use .net for shell extensions. IIRC it has to do with the fact that you can't have runtime dependencies on different versions of the .net runtime.

Remulak's right about the reason that you shouldn't do this with .NET. Because this method works only for Windows 95 / Windows 98 (not on XP, Vista, x64 - 64-bit Windows), to add items to Windows Explorer Shell context menu you should use, according to Microsoft guidelines, appropriate .Net component - Windows Explorer Shell Context Menu. What you may also do is to write a launcher program in C/C++ that invokes your program with the arguments you want.

Adding a "dumb" context sensitive shortcut to the right-click menu is ok as its simply a registry entry & wont invoke the underlying application.

Could you elaborate on this (why not to use C#, why a launcher would help, and what a launcher would look like)? I was going to write a C# program which is invoked via shell extensions, but I'll do it this way if that would be bad practice.

Sure. You write a context menu item so that you can invoke your program (or an operation of your program) on a specific file system object, and what the context menu item gives you is the ID file system object that was clicked on. For instance, when you right-click on a text file and select "Notepad" in the "Open With" menu, Explorer raises "%systemroot%\notepad.exe %1" where %1 is the path of the text file you clicked on. Write a launcher program that takes the arguments that Explorer gives it and passes those directly to your C# program.

That's the theory anyway, I haven't actually done this myself. This. The "Don't use .NET" rule only applies to shell extensions, not context menu shortcuts.

There is no need for a launcher! Just follow the instructions in the article prot linked to.

You shouldn't use C# to write shell extensions because of a limitation with current versions of the .NET runtime. Namely, that only one version of the runtime can be loaded into a process at a time.

Suppose you write your shell extension and it uses the 2.0 runtime. When your shell extension is instanced, the 2.0 .NET runtime gets loaded into explorer.exe's address space, and as far as you can tell, everything works fine.

Now, some time down the road, you (or one of your extension's users) installs a different shell extension, built on the theoretical 4.0 .NET runtime; requiring new features in that runtime like automatic dynamic compiler dispatch generic routing. This is where things start to go pear-shaped when it comes to shell extensions. One of two things happen:

- Your extension gets loaded into explorer.exe's process first. Explorer.exe has the 2.0 runtime loaded, and your extension works fine. When the other extension tries to load, it fails because it can't load up the 4.0 runtime; and it requires features the 2.0 runtime doesn't support.

- The other extension gets loaded first, causing explorer.exe to have the 4.0 runtime loaded. The other extension works great, but now when your extension loads, it loads into the 4.0 runtime, where it runs the risk of having subtle bugs introduced as the result of backwards incompatibilities in the runtime.

The above is the reason that Microsoft still considers managed code as verboten for use in any Windows core

components. Of the two possibilities, the latter is the most preferable since backwards-incompatibility bugs are less common than forwards-incompatibility bugs; but unfortunately you can't uncook an egg -- you don't know you need the 4.0 runtime until you try to load the extension that needs it, and by then you've already loaded the 2.0 runtime and can't unload it; and the small, but non-zero risk that there are backwards-compatibility bugs makes it a bad idea to always just load the latest runtime available on the machine.

With the Silverlight build of the CLR, this is beginning to change, however, as Silverlight and all future editions of the full CLR will support side-by-side loading within a process; that is, you can have any number of post-Silverlight runtimes loaded, and up to one of the 1.0/1.1/2.0 CLR. But until that shiny new CLR is upon us, it is highly recommended not to use managed code to write shell extensions; or any plugins for non-managed applications that don't explicitly specify a single version of the runtime to use.

Thanks for the info, guys! What I ended up doing was just going through the folder options File Types dialogue in explorer and adding the context menu item that way for the various extensions it can accept.

This is just a tool for myself, so I don't feel the need to programmatically change the reg keys.

If anyone cares to know, it uses the Microsoft Office Document Imager (MODI) interop services to quickly OCR any image. Or OCR every image in a folder when you use it on a folder. It just outputs a textfile of the same name.

I was too lazy to convert to TIF myself (MODI only likes tif images) so I used Photoshop CS3 interop services for the converting work.

bloated? Yes.

Heavily dependent on third party softwares and versions? Yes.

But it works like a charm and makes my life so much easier.

If anything this has been a fun exercise in making stuff for the context menu, interop programming and an intro to Photoshop scripting, which I've been wanting to dip into for awhile!

Well. There's more than one way to get an entry into a context menu. First of all, which context menu? A specific file type? All files? Just directories? By your description there, it sounds like a few file types. This is possible to do without a shell extension, by adding a verb to the directory or extension, which is actually what you're doing with file types.

For example, this .reg file could add your program to every directory's context menu (assuming your program could understand what to do when passed a dir name as an argument):

```
code:Windows Registry Editor Version 5.00 [HKEY_CURRENT_USER\Software\Classes\Directory]
[HKEY_CURRENT_USER\Software\Classes\Directory\shell]
[HKEY_CURRENT_USER\Software\Classes\Directory\shell\OpenWithProgram] @="Convert images to text"
[HKEY_CURRENT_USER\Software\Classes\Directory\shell\OpenWithProgram\command] @="\"c:\my
program\ocr.exe\" \"%1\""
```

For file types, verbs are added to that extension's progid. Here is an example that adds a command to the context menu of .jpg files. On my machine, .jpg files have a progid of "jpegfile", so that's where the command goes. You can look up an extension's progid by checking the default value of HKEY_CURRENT_USER\Software\Classes\<the extension> key.

```
code:Windows Registry Editor Version 5.00
[HKEY_CURRENT_USER\Software\Classes\jpegfile\shell\OpenWithProgram] @="Convert images to text"
[HKEY_CURRENT_USER\Software\Classes\jpegfile\shell\OpenWithProgram\command] @="\"c:\my program\ocr.exe\"
\"%1\""
```

There's more places to put verbs, too, but that should get you going. Note that you can easily manipulate the registry in code, and you don't need administrative permissions to write to HKCU, either, which is great.

E2: On the topic of shell extensions, it's annoying that I can't write them in C#. I've been thinking, it should be possible to write an unmanaged shell extension that, when loaded, loads a managed library in it's own address space, and passes the calls right to the managed library. This would be possible, right?

This I find incredibly interesting. I've always found it odd that Microsoft wrote an entirely new language and framework and then never actually used it themselves.

Hopefully Windows 7 will change that.

Don't misunderstand, Microsoft is using .NET to develop applications more and more, just not for anything that installs as part of a standard Windows installation -- outside of Windows Media Center, which is kinda an odd duck out.

I'm not sure if the restriction against using managed code in core components will be lifted by Windows 7 or not; but personally I'd think not since they haven't yet shipped or even made overtures that they're planning on shipping soon a side-by-side enabled CLR. Silverlight doesn't qualify since it's not a general purpose CLR build.

I think it'll still be another couple Windows revs before we see managed code used by the OS itself. They'll want the newer tech out there first, and proven.

new question: I would like to make a multi level item on the menu- an item with a sub menu. How would I go about doing this? Most searching I do refers to a IContextMenu object for .NET

All articles and tutorials I've come across in my googling have either been for adding a single item or adding a multi level item to the Internet Explorer context menu.

This is going to require a shell extension, which means no .Net.