### [Creating a new DXCore Plugin](http://rorybecker.blogspot.com/2009/04/creating-new-dxcore-plugin.html)

## Monday, April 13, 2009

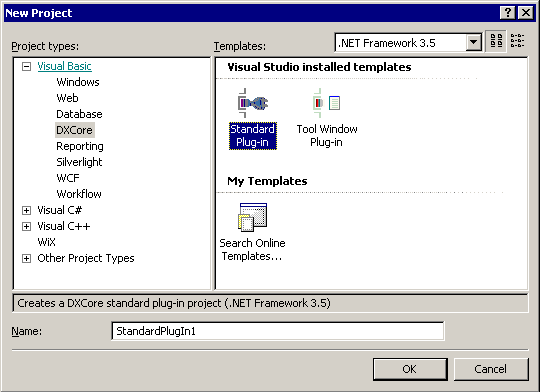
[**Updated**: This post has been updated to account for some small changes in the 9.1.3 release]

There are certain steps you must follow in order to create a DXCore plugin of any kind. This blog post will outline and explain those steps.

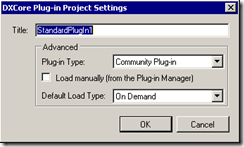
So the first step is to create the right sort of project.

If all is installed correctly you should be able to find the 2 available projects (‘Standard Plug-in’ and ‘Tool Window Plug-in’) in the DXCore section under either the Visual Basic or Visual C# sections in the New Project dialog.

Initially I’ll suggest that you pick ’Standard Plug-in’ as you can always add the Tool window later if you need to.

[](http://lh5.ggpht.com/_qRWG2XybUkY/SeOY-cacpNI/AAAAAAAAAT0/aydeDKfmmbw/s1600-h/FileNewDXCorePlugin4.png)

Once you’ve given your plugin a Name, and hit Ok, you’ll be presented with the Plug-in Project settings dialog.

[](file:///C:\Documents%20and%20Settings\Rory\Local%20Settings\Temp\WindowsLiveWriter-429641856\supfiles2E6746B\InitialPluginSettings9139.png)

**Plug-in Type: System Plug-in:**will cause your plugin to be deemed a ‘System plugin’ Duh! But what does this mean? Simply put, system plugins are loaded ahead of non system plugins. 90% of all plugins have no need to do this… I have yet to use this facility in any plugin I have written.

**Plug-in Type: Plugin:** Will cause your plugin to be built to “IDETools\System\DXCore\Bin\PlugIns\”. I’m guessing that this is something that [DevExpress](http://www.devexpress.com/) do themselves. Although I see no reason not to use the Community option.

**Plug-in Type: Community Plug-in:**is a new option with version 9.1.3. This is due, from what I can see, to the new Community Plugin Path option. This option sets the compile path of your new plugin, to your current community folder (Setup when you first loaded CodeRush or RefactorPro or DXCore after installing 9.1.3.) This means that, if you accepted the default, your plugin will exist under your ‘My Docs’ folder and as such will not require you to Launch Visual Studio with Admin rights in order to compile your plugin. – **You should pretty much always pick this option**.

**Checking ‘Load manually’**will cause an entry to be added to the ‘plugin manager’ (another plugin), which in turn will prevent the DXCore from automatically loading your plugin.

Why would you do this?

Well, suppose that whilst developing your plugin, you are suddenly required to do something else. (Heaven forbid that anything should come ahead of this most important work). What you do is you go and immediately fire up another copy of VS and you load what ever you need into that session and off you go.

The trouble is that (assuming that you’ve compiled your plugin at least once) DXCore has now loaded a copy of your unfinished/unpolished plugin into memory. This might not be a problem but there are 2 ways I can think of that might mean it is…

1.> You’re writing the most killer refactoring known to man, and you’ve just completed the ‘DeleteAllThisUselessCrap’ routine but haven’t quite gotten around to writing the ‘InsertGloriouslyShineyCode’ and if activated, without taking suitable precautions then you could be [up the swanny](http://en.wikipedia.org/wiki/Swanny).

2.> You might want to tab back to to some more development on your plugin whilst you’re waiting for the results of your latest check-in on your ‘Important project’, but unfortunately the copy of VS hosting your ‘Important Project’ is maintaining a lock on your plugin’s dll and as such you’ll be unable to compile it, much less test it.

So in order to avoid these scenarios, I always tick the ‘Load Manually’ tick box which then ensures that the DXCore never \*automatically\* loads my plugin. Then when I launch a copy of VS for testing it, I locate it in the plugin manager and double click it in there. At this point DXCore wakes up, notices the plugin and loads it. In doing this you have only loaded your plugin into the address space of a single instance of VS, which exists entirely for the purpose of testing your plugin. This also means that when you shut down this single copy of VS, you’ll be able once again to resume development of your plugin. Great huh?

Note: This setting is local to your development installation only and will not affect any users of your plugin.

**Default Load Type**

I asked AlexZ (one of the great brains behind the DevExpress IDE Tools) about the ‘Default load type’ setting (and most everything else as well :) – Alex is a great source of info) and he had this to say:

**On Demand** - plugin is loaded when some event happens for which plugin should work. e.g. if you plugin contains some action - then it will be loaded only when action is being executed. Or it may define context or string provider or any other provider - and plugin will be loaded when provider is really accessed.   
**On Idle** - Plugin will be loaded on editor idle, IOW when VS editor is being idle.   
**At start-up**- Plugin will be loaded on start-up - however it will add time to overall loading time.

So which one should you select.. well the answer is it depends.. Personally I typically leave the default of **‘On demand’** in place as this means your plugin should not affect the start-up time of VS. This might mean that you have a minor slow down as you activate your plugin for the first time, but it will not contribute to slowing down the start up time of VS.

So to summarize… I recommend that you tick the ‘Load manually’ option and leave everything else (except perhaps the name) exactly as default … then hit Ok

At this point VS churns away and provides you with a nice clean plugin project ready for you to create the greatest plugin known to man… after all .. that’s what you were going to do right :)

I am now contractually obligated to make you wait until the next instalment before I reveal anything else :)