Back to Basics: Tips for greater developer productivity

By Bill Wagner
May 2010

Visual Studio 2010 is out, and this is a good time to look at some of the default behavior in Visual Studio, and the quickest path to modifying those defaults to suit your (or your company's) standards. After all, Visual Studio is not custom software for you or your organization. Instead, it contains many features that enable very rich customization.

Many of the features I'm discussing are not new to Visual Studio 2010. If you or your organization aren't upgrading immediately, most of these tips are still available to you.

For almost all developers, a new project means running one of the Visual Studio wizards to create the starter code. There are many things to like about wizards. All the Visual Studio wizards generate code a lot faster than I can type code by hand. The wizards also do a good job of setting reasonable defaults for the project settings and the code structure. However, it's not exactly what I want. In some cases, I always want the same changes. In other cases, I want different behavior on a case by case basis.

I'll walk you through the process, and then show you how Visual Studio executes project and new item wizards. That will show you how to modify what Visual Studio does when you execute one of the standard templates. You'll learn how to modify existing templates, and how to create your own item and project templates to match your own coding standards and preferences.

You've seen this code thousands of times. I created a C# class library, named "DevCenterSample", and here's the code for class1.cs:

```
using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;



namespace DevCenterSample

{

    public class Class1
```

```
    {

    }

}
```

Like every new project, the default namespace comes from the project name, and you have an initial class named “Class1”.

There are four changes I make every time. You probably do as well. I make them in a particular way because of how Visual Studio responds to making those changes.

The first change is to rename the class. When you do that, change the file name instead of the class name. Visual Studio knows that example.cs should contain a class named Example. If you change the file name from class1.cs to Sample.cs, Visual Studio asks you whether you want to change the class name from Class1 to Sample. That’s a small time saver, but it adds up.

Next, I always change the namespace for the project. I do this by using the "Rename" refactoring in Visual Studio. Once again, the reason is that Visual Studio does a little more work for me by making my changes in this way. Look at Figure 1, which shows the current application settings, including the default namespace.

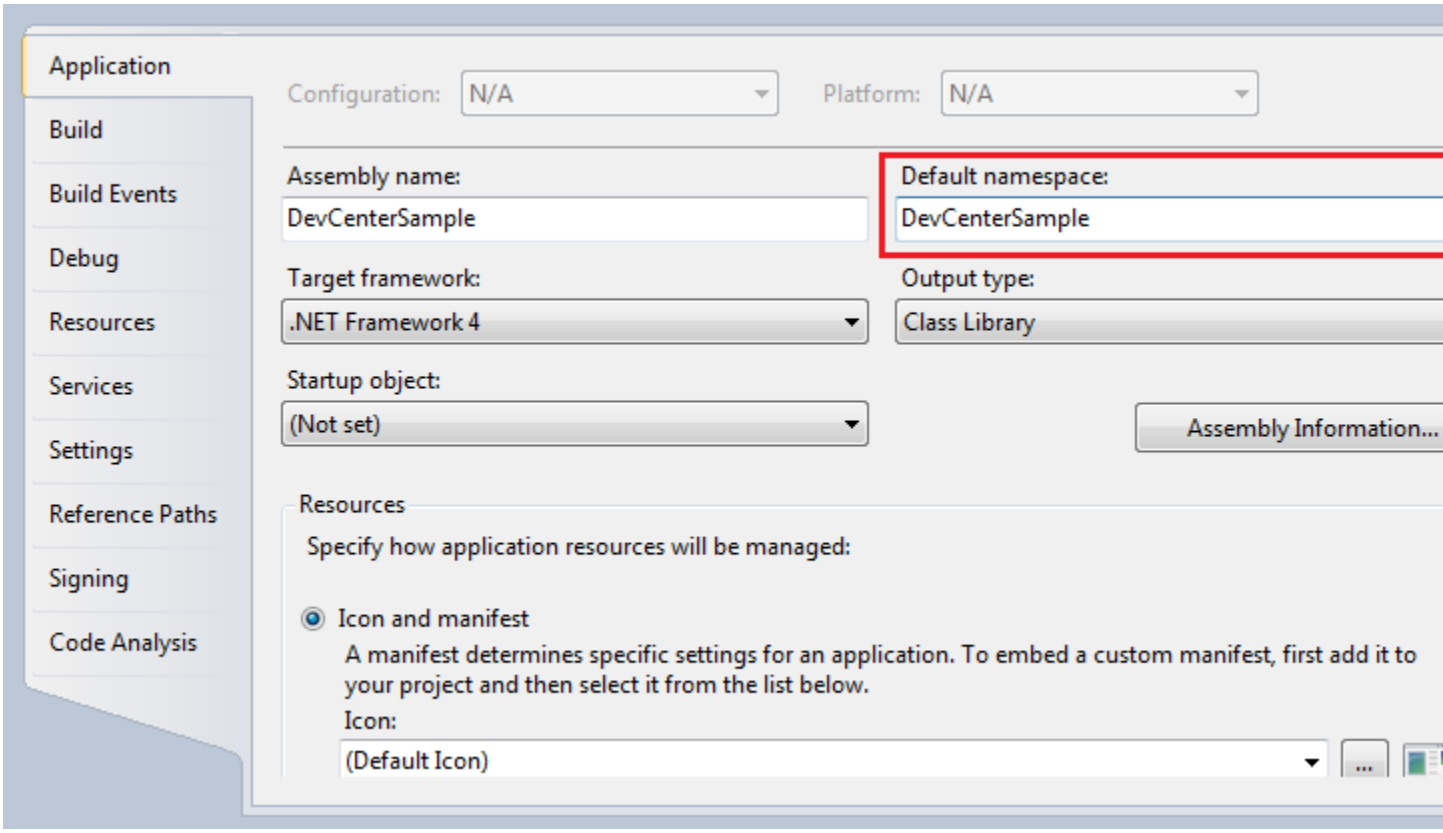

Figure 1. Default project settings

Right-click the namespace in your class1.cs file and select Refactor:Rename. Pick a new name. My preference for most projects is to use <companyName>.<Project> as the default namespace. Some classes inside a project may be placed in sub-namespaces for organizational reasons. For illustrative purposes, I'll use "DevCenter.Article.Sample". When you use the Visual Studio refactoring, it will also change the default namespace for all new elements in your project. After you make that refactoring, the project settings now reflect that new namespace:

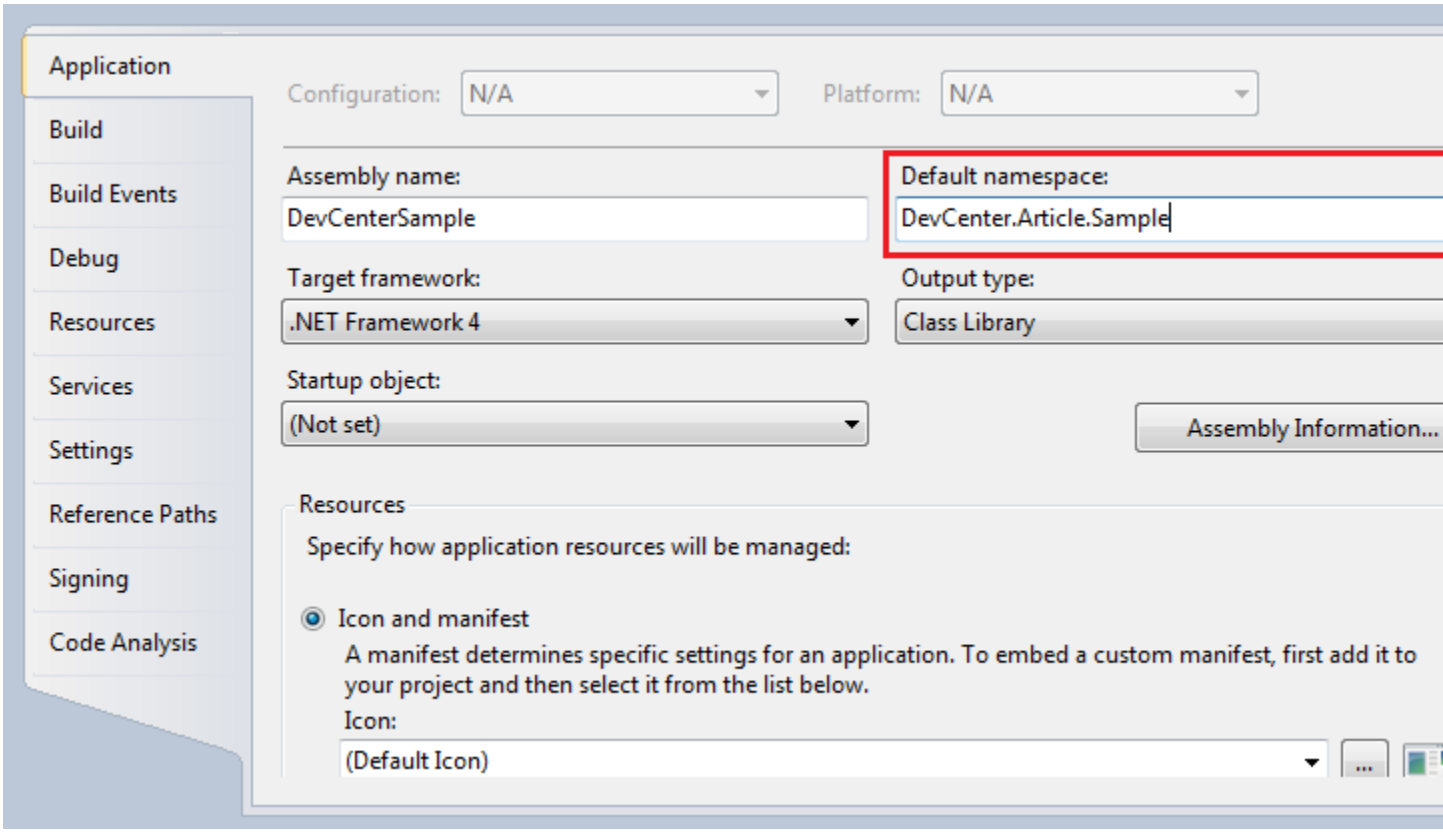

Figure 2. Updated Project Settings

That takes care of two of the changes. The third is to modify the using statements. I prefer to have the using directives inside the enclosing namespace. In addition, I prefer to qualify the namespace with the global alias qualifier. That means my first class looks like this after these modifications:

```
namespace DevCenter.Article.Sample

{

    using global::System;

    using global::System.Collections.Generic;

    using global::System.Linq;

    using global::System.Text;
```

```
    public class Sample
    {
    }
}
```

With the new block selection and editing feature in Visual Studio 2010 makes, it's much easier to make these changes. This MSDN Blog entry gives a demo. Briefly, if you hold the ALT key, you can select a block across lines. For this purpose, select the point before each of the namespaces. Then, type "global::" once, and you'll see it appear on all four lines.

I prefer putting the using directives inside the namespace for a file. It ensures that a particular namespace is only imported into the global namespace inside the enclosing namespace. The reason for prefacing each namespace with the global alias qualifier is that it ensures that I don't accidentally refer to the wrong namespace. Eric Lippert discussed both of these issues in this blog post.

Finally, look at AssemblyInfo.cs. The section that I always modify is here:

```
[assembly: AssemblyTitle("DevCenterSample")]

[assembly: AssemblyDescription("")]

[assembly: AssemblyConfiguration("")]

[assembly: AssemblyCompany("Microsoft")]

[assembly: AssemblyProduct("DevCenterSample")]

[assembly: AssemblyCopyright("Copyright © Microsoft 2010")]

[assembly: AssemblyTrademark("")]

[assembly: AssemblyCulture("")]
```

Obviously you should change most of these. You'll only see the Microsoft name in the copyright and company string if you've installed the prerelease versions of Visual Studio. The prerelease versions have the registration keys and other associated license fields already set, so you don't get to change that information. If you've only installed released versions and you enter the company name you want for the default, you'll likely never have to change this. I mention it this little bit of changes because you should know where Visual Studio gets its information when I explain the next set of changes.

None of these changes take that much time. But the more you create new code files or new projects, the more that time adds up. When you find that you're repeating the same work over and over, it's time to figure out how to automate it. Visual Studio uses templates (both Item Templates and Project Templates) to control what code gets generated for new projects or new items. In fact, everything is controlled by template files: what assemblies to reference, project settings, code files, and other files for web and other project types. None of these elements are hard coded. You can create your own versions of any of the common templates when you find yourself modifying the code generated by Visual Studio.

Visual Studio provides all the mechanisms you need to create new project or item templates. All the information you need is in this MSDN article, so I won't go into explicit detail on how to create new project or item templates. However, I will offer a couple cautions to you before you modify the standard templates. When you read the process to create a new item or project template, you may be tempted to modify the existing Visual Studio templates. They are, after all, just text files stored on your hard drive. I discourage this solution because it is very brittle. Microsoft does not support this scenario, and you'll soon learn why. If you modify the default templates, upgrades to those components will overwrite your changes, and you'll need to recreate your changes and merge them into any updates you've made.

Visual Studio has many ways to do common tasks. Many times, you'll find that one way provides some extra functionality over others. I've shown you a couple of those instances and you should remember to leverage them when you modify your own projects. In addition, I've pointed you toward techniques to create your own versions of project and item templates. When you find yourself repeating the same modifications to new items in your projects, create your own custom templates to automate those changes. Have fun with Visual Studio 2010.