



## Create Boilerplate Templates

Save time and repetitive coding by customizing the code contained in new project files you create in VS.NET.

by Jonathan Goodyear



September 2002 Issue

Technology Toolbox: C#, ASP.NET

Visual Studio .NET has many hooks you can use to customize the way it behaves. If you're frustrated by something—such as the limited boilerplate code VS.NET's New Project Item Wizard produces (see [Listing 1](#))—there's usually a way to customize it to suit your needs. In this column, I'll show you how to customize the code VS.NET creates for files such as class libraries and ASP.NET Web Forms by stepping you through the process of creating your own project item template. You'll save yourself time, trouble, and repetitive coding, and give your future .NET projects increased standards compliance.

Each of the New Project Item Wizards has a template contained under the VS.NET file path. I'll assume you didn't modify your install path when you installed VS.NET, and that it's located at C:\Program Files\Microsoft Visual Studio .NET\. I'll call this folder the root folder. The root folder contains a subfolder for each supported .NET language that's installed. I'll show you how to create a C# template in this article, but you can mirror the instructions to create a Visual Basic .NET template as well.

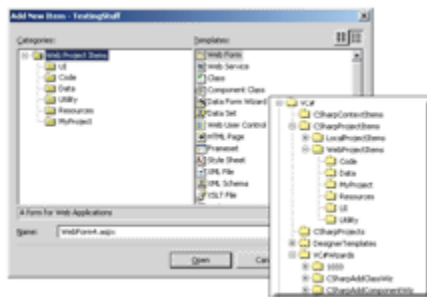


Figure 1. [Add Your Own Folder.](#)

The VC# subfolder under the root folder contains two subdirectories you'll need for this project: CSharpProjectItems and VC#Wizards. The CSharpProjectItems subfolder contains a LocalProjectItems subfolder and a WebProjectItems subfolder. Expand the WebProjectItems subfolder because you'll be building a Web application. Notice that this folder and all its subfolders match up exactly with VS.NET's Add New Item Wizard. Create a new folder named "MyProject"; it's added to the wizard (see [Figure 1](#)). However, if you click on the MyProject folder in the wizard, you'll see that it's empty. To add your own templates, you need to add a file to your MyProject folder named MyProject.vmdir. This is a VS.NET directory file, and you can use it to point VS.NET to the

wizards to build your file templates. You can expedite the process by copying the WebProjectItems.vmdir file from the WebProjectItems folder and renaming it.

Open the MyProject.vmdir file in Notepad; it contains a cryptic-looking listing for each type of file you can create using the Add New Item Wizard. Make things simple for now, and remove all the file types except for the ones that begin with ".\CSharpAddWebFormWiz.vsz" and ".\CSharpAddClassWiz.vsz," including the lines that start with UI, Code, Data, Utility, and Resources. Your file should now have only two rows of data in it. It's never a good idea to overwrite any of the existing templates, so append ".\MyProject" to the beginning of each of these wizard filenames, so that now the rows begin with ".\.\MyProjectCSharpAddWebFormWiz.vsz" and ".\.\MyProjectCSharpAddClassWiz.vsz." You need the extra ".\" because your folder is one level deeper in the directory structure, and these are paths to actual wizard files (which I'll talk about a bit later).

### Be Careful What You Change

At this point, you might be tempted to fiddle with some of the other pieces of data for each row in the file. Be careful when you do this. The Globally Unique Identifier (GUID) you see on each line is the

GUID for C# itself and must remain there. The numbers with pound signs (#) next to them are indexes into a private resource file that determines the name and description of the item that's displayed in the Add New Item Wizard.

You can change your custom project item's name by substituting the first index value with a textual name. For now, change the first index values of your custom project items to MyProjectClass and MyProjectWebForm. Be sure to remove the pound sign from the index value as well. The second index value with the pound sign controls the extended description that appears in the gray box below the treeview in the Add New Item Wizard. Change the index values to "Class library Customized for MyProject" and "Web Form Customized for MyProject."

The fourth item in the bar (|) delimited list of data is a number that determines the order in which the items are displayed in the wizard. The item with the lowest number is positioned at the top, and so on. Set these ordering index values to 20 and 10, which makes MyWebForm appear first in the Add New Item Wizard. The only other piece of data that makes sense to customize is the last item on each row. Each of the files created by your template gets this default name unless you change it in the Add New Item Wizard (numbers are appended to the end to prevent duplicate filenames). To demonstrate, change the Web Form default name to Screen.aspx, and the class library default name to LogicClass.cs.

Earlier, you changed the names of the wizard file pointers in your MyProject.vmdir file. Notice how each wizard file pointer begins with "..\..\"? Well, traverse two levels up the directory tree to the CSharpProjectItems folder, and you'll find a file with a VSZ file extension for each template in the Add New Item Wizard. Make a copy of the CSharpAddWebFormWiz.vsz file and rename it to MyProjectCSharpAddWebFormWiz.vsz. Do the same thing for the CSharpAddClassWiz.vsz file. Inside each of these files is the information necessary to launch your template wizard:

```
VSWIZARD 7.0
Wizard=VsWizard.VsWizardEngine
Param="WIZARD_NAME ="
    "MyProjectCSharpAddWebFormWiz"
Param="WIZARD_UI = FALSE"
Param="PROJECT_TYPE = CSPROJ"
```

The first line indicates that this wizard is meant to run using VS7 (numbered for compatibility with future versions of VS.NET). The Wizard item indicates the wizard engine to use. In the three rows that begin with "Param=", the WIZARD\_NAME parameter is the name of the wizard folder containing the template code (which you'll create next). The WIZARD\_UI parameter lets the wizard engine know whether the wizard has a user interface. In your case, it doesn't. The PROJECT\_TYPE parameter indicates the language of the project this wizard is designed for. If this were a VB wizard, this parameter would be VBPROJ instead of CSPROJ. The only thing you must do in each of these files is append "MyProject" to the WIZARD\_NAME parameter.

### Create the Boilerplate Code

You get to create your template's boilerplate code now that you've finished all the plumbing. Earlier I mentioned the VC#Wizards folder directly under the root folder. If you click on this folder, you'll notice it has a subfolder corresponding to each wizard file (with the VSZ file extension) in the WebProjectItems folder. The VS.NET wizard engine knows which folder to look in by examining the WIZARD\_NAME parameter in the wizard files you just created. Find and make a copy of the wizard folders for CSharpAddWebFormWiz and CSharpAddClassWiz, appending "MyProject" to each one.

Each of your wizard folders has two subfolders, Scripts and Templates. The Scripts folder contains the default.js file (under another subfolder named "1033"). Don't modify this file, because it uses symbols that are private to VS.NET. These symbols can change at any time, so you risk breaking your template if you change anything in that file. Forget it exists. The Templates folder (actually, the "1033" subfolder underneath it) contains the files you can modify. The MyProjectCSharpAddWebFormWiz wizard presents you with two files. The Templates.inf file merely points to the files needed for the wizard (only one in the case of both your new wizards). You don't need to rename the file listed in Templates.inf, because VS.NET extracts the proper name for your new files from the MyProject.vmdir file you built earlier.

Open the WebForm1.aspx file and add any custom code you want. You'll notice a few symbols such as \$FILENAME\$, SAFE\_ITEM\_NAME, DEFAULT\_CLIENT\_SCRIPT, and DEFAULT\_HTML\_LAYOUT. You can change or remove these, but I wouldn't recommend it. VS.NET

uses them to wire up the templates to the file/class names that you specify in the Add New Item Wizard, as well as your VS.NET IDE preferences.

As an example, I customized the <title> HTML tag with the name of a fictional business (XYZ Company) and the project name (MyProject). I also added a default page title and <hr> tag, as well as a copyright message at the bottom (see [Listing 2](#)). In a real-life project, you'd want to add any code that every page needs to implement but might be forgotten otherwise. Open up the NewCSharpFile.cs file in the "\Templates\1033" subfolder of the MyProjectCSharpAddClassWiz wizard folder. You might want to replace the SAFE\_NAMESPACE\_NAME symbol with the namespace you want each of your classes to use, but you should leave the SAFE\_CLASS\_NAME symbol alone, because it corresponds to the filename you specify in the Add New Item Wizard. Some appropriate things to add to class templates are commonly used namespace references, default comments (including company and project name), licensing disclaimers, and default exception-handling logic (see [Listing 3](#)).

Now that you've finished your wizard template, try it out in VS.NET. If you have VS.NET open already, shut it down and restart it. The IDE caches your previous wizard settings, so you want to force them to be reloaded with your new wizard options. Open a C# ASP.NET project, and open the Add New Item Wizard. Your MyProject folder should be there. Click on it and select the Web Form option. Notice that it indicates Screen1.aspx as the default filename. Click on the Open button. You should be presented with a Web Form that reflects the boilerplate code you added to your wizard template (see [Figure 2](#)).

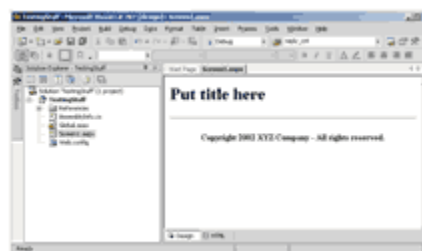


Figure 2. [Add Boilerplate UI Code.](#)

The steps you've taken to create your wizard template might seem like a lot of trouble to go through, but remember all the time having a boilerplate template will save you. I hope I've also encouraged you to explore VS.NET to find some of the neat (and often undocumented) things you can do with it. Remember to always make backup copies of any configuration files before you modify them, in case something goes wrong and you need to revert back to the way things were before.

### ***About the Author***

Jonathan Goodyear is the president of ASPSoft ([www.aspssoft.com](http://www.aspssoft.com)), an Internet consulting firm based in Orlando, Fla. He's MSCD-, MCP-, and CLS-certified and is the coauthor of *Debugging ASP.NET* (New Riders Publishing). Reach him at [jon@aspssoft.com](mailto:jon@aspssoft.com) or through his angryCoder eZine at [www.angryCoder.com](http://www.angryCoder.com).