

T4 TEMPLATES FOR THE .NET COMPACT FRAMEWORK

Jeff Doolittle, CTO (jeffd@sjvtech.com)

SJV Technology Group, LLC (<http://www.sjvtech.com>)

PURPOSE

The Microsoft Text Template Transformation Toolkit (T4) is a tool which you can use to perform custom code generation in for your applications. If you want to learn more about T4, please view this topic in the MSDN library:

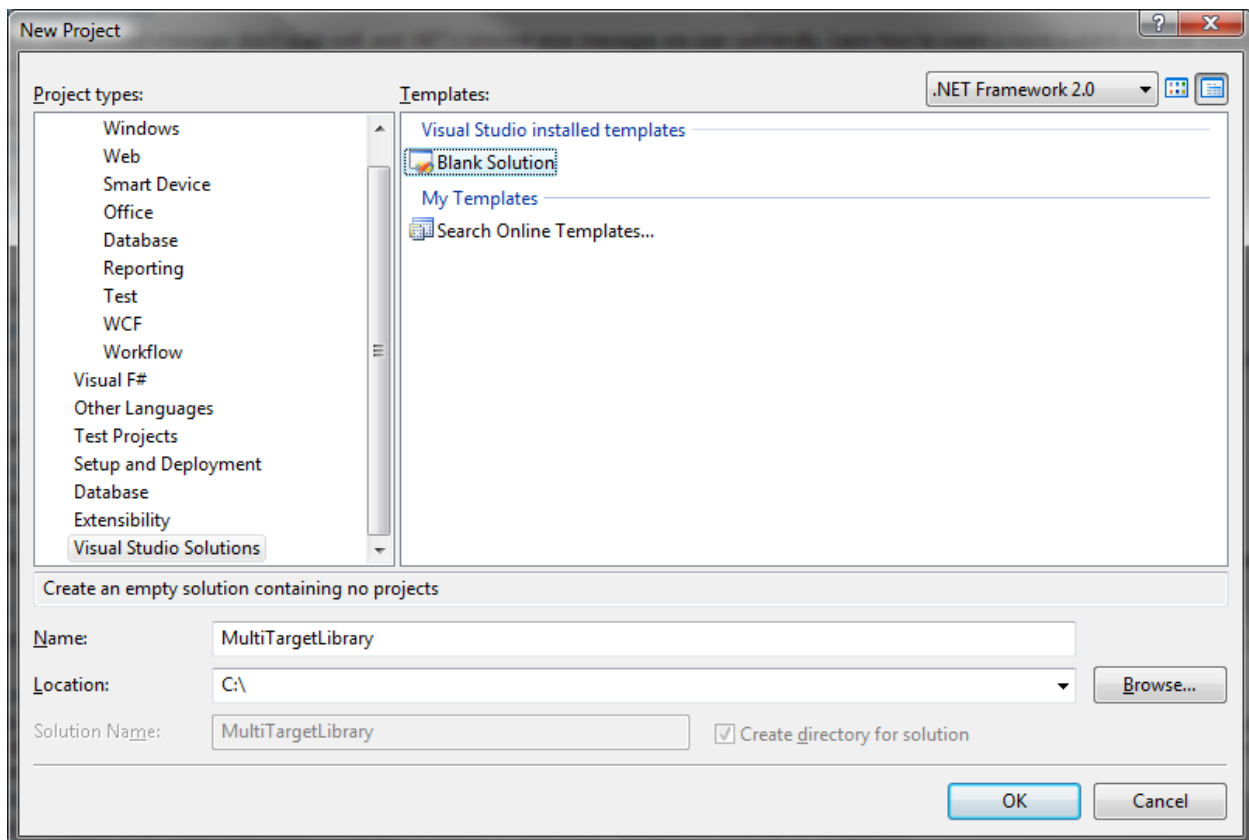
<http://msdn.microsoft.com/en-us/bb126445.aspx>. This guide will assist you in creating a Visual Studio solution that is capable of leveraging T4 templates in a .NET Compact Framework application.

RATIONALE

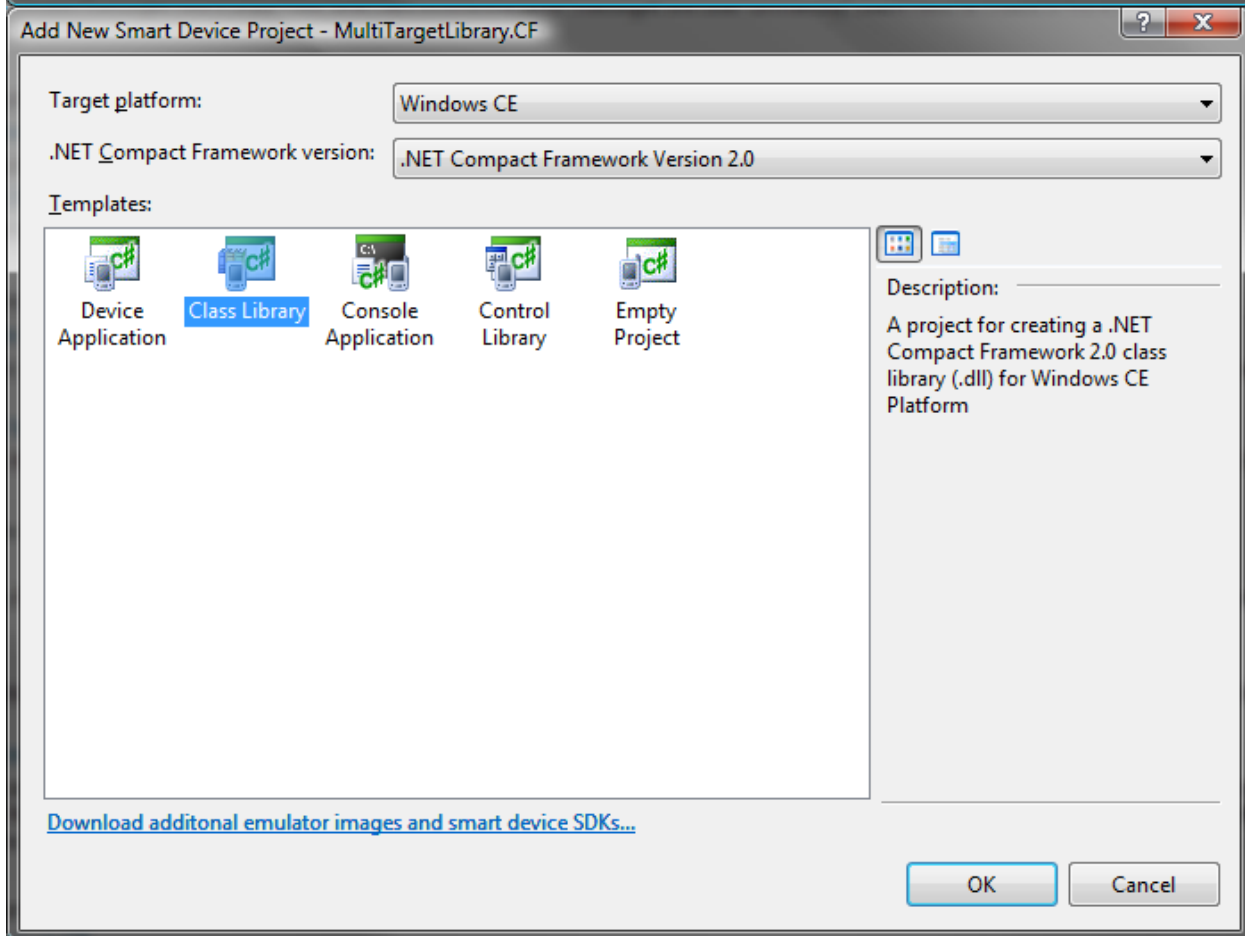
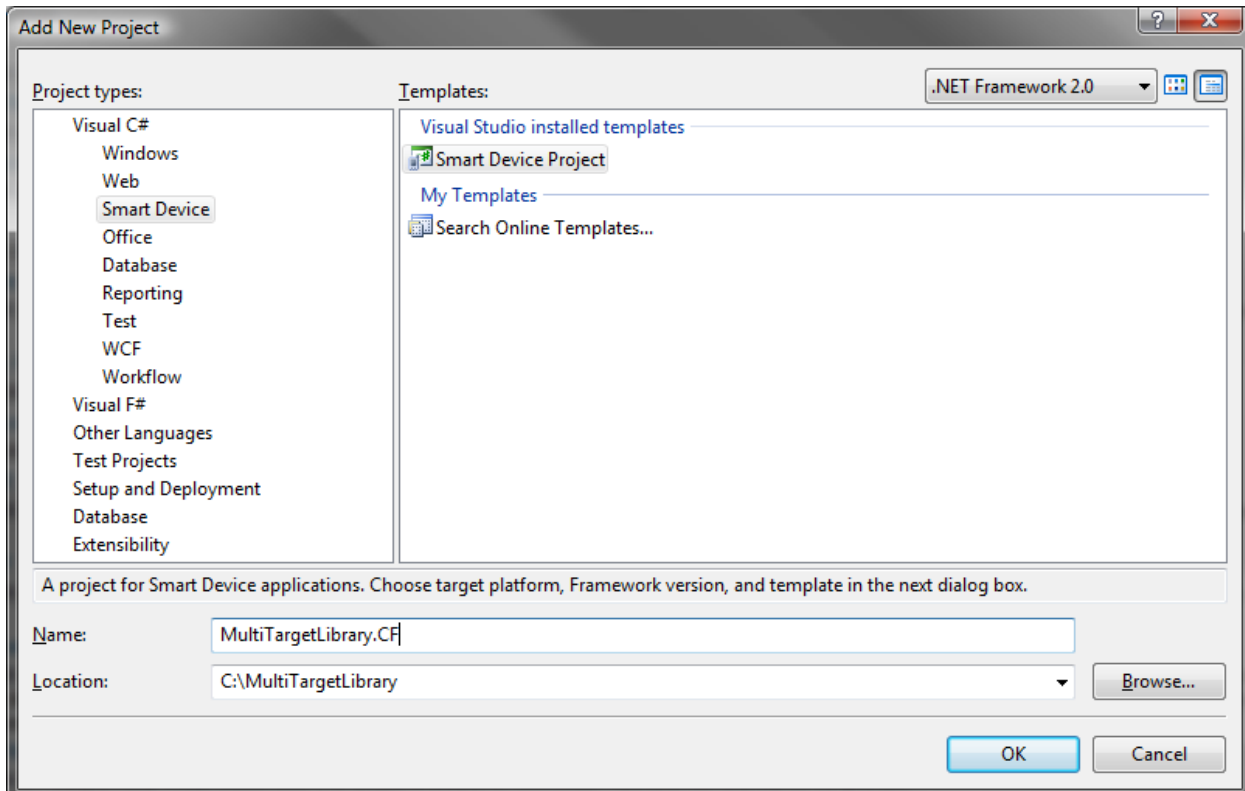
The power of templating and code generation are not available “out-of-the-box” in a Smart Device project. These projects target the .NET Compact Framework (CF), so they cannot access members of the System.CodeDom namespace. But System.CodeDom is necessary for the T4 engine to function. However, with a fairly simple work-around, you can bring the power of T4 templates to CF applications.

PROCESS

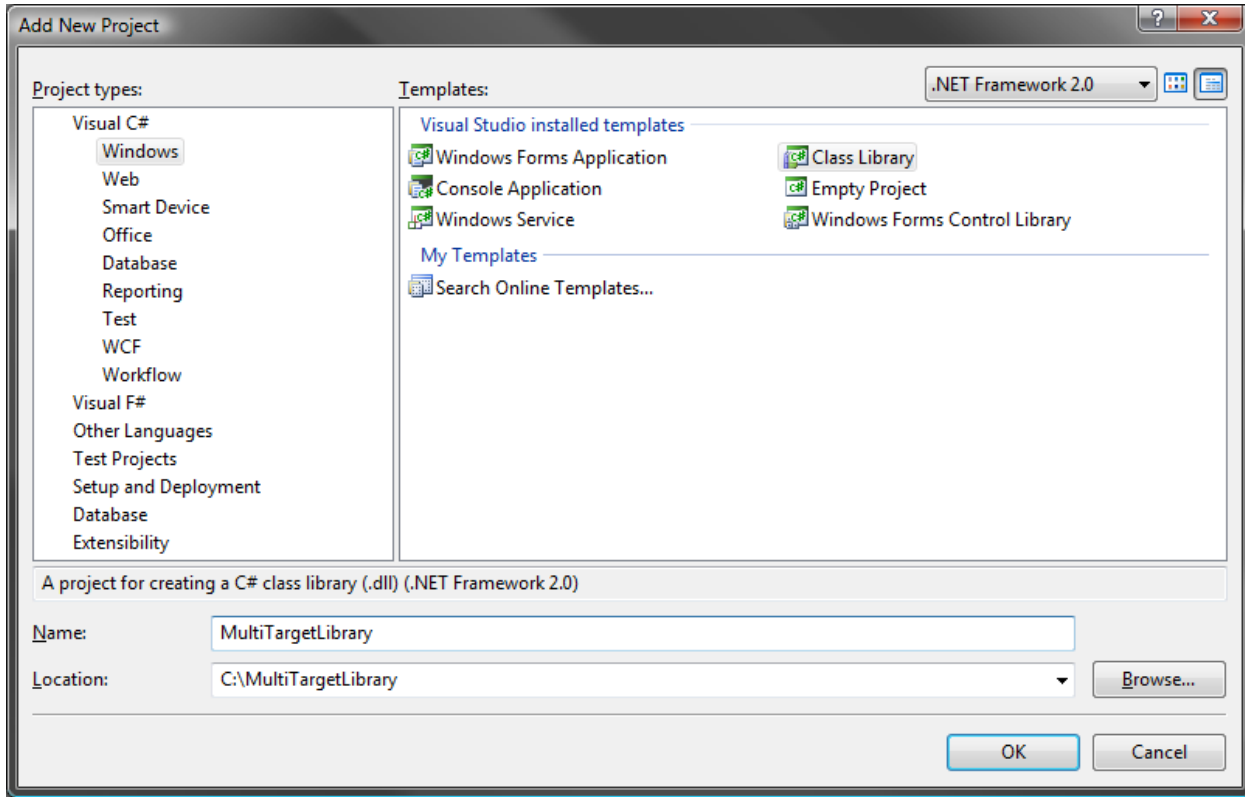
Create a new blank solution:



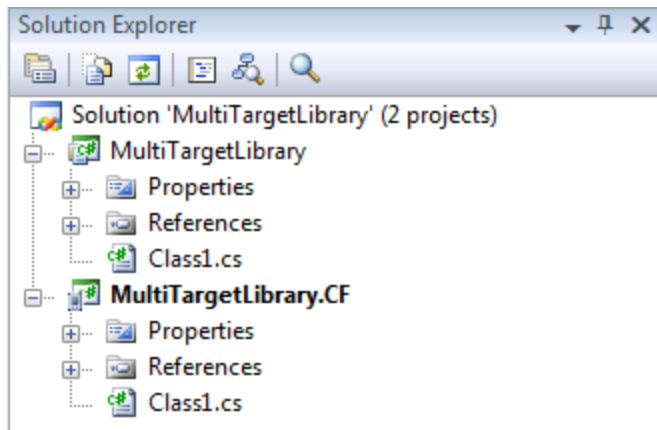
Create a new Smart Device project for the platform and CF version of your choice



Add a new Class library project for the full .NET Framework



Your solution should look like this:



Remove the “Class1.cs” files from both projects and then quit from Visual Studio. Now we’re going to switch gears a bit and make some changes in the file system, outside of Visual Studio.

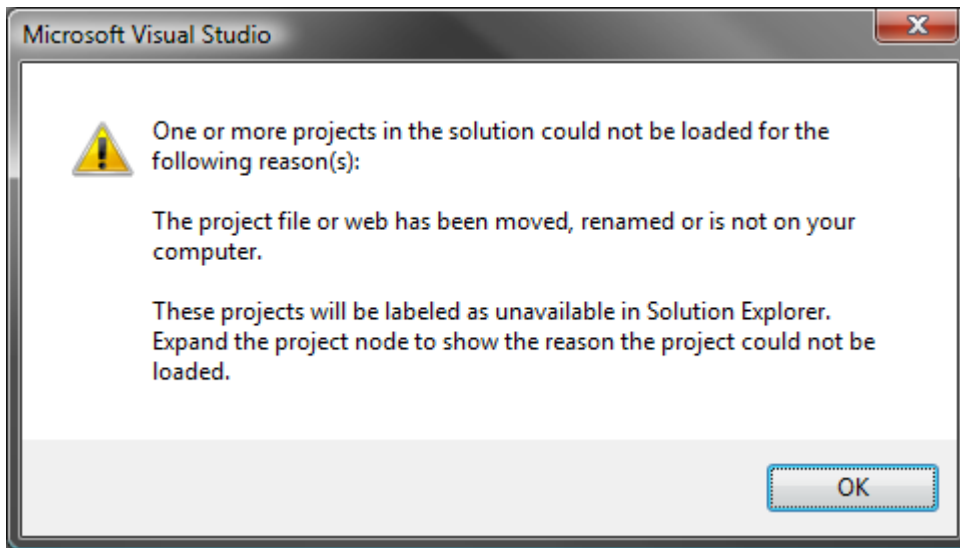
Now you have two separate projects in two separate folders. Our goal is to unify these projects so they exist in the same folder. This will allow them to share files between projects. Otherwise, you’d have to copy/paste generated code files between projects—a frustrating and potentially error-prone extra step. Another note to consider here is the fact that the AssemblyInfo classes generated by Visual Studio are slightly different for CF projects versus full .NET framework projects. Because of this, we’ll need to perform some renaming tricks in order to get each project to reference its appropriate AssemblyInfo code file. When you have finished these steps, you will still have two separate projects in your solution, but you will only have one project folder in the file system.

Using Windows Explorer, open the folder containing your solution and then perform the following steps:

- 1) Change the file name of <root>\MultiTargetLibrary\MultiTargetLibrary.CF\Properties\AssemblyInfo.cs to AssemblyInfo.CF.cs
- 2) Move this file into the folder <root>\MultiTargetLibrary\MultiTargetLibrary\Properties
- 3) Move the file <root>\MultiTargetLibrary\MultiTargetLibrary.CF\MultiTargetLibrary.CF.csproj to the folder <root>\MultiTargetLibrary\MultiTargetLibrary so that it is in the same folder as your other project file
- 4) Delete the folder <root>\MultiTargetLibrary\MultiTargetLibrary.CF

Now we'll return to Visual Studio to complete the process.

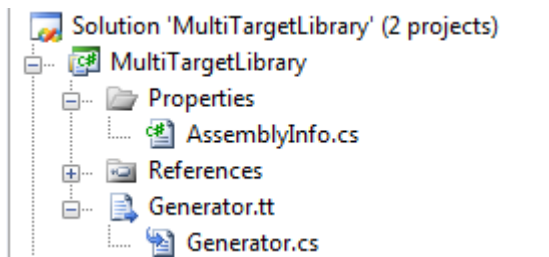
- 1) Open your solution again in visual studio.
- 2) Click "OK" when you see the following message:



- 3) Change the file path of the MultiTargetLibrary.CF project so that it references the new location of the .csproj file (use the Visual Studio "Properties" window to make this change. If you can't see the Properties window, just press the F4) key.
- 4) Right click on the MultiTargetLibrary.CF project and click "Reload Project"
- 5) In Solution Explorer, click the "Show All Files" button so you can see files not currently included in the project



- 6) Exclude (don't delete!) AssemblyInfo.cs from the MultiTargetLibrary.CF project and Include AssemblyInfo.CF.cs in this project
- 7) Add a new Text File to the MultiTargetLibrary solution called "Generator.tt"
- 8) After you have created this file, the T4 template engine should have generated a .cs file called "Generator.cs"



- 9) Select the MultiTargetLibrary.CF project, and click twice on the “Show All Files” button
- 10) Include the file “Generator.cs” in the MultiTargetLibrary.CF project (you can click “Show All Files” again at this point if you so desire to hide unnecessary files)
- 11) Build your solution and voila! Templating for Compact Framework is now available to you.

As a test, try pasting the following code into the Generator.tt file:

```
<#@ template language="C#" #>
// <autogenerated>
// This code was generated by a tool. Any changes made manually will be lost
// the next time this code is regenerated.
// </autogenerated>
using System;

namespace <#= this.Namespace #> {
    public class <#= this.ClassName #>
    {
        public static void HelloT4()
        {
            Console.WriteLine("Hello, T4 4 CF!");
        }
    }
}
<#+
string Namespace = "MultiTargetLibrary";
string ClassName = "MyClass";
#>
```

The code generated by the template is now available to you in both project files.

RECOMMENDATIONS

- 1) Change the default namespace for the MultiTargetLibrary.CF project to “MultiTargetLibrary” so that both projects have the same default namespace.
- 2) Change the build output folders for both projects and for both Release and Debug mode to the following:
 - a. MultiTargetLibrary (DEBUG): ..\bin\debug-net\
 - b. MultiTargetLibrary (RELEASE): ..\bin\release-net\
 - c. MultiTargetLibrary.CF (DEBUG): ..\bin\debug-netcf\
 - d. MultiTargetLibrary.CF (RELEASE): ..\bin\release-netcf\
- 3) Change the Assembly name of the MultiTargetLibrary.CF project to “MultiTargetLibrary”

CONCLUSION

With the power of T4 templates available to you for the Compact Framework, you can now leverage code generation and programming best practices in your Smart Device applications.