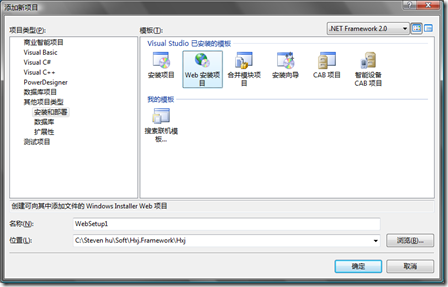
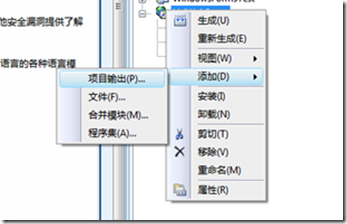
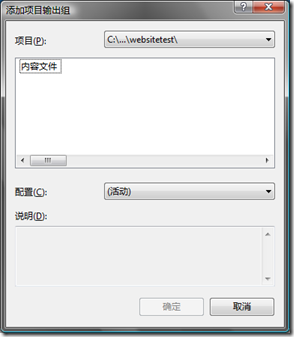
在测试Web项目中新建一个Web安装项目，如下图：

[](http://images.cnblogs.com/cnblogs_com/huxj/WindowsLiveWriter/d24a94621d11.Net_C7AE/image_2.png)

在新建的安装项目，右击项目如下图：

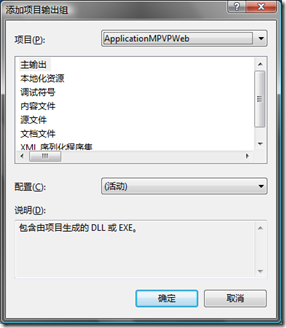
[](http://images.cnblogs.com/cnblogs_com/huxj/WindowsLiveWriter/d24a94621d11.Net_C7AE/image_8.png)

选择项目输出，选择要输出的项目，如下图：

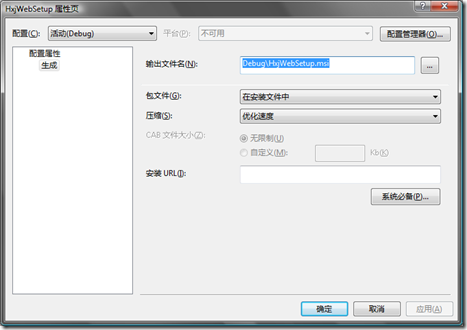
[](http://images.cnblogs.com/cnblogs_com/huxj/WindowsLiveWriter/d24a94621d11.Net_C7AE/image_10.png)

我选一个Web网站，则只有内容输出选项，选中内容文件选择确定。

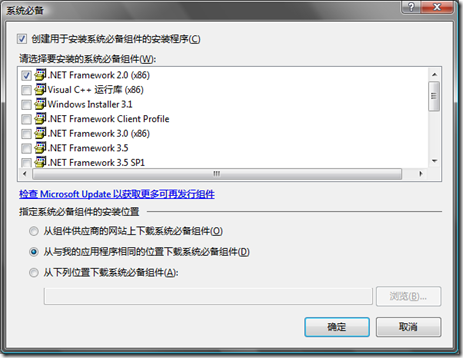
如果选Web项目，则如下图：

[](http://images.cnblogs.com/cnblogs_com/huxj/WindowsLiveWriter/d24a94621d11.Net_C7AE/image_12.png)

右击项目点击属性，则出现如下图：

[](http://images.cnblogs.com/cnblogs_com/huxj/WindowsLiveWriter/d24a94621d11.Net_C7AE/image_20.png)

再点击系统必备：

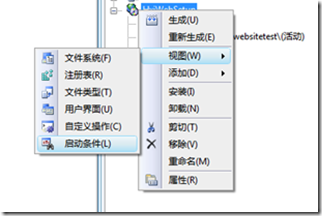
[](http://images.cnblogs.com/cnblogs_com/huxj/WindowsLiveWriter/d24a94621d11.Net_C7AE/image_22.png)

这些选择打包程序需要包含的组件，主要是我们需要.Net Framework组件，我们需要打包进来。

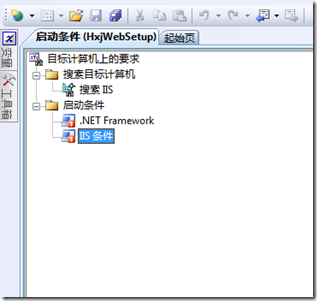
这里将.Net Framework 2.0（X86）打钩，并且在下面指定系统必备组件的安装位置选择第二项，这样就打钩的组件就会跟进安装包了。

第一项则是组件从微软网站上下载，第三项则是指定自定义位置。

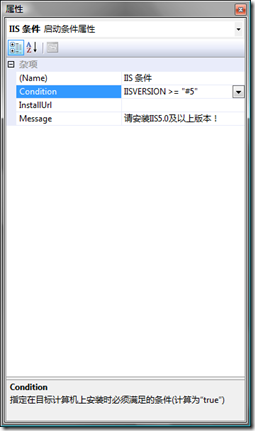
点击确定，接下来，我们右击项目，如下图：

[](http://images.cnblogs.com/cnblogs_com/huxj/WindowsLiveWriter/d24a94621d11.Net_C7AE/image_16.png)

点击启动条件，如下图：

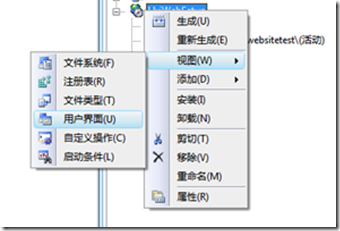
[](http://images.cnblogs.com/cnblogs_com/huxj/WindowsLiveWriter/d24a94621d11.Net_C7AE/image_18.png)

我们可以看到启动条件，我们来看看IIS条件，右击IIS条件，查看属性窗口，如下图：

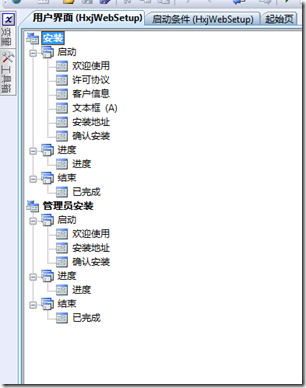
[](http://images.cnblogs.com/cnblogs_com/huxj/WindowsLiveWriter/d24a94621d11.Net_C7AE/image_26.png)

这里可以看到设置条件，IISVERSION >= "#5"表示IIS版本需要5.0以上，如果需要6.0以上则是IISVERSION >= "#6"。

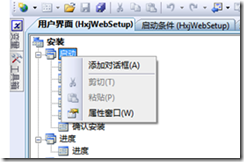
下面我们来设置安装界面。右击项目如下图：

[](http://images.cnblogs.com/cnblogs_com/huxj/WindowsLiveWriter/d24a94621d11.Net_C7AE/image_30.png)

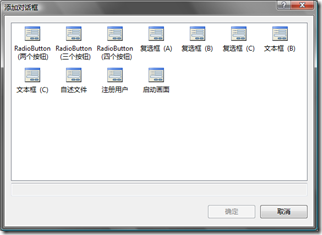
选择点击用户界面，如下图：

[](http://images.cnblogs.com/cnblogs_com/huxj/WindowsLiveWriter/d24a94621d11.Net_C7AE/image_32.png)

这里我已经添加了三个步骤，分别是许可协议、客户信息、文本框(A)这三个。

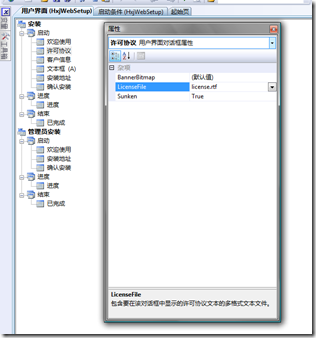
[](http://images.cnblogs.com/cnblogs_com/huxj/WindowsLiveWriter/d24a94621d11.Net_C7AE/image_36.png)

通过右击启动，点击添加对话框，如下图：

[](http://images.cnblogs.com/cnblogs_com/huxj/WindowsLiveWriter/d24a94621d11.Net_C7AE/image_38.png)

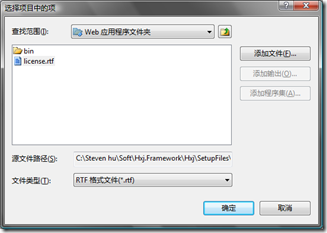
来选择我们需要的步骤。

右击许可协议步骤，查看属性如下图:

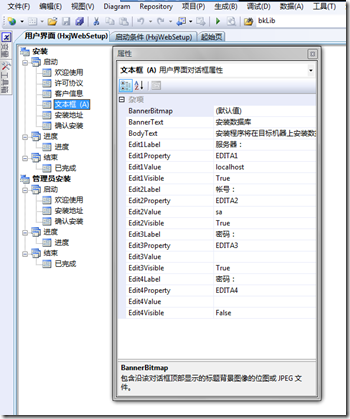
[](http://images.cnblogs.com/cnblogs_com/huxj/WindowsLiveWriter/d24a94621d11.Net_C7AE/image_40.png)

我们知道在安装很多软件的时候都会有一步是许可协议，然后让我们点击同意，然后再下一步，就是这里啦。

我们这里添加了一个license.rtf文件到安装项目，这里就可以选择这个文件了。

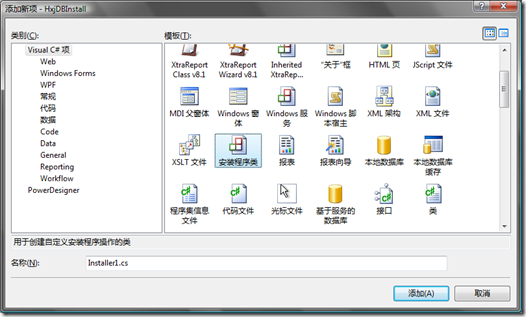
[](http://images.cnblogs.com/cnblogs_com/huxj/WindowsLiveWriter/d24a94621d11.Net_C7AE/image_42.png)

由于在安装过程也需要设置数据库，所以我们还需要让安装用户在安装过程中输入数据库服务器信息，这里就是文本框(A)这个步骤啦。右击步骤查看属性窗口如下图：

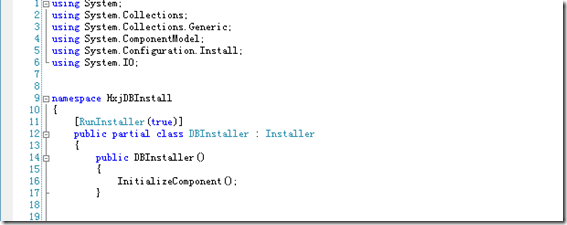
[](http://images.cnblogs.com/cnblogs_com/huxj/WindowsLiveWriter/d24a94621d11.Net_C7AE/image_44.png)

这里有四个文本框可以使用，我们只需要三个数据库服务器、帐号、密码，所以第四个Visible设置为false。

由于需要配置数据库，则我们需要再建一个项目，新建一个类库项目，然后添加一个安装类，如下图：

[](http://images.cnblogs.com/cnblogs_com/huxj/WindowsLiveWriter/d24a94621d11.Net_C7AE/image_46.png)

生成一个安装类如下图，是一个继承Installer类的类。

[](http://images.cnblogs.com/cnblogs_com/huxj/WindowsLiveWriter/d24a94621d11.Net_C7AE/image_48.png)

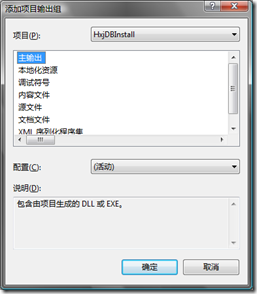
然后我们需要在安装类中编写代码啦。

我们重写Install方法：

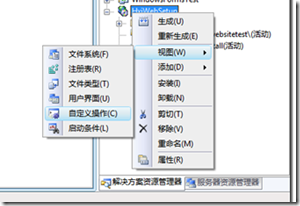
public override void Install(IDictionary stateSaver)

这里就是我们需要写配置数据库信息。

然后我们在安装项目中添加这个项目，如下图：

[](http://images.cnblogs.com/cnblogs_com/huxj/WindowsLiveWriter/d24a94621d11.Net_C7AE/image_50.png)

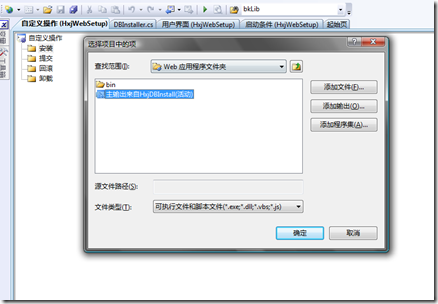
然后右击安装项目，如下图：

[](http://images.cnblogs.com/cnblogs_com/huxj/WindowsLiveWriter/d24a94621d11.Net_C7AE/image_54.png)

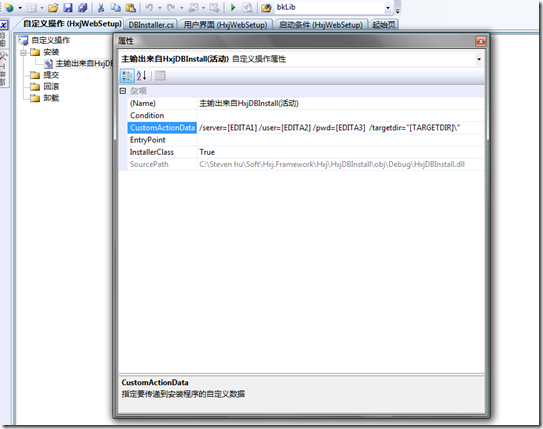
选择自定义操作，则出现自定义操作界面如下图：

[](http://images.cnblogs.com/cnblogs_com/huxj/WindowsLiveWriter/d24a94621d11.Net_C7AE/image_56.png)

然后我们在安装的操作中添加刚才的新建的项目，右击安装，添加自定义操作，如下图:

[](http://images.cnblogs.com/cnblogs_com/huxj/WindowsLiveWriter/d24a94621d11.Net_C7AE/image_58.png)

选中并点击确定。接下来我们需要在安装过程传递输入的数据传递这个项目中，右击安装下面的刚添加的主输出，如下图：

[](http://images.cnblogs.com/cnblogs_com/huxj/WindowsLiveWriter/d24a94621d11.Net_C7AE/image_60.png)

属性框中的CustomActionData就是指定要传递到安装程序的自定义数据。

然后我们再回到新建的那个安装类，在Install方法就可以接收安装过程中输出的数据库信息，如下：

public override void Install(IDictionary stateSaver)

{

base.Install(stateSaver);

string databaseServer = Context.Parameters["server"].ToString();

string userName = Context.Parameters["user"].ToString();

string userPass = Context.Parameters["pwd"].ToString();

string targetdir = Context.Parameters["targetdir"].ToString();

//这里操作添加数据库，只要执行创建数据库的脚本就可以了。

//这个是测试在安装目录下添加接收到的用户填写的数据库信息

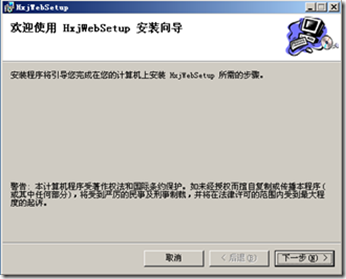
File.WriteAllText(Path.Combine(targetdir, "log.txt"), databaseServer + "/n/r" + userName + "/n/r" + userPass);

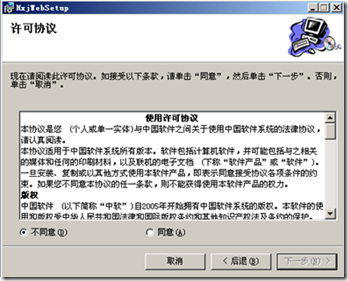
}

这里Context.Parameters["server"]的Server和上面设置CustomActionData是对应的。

这样就完成了安装包的制作了，编译生成一下，我们测试一下安装包是否真的有效。

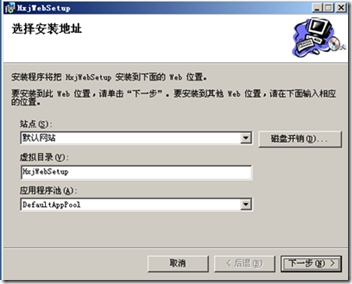
下面接一下制作完安装过程：

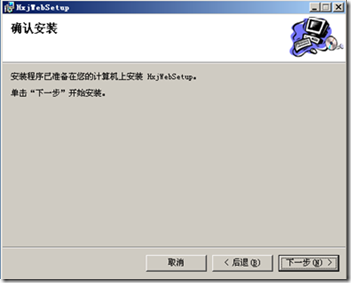
[](http://images.cnblogs.com/cnblogs_com/huxj/WindowsLiveWriter/d24a94621d11.Net_C7AE/image_66.png)

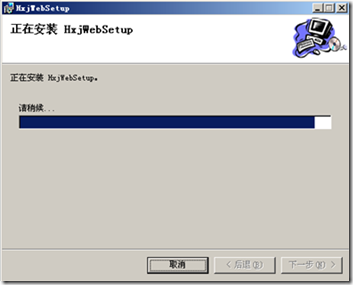
[](http://images.cnblogs.com/cnblogs_com/huxj/WindowsLiveWriter/d24a94621d11.Net_C7AE/image_70.png)

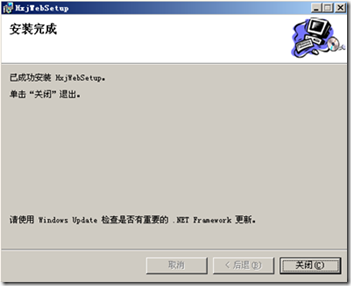
[](http://images.cnblogs.com/cnblogs_com/huxj/WindowsLiveWriter/d24a94621d11.Net_C7AE/image_74.png)

[](http://images.cnblogs.com/cnblogs_com/huxj/WindowsLiveWriter/d24a94621d11.Net_C7AE/image_78.png)

[](http://images.cnblogs.com/cnblogs_com/huxj/WindowsLiveWriter/d24a94621d11.Net_C7AE/image_82.png)

[](http://images.cnblogs.com/cnblogs_com/huxj/WindowsLiveWriter/d24a94621d11.Net_C7AE/image_86.png)

[](http://images.cnblogs.com/cnblogs_com/huxj/WindowsLiveWriter/d24a94621d11.Net_C7AE/image_90.png)

[](http://images.cnblogs.com/cnblogs_com/huxj/WindowsLiveWriter/d24a94621d11.Net_C7AE/image_94.png)

安装过程就这么多了。

下面我们来看看是否已经安装成功，如下图：

[](http://images.cnblogs.com/cnblogs_com/huxj/WindowsLiveWriter/d24a94621d11.Net_C7AE/image_98.png)

说明Web网站已经成功被安装到虚拟目录下了。

# 如何在安装程序中包含.Net Framework4.5 Component.

# How to: Include Prerequisites with a ClickOnce Application

**Visual Studio 2012**

[Other Versions](javascript:;)

http://i3.msdn.microsoft.com/Areas/Epx/Content/Images/ImageSprite.png

13 out of 29 rated this helpful - [Rate this topic](http://msdn.microsoft.com/en-us/library/hh873130%28v=vs.110%29.aspx#feedback)

Before you can distribute prerequisite software with a ClickOnce application, you must first download the installer packages for those prerequisites to your development computer. When you publish an application and choose Download prerequisites from the same location as my application, an error will occur if the installer packages aren’t in the Packages folder.

### To add an installer package by using Package.xml

1. In File Explorer, open the Packages folder.

By default, the path is C:\Program Files\Microsoft SDKs\Windows\v8.0a\Bootstrapper\Packages on a 32-bit system and C:\Program Files (x86)\ Microsoft SDKs\Windows\v8.0a\Bootstrapper\Packages on a 64-bit system.

1. Open the folder for the prerequisite that you want to add, and then open the language folder for your installed version of Visual Studio (for example, en for English).
2. In Notepad, open the Package.xml file.
3. Locate the Name element that contains http://go.microsoft.com/fwlink, and copy the URL. Include the LinkID portion.

|  |
| --- |
| **NoteNote** |
| If no Name element contains http://go.microsoft.com/fwlink, open the Product.xml file in the root folder for the prerequisite and locate the fwlink string. |
| **Important noteImportant** |
| Some prerequisites have multiple installer packages (for example, for 32-bit or 64-bit systems). If multiple Name elements contain fwlink, you must repeat the remaining steps for each of them. |

1. Paste the URL into the address bar of your browser, and then, when you are prompted to run or save, choose Save.

This step downloads the installer file to your computer.

1. Copy the file to the root folder for the prerequisite.

For example, for the Windows Installer 4.5 prerequisite, copy the file to the \Packages\WindowsInstaller4\_5 folder.

You can now distribute the installer package with your application.

### To add an installer package for the .NET Framework 4.5

1. In File Explorer, open the Packages folder.

By default, the path is C:\Program Files\Microsoft SDKs\Windows\v8.0a\Bootstrapper\Packages on a 32-bit system and C:\Program Files (x86)\ Microsoft SDKs\Windows\v8.0a\Bootstrapper\Packages on a 64-bit system.

1. Open the DotNetFX45 folder.
2. From the [Download Center](http://go.microsoft.com/fwlink/?LinkId=225702), download the package file dotNetFx45\_Full\_x86\_x64.exe, and then copy it to the DotNetFX45 folder.
3. For localized applications (any locale other than EN), open the language folder for the application’s locale.
4. In Notepad, open the Package.xml file.
5. Locate the Name element that contains “DotNetFX45FullLanguagePackBootstrapper”, and copy the URL, including the LinkID portion.
6. Paste the URL into the address bar of your browser.
7. When you're prompted to run or save, choose the Save button.

This step downloads the installer file to your computer.

1. Copy the file to the DotNetFX45 folder.

You can now distribute the installer package with your application.

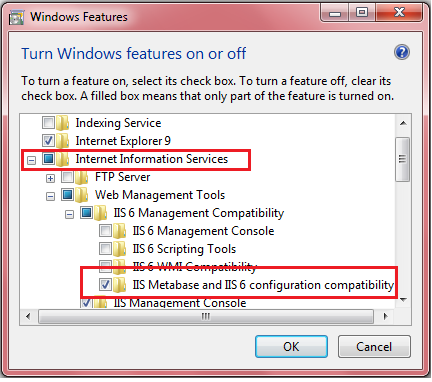
# 常见错误解决方案：

# 

# Step 1: go to control panel -> programs -> Turn Windows Features On or Off

# http://3.bp.blogspot.com/-Sl4Rt0GHFAs/T4W3RiijtzI/AAAAAAAABC4/RrM4I0Q1Lxs/s1600/TurnWindowsFeatures.png

# Step 2: Select IIS 6 Management Compatibility -> IIS Metabase and IIS 6 configuration compatibility



# Step 2: Windows 2008 / Windows 2012 Server

# Go to "Add Role Service"

# Management Tools -> IIS 6 Metabase compatibility

# 

创建自定义窗口

<http://www.codeproject.com/Articles/568476/Creating-an-MSI-Package-for-Csharp-Windows-Applica>

<http://www.codeproject.com/Articles/181395/How-to-create-custom-dialog-boxes-using-the-Web-Se>

**案例一**

## Introduction

There are number of ways provided by Microsoft to create a setup project for windows application.

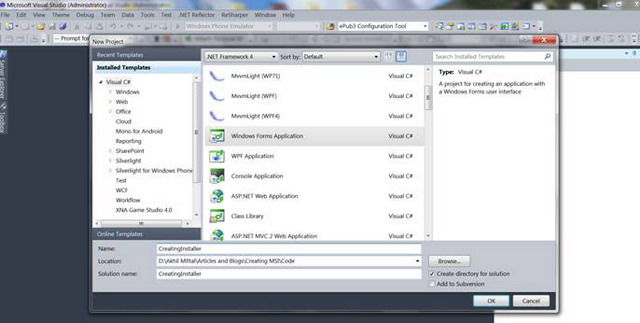
But when I started to create one, I got nothing but queries and confusions of how to start and where to start. There are numerous articles I found explaining to create a setup project, but some did not work, and some did not have a live example to follow.

The driving force for me to write this article is my QC team, who accept the main application for testing, and who also verified my setup installer with their 100% effort. And guess what, they successfully found bugs in that too.

In this article I would like to explain a step by step process to create a windows application and a setup installer for the same in a very simple manner, that is easy to understand and follow knowing that there are a number of other ways to do the same thing.

## Start the Show

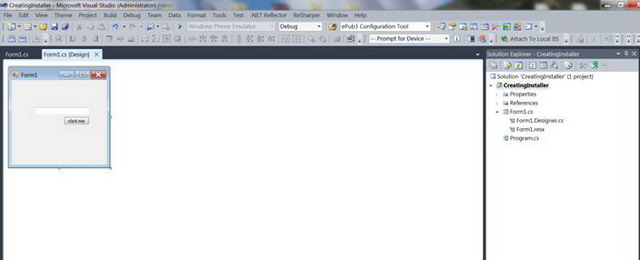
First, let’s create a simple one form windows application, with only a text box and a button.



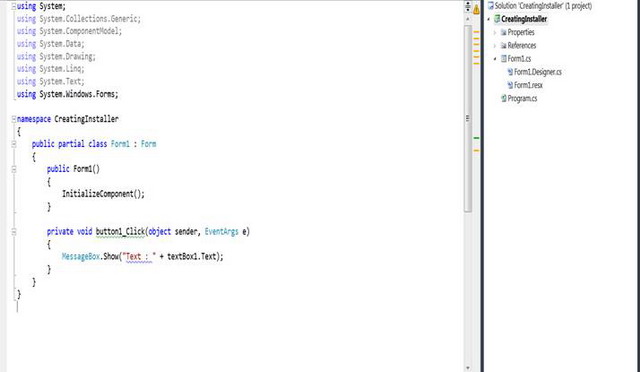
Creating a windows application is just for the sake of having one to install.

I gave the name CreatingInstaller to my windows application, obviously you can choose your own.

Adding a new Windows Form Application in my solution and adding a text box and button to the default form resulted in the figure as shown below. Decorate the control properties however you want.

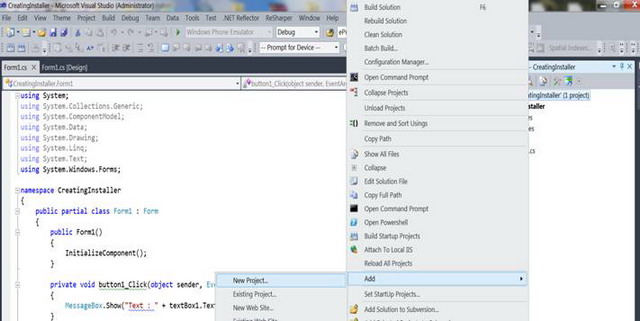


Just wanted to write few lines of code, so I binded the button’s click event to show text box's text

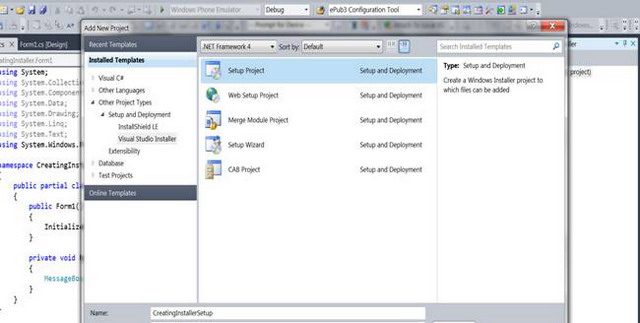


## Primary Objective

So far so good. Now let’s create an installer for the same windows application. Right click on the solution and add a new project to your solution like in following figure:

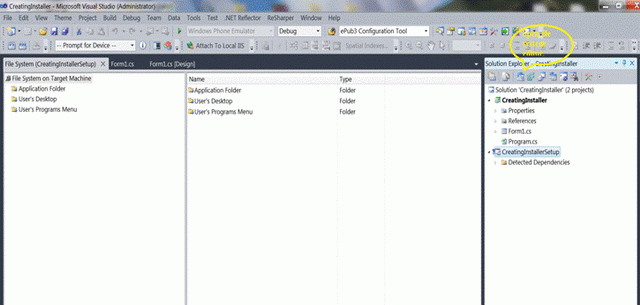


And add a setup project by **Other project Types->Setup and Deployment->Visual Studio Installer**

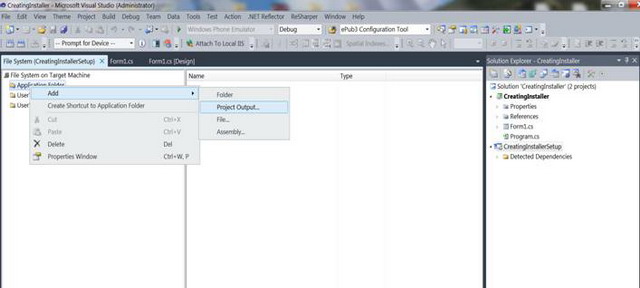


The project will be added to the solution. Now open the file system editor by clicking on the project and select the option to open file system editor, like in below figure:

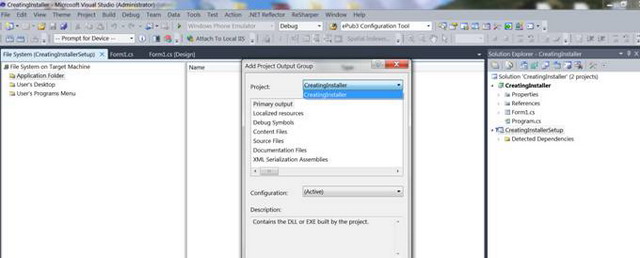
You'll get to see Application Folder, User’s Desktop and User’s Program Menu.



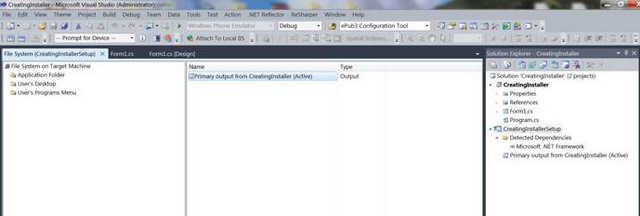
Right click on Application Folder and add an output project. Out project specifies the project we are creating an installer to, like in the following figure:



Select CreatingInstaller (i.e. the windows application project name) in the add output project window and select it as a primary output as shown below and click OK.



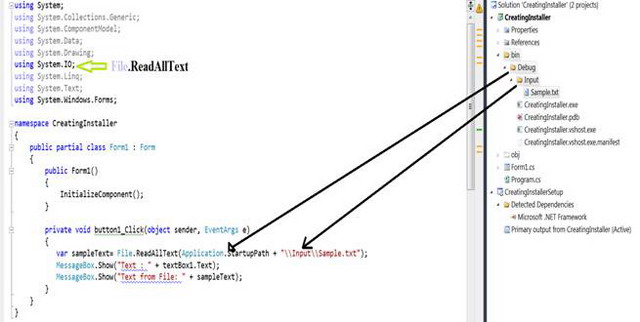
The Primary output will be added as shown below, having type defined as Output.



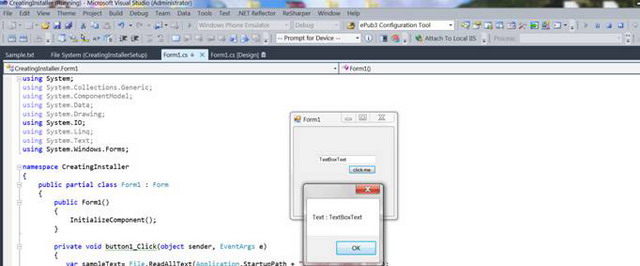
In the meanwhile, let's add some more functionality to our windows application. Let's read a file and show its output in a message box upon a button click. Therefore, just add a text file. I called it Sample.txt to the bin\debug\Input folder, input is the custom folder I created to place my txt file.

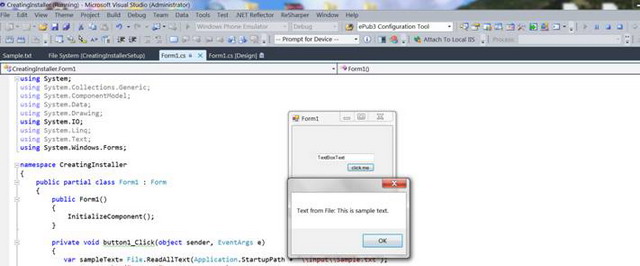
Write a few lines of code just to read the txt file from the Startup path. In my case bin\debug, it could also be bin\release as per the project build, and specify the file folder name and file name to read the content. I chose to keep my txt file at the startup path so that I could explain how we can create files and folders at the time of installation. Now we also need this Input folder and a Sample.txt file at the time of installation to be located at the location of installed application.

For file operations I added the namespace System.IO though it is unnecessary to do so.



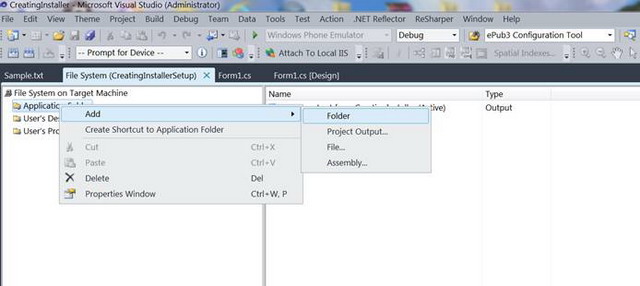
Therefore, running the application will show two message boxes, one after the other showing text box text and text from Sample.txt file.



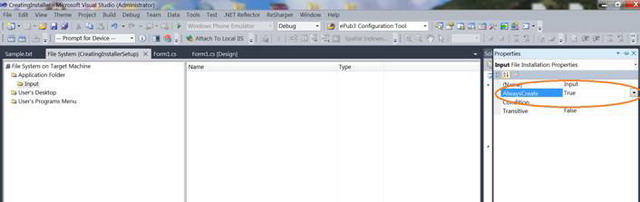


Now this folder creation logic has to be implemented in our setup project, so that when the application installs, it has all the pre-requisites required to run the application, like the Input folder and the Sample.txt file.

So, right click on Application Folder in File system editor and add a folder. The folder will be created just below the Application Folder, name that folder **Input**.



Right-click on folder, select properties, and mark the Always Create property to True. That means the folder will always be created whenever we run the installer, after a fresh build release.

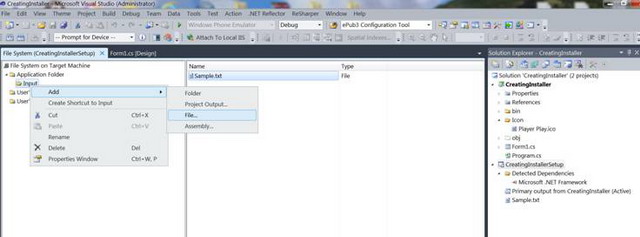


## Create Shortcuts

You can decorate your form to add an icon to it, and that icon will also be required at the time of installation to create a shotcut icon to our application. Add an icon to the form like in below mentioned figure:

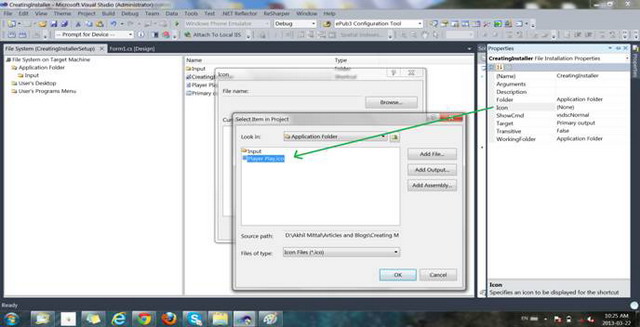


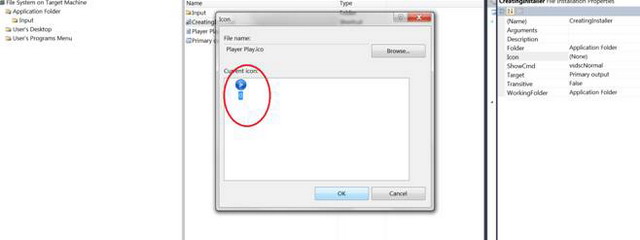
Time to add the Sample.txt file. Right click the Input folder created and Add file to it, browse for the Sample.txt file in the Windows Application project we created earlier.



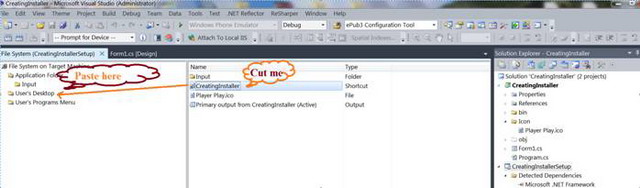
To create a shortcut to the application, right click on Primary output in middle window pane and select Create shortcut to Primary output, name that shortcut as CreatingInstaller.

Select the properties of the shortcut, by right clicking it and add an icon to it. This icon will be created on the desktop when the application launches. The below figures explain how to add an icon.

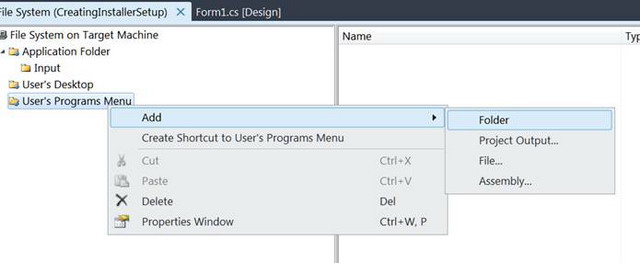




Cut the shortcut created at Application Folder and Paste it under User’s Desktop Folder.

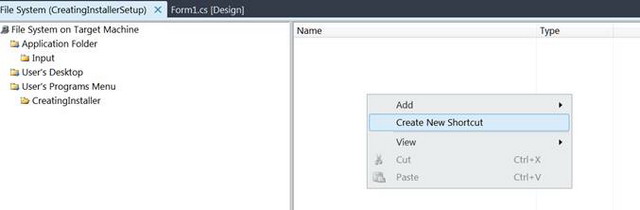


For shortcuts to be created at the User’s Program Menu, add a new folder to the User’s Program Menu. This will be created at the program’s menu location in that folder. Create a new shortcut pointing to the primary output as we did when we created a desktop shortcut. The three images below describe the process:



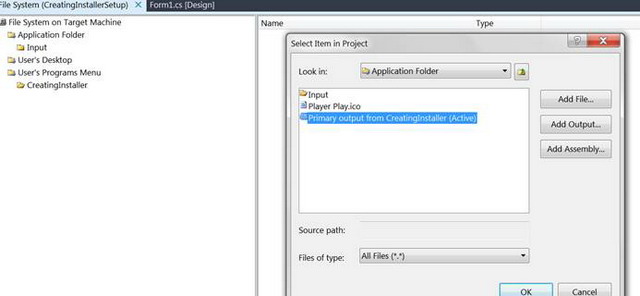
Name the folder CreatingInstaller.

Right click on middle window pane to create a new shortcut.

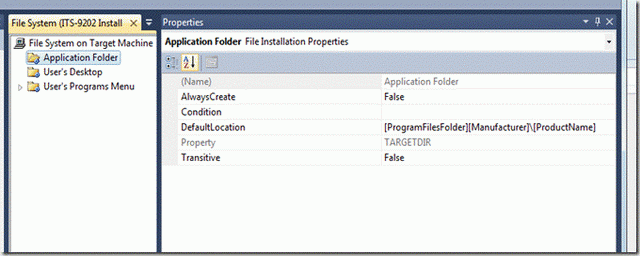


Select shortcut source to primary output selected.

Also add icon to shortcut, as done for Desktop shortcut.



Right click Application folder to set the properties of where to install the application.

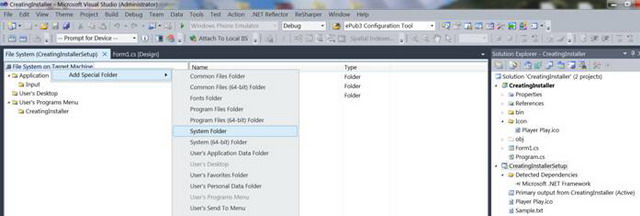


## Uninstall

We always have an option to uninstall the application from the control panel’s Programs and Features list, but how about creating our own uninstaller? That is also under the programs menu so we do not have to disturb the control panel.

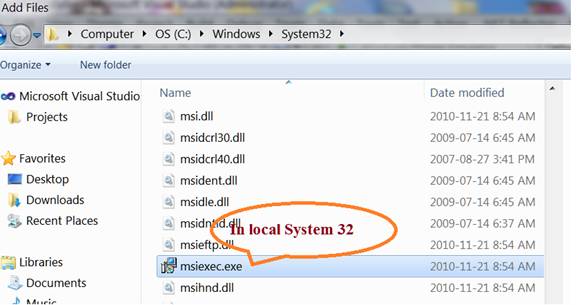
### Step 1

Right click on File System on target Machine and Add Special Folder->System Folder as shown in below figure.

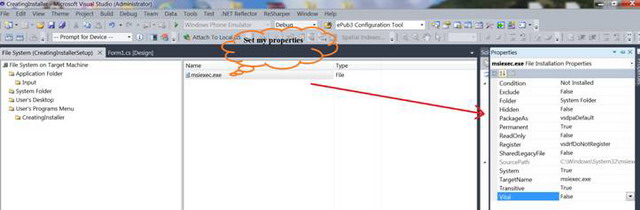


### Step 2

Right click on the newly created system folder and browse for the msiexec.exe file in the local System.Windows32 folder. This file takes responsibility to install and uninstall the application based on certain parameters specified.

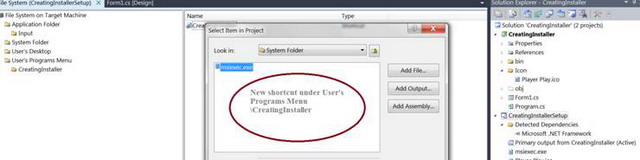


Set the properties of the file exactly as shown in the figure:



### Step 3

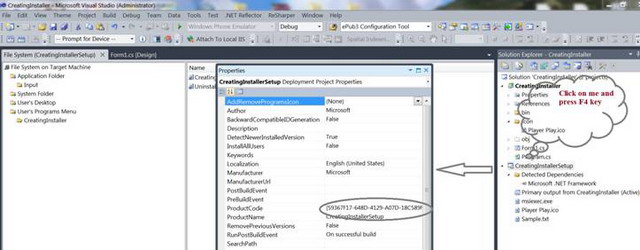
Now create a new shortcut under the User’s program Menu and point its source to msiexec as shown below. You can add more icons and a name to your shortcut. I have given it the name "Uninstall."



### Step 4

Press F4 key by selecting the setup project. We see a list of properties, which we can customize as per out installation needs, like Product name, Author, Installation location. I’ll not go into a deep discussion about all of this, as they are quite easy to understand and set.

Just take a note of the product code shown below in the list of properties. We would need product code as a parameter to msiexec for uninstallation.



### Step 5

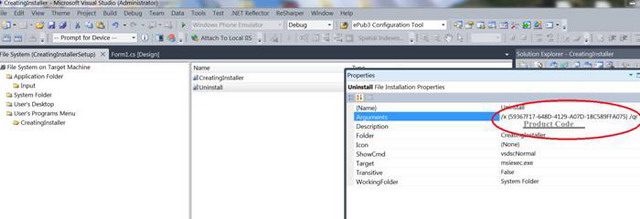
Right click the Uninstall shortcut and set the arguments property as shown in below figure:

http://www.codeproject.com/images/minus.gifCollapse | [Copy Code](http://www.codeproject.com/Articles/568476/Creating-an-MSI-Package-for-Csharp-Windows-Applica)

/x {product code} /qr

/x is for uninstalltion.

You can get the whole detailed list of parameters and their use at [http://technet.microsoft.com/en-us/library/cc759262(v=ws.10).aspx](http://technet.microsoft.com/en-us/library/cc759262%28v=ws.10%29.aspx). Chose whichever one you like.



### Step 6

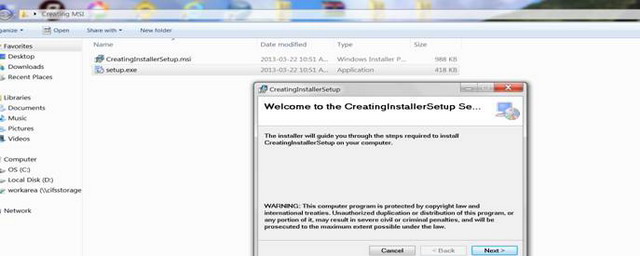
Save all and Rebuild the setup project.

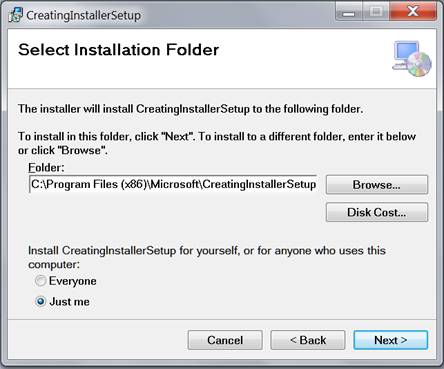
## Job Done!

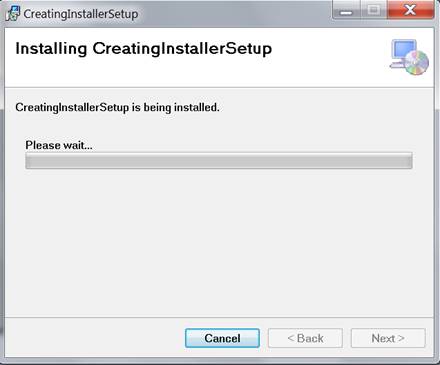
Now our setup is ready to install our windows application.

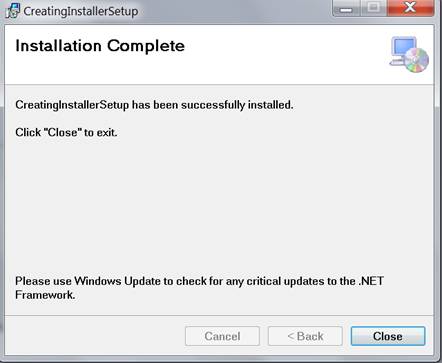
Just browse the debug folder location of Setup project. We find an msi and a setup.exe. You can run either to initiate setup.

When we started we saw a setup wizard with screens that welcomed the user, asked for the location to install (while the default location was already set.)



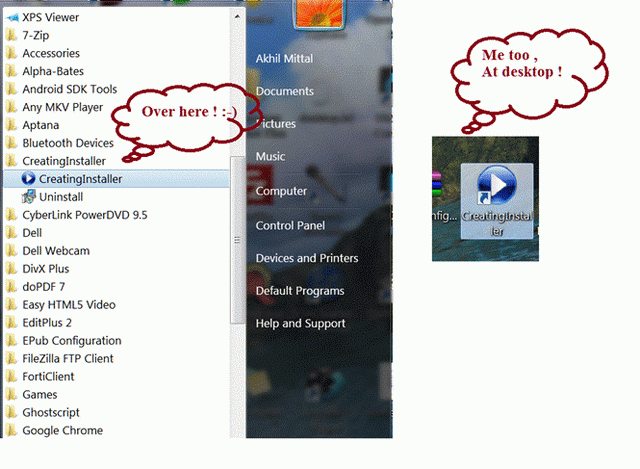






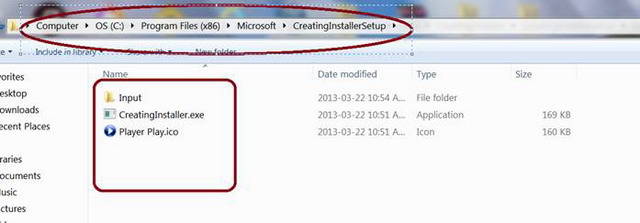
After completing the wizard, click the close button.

Now that the job is done we can see our shortcuts to the application created at desktop and User’s Program Menu like in below given figure.

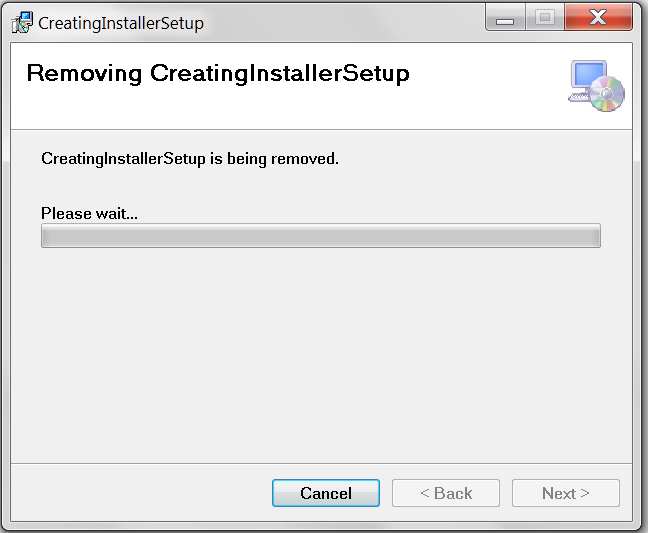


Now if we navigate out to the installation location we can also see the Input folder created and the Sample.txt file resting inside it.

Run the application and see the output.



Click on uninstall to remove the application. The wizard launches as shown below:

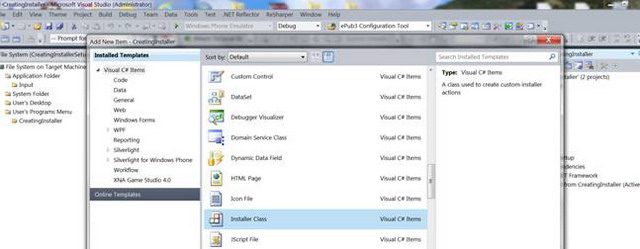


## Custom Actions

Just wanted to give a glimpse of Custom Actions we can define, while creating the setup.

Custom actions are the actions which contain customized functionality apart from default one at the time of installation and uninstallation. For example, my QC team reported a bug that when running the application while simultaneously in background uninstalling the application, the application still keep on running. As per them it should show a message or close during the uninstallation. It was hard to explain to them the reason for this, so I opted for implementing their desire in the setup project.

1. Just add an installer class to the windows application we created earlier. When we open the installer class we can see the events specified for each custom action i.e. for Installation, Uninstallation, Rollback, Commit.



My need was to write code for the uninstallation, so I wrote few lines to fulfill the need.

The code contains the logic to find the running EXE name at the time of uninstallation. If it matches my application EXE name, it kills the process. Not going into more details to it.Just want to explain the use of custom actions.

http://www.codeproject.com/images/minus.gifCollapse | [Copy Code](http://www.codeproject.com/Articles/568476/Creating-an-MSI-Package-for-Csharp-Windows-Applica)

using System;

using System.Collections;

using System.Collections.Generic;

using System.ComponentModel;

using System.Configuration.Install;

using System.Diagnostics;

using System.Linq;

namespace CreatingInstaller

{

[RunInstaller(true)]

public partial class Installer1 : System.Configuration.Install.Installer

{

public override void Install(IDictionary savedState)

{

base.Install(savedState);

//Add custom code here

}

public override void Rollback(IDictionary savedState)

{

base.Rollback(savedState);

//Add custom code here

}

public override void Commit(IDictionary savedState)

{

base.Commit(savedState);

//Add custom code here

}

public override void Uninstall(IDictionary savedState)

{

Process application = null;

foreach (var process in Process.GetProcesses())

{

if (!process.ProcessName.ToLower().Contains("creatinginstaller")) continue;

application = process;

break;

}

if (application != null && application.Responding)

{

application.Kill();

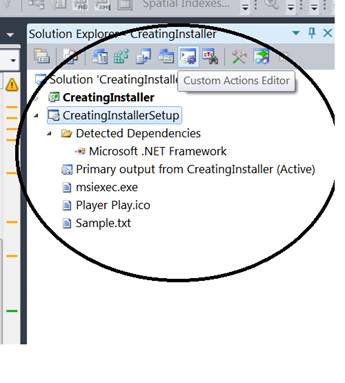
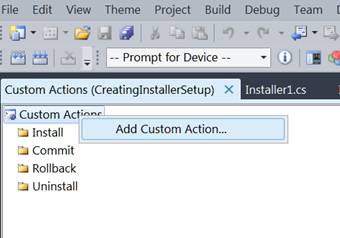
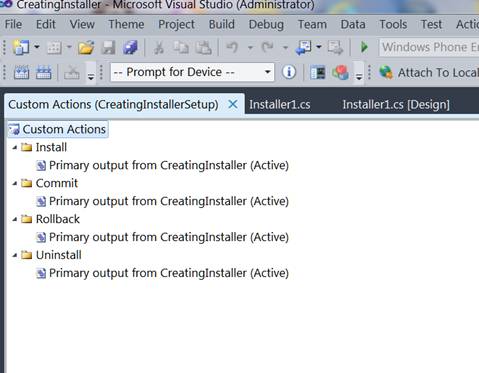
base.Uninstall(savedState);

}

}

}

}

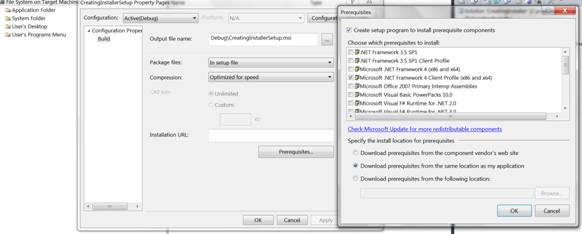
1. Click on the Custom Actions Editor after selecting the **CreatingInstallerSetup** project.  
     
   
2. We see the custom action editor pane on left window. Right click it to add a custom action and select the primary output in the Application Folder.  
     
   
3. We see primary output added as custom actions now. at the time of uninstallation my custom action will be fired and the application will be closed while uninstalling it.  
     
   

## .NET Framework

What if the installation machine does not have a .NET framework? We can specify our own package supplied with installation so that our application does not depend on the .NET framework of the client machine, but points to the package we supplied to it to run.

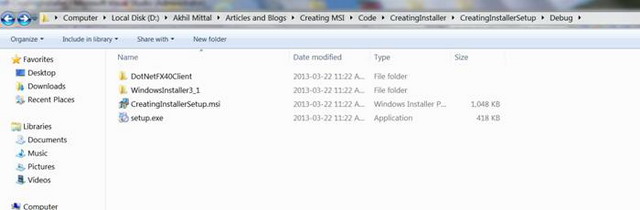
Right click on Setup project, to open properties window.

Here we can specify pre-requisites for the application to install. Just click on Prerequisites button and in the opened prerequisites window, select the checkbox for the .NET Framework application that needs to follow, and select the radio button at number 2 (i.e. Download prerequisites from the same location as my application.) Press OK, but save the project and re-build it.



Now when we browse the Debug folder of the Setup project we see two more folders as a result of the actions we performed just now.

Now this whole package has to be supplied to the client machine for the installation of the application.



Now re-install the application from setup.exe, and launch it using shortcuts.

## Conclusion

The tutorial covers the basic steps for creating the installation project. I did not go very deep explaining the registry or license agreements though. There are many things to be explored to understand and master this topic. However, this article was just a start for a developer to play around with setup and deployments. Happy Coding!

Please visit my blog [**A Practical Approach**](http://csharppulse.blogspot.in/)for more informative articles.

**案例二**

## Introduction

For most software, the development is not considered complete as late as the deployment of the solution, whether it is a website or a desktop solution. There are numerous ways of deploying your software for the end user. For example, when deploying a site, you can either publish the site using Visual Studio's Web Deploy options, or you can create a Web Setup Project (a great article on creating a web setup project is available on Scott Gu’s blog: <http://weblogs.asp.net/scottgu/archive/2007/06/15/tip-trick-creating-packaged-asp-net-setup-programs-with-vs-2005.aspx>), or you can even simply copy and paste the whole website to the server and do all the configuration manually.

If you are using the VS Web Setup Project, there is quite a possibility that you would like to perform some additional tasks which may require you to add some custom logic for your software to start some kind of service or for setting the connection string for your application. For that purpose, you may require some additional input from the user which can be obtained by simply using the dialog boxes provided in the VS Web Setup Projects. These are a set of predefined dialog templates which are setup according to our own requirements. There are a few limitations though, which are listed below:

* You cannot add a combination of controls other than those which are already provided, i.e., you may want to use a check box, two text boxes, and a radio button. In this case, you cannot use the predefined dialogs.
* You have no access whatsoever over the controls in the dialog box; for example, you would want to have a Form where the user checks the “Windows Authentication” option and you would want to hide the username and password fields provided for SQL Server authentication. This cannot be achieved using the predefined dialog boxes.
* You will have to do tedious work in order to get a variable from your dialog to a custom action where you actually want to use the values provided in the form. For example, you will have to add something like below for your custom action data from your user interface, and will use the context in the custom action to get the value, which can become confusing at times.

http://www.codeproject.com/images/minus.gifCollapse | [Copy Code](http://www.codeproject.com/Articles/181395/How-to-create-custom-dialog-boxes-using-the-Web-Se)

/targetdir="[TARGETDIR]\" /targetvdir="[TARGETVDIR]" /targetsite="[TARGETSITE]"

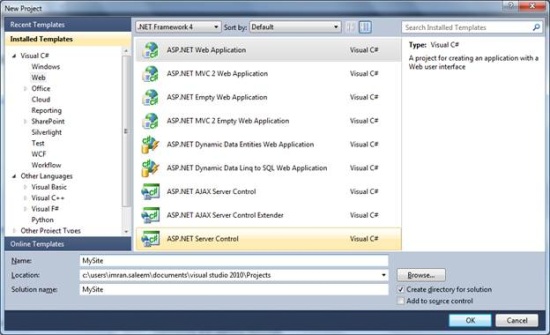
* You cannot add any kind of validation for your input; there is no way to make sure the user has entered an input and that it is valid.

Sometimes you would want to go beyond the provided functionality in the setups and have some control over the flow and execution of operations which you want to perform during the setup. The good news is, you can simply use your self-created Windows Form and get it to execute during the setup process. This is a good way of achieving full control over your custom operations.

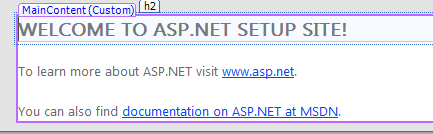
Now I will give a step by step walk through of creating custom dialog boxes and adding them to your web setup projects.

## Creating a Web Site

1. Open your Visual Studio and click File -> New -> Project, and from the installed templates section, select Web, and select the template you want to use for your website. In this case, I am using ASP.NET Web Application.

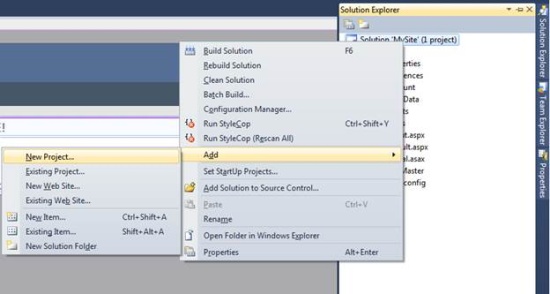


1. In the default page, I change the standard “WELCOME TO ASP.NET!” text to “WELCOME TO ASP.NET” to make sure this is our page.

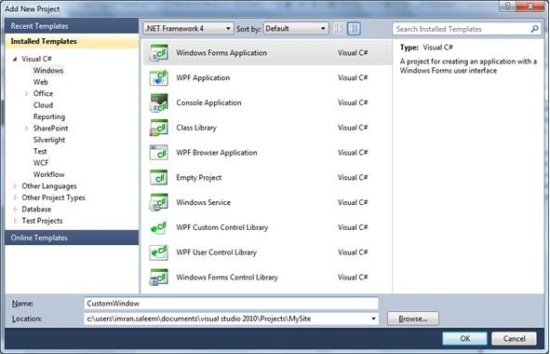


## Creating a Windows Form for our Custom Dialog

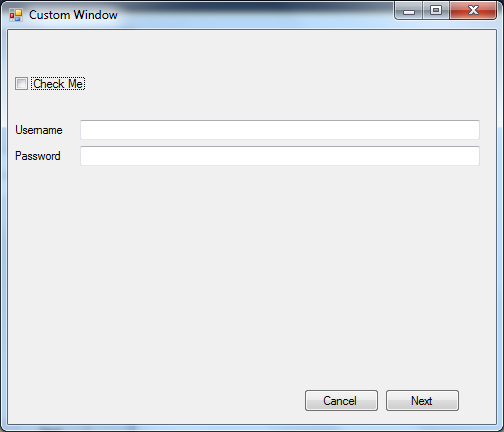
1. Right click the solution in Solution Explorer, and click the “New Project” sub menu item in the Add menu.



1. In “Windows” Installed Templates, choose Windows Forms Application. Enter “Custom Window” in the Name field for the project.



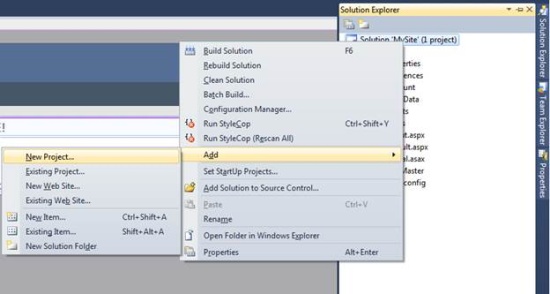
1. A blank Windows Form will be shown in the Designer. Design the window as per your requirements. I have added a few controls like below, but in the actual setup, you would make the screen as close as possible to the setup steps.



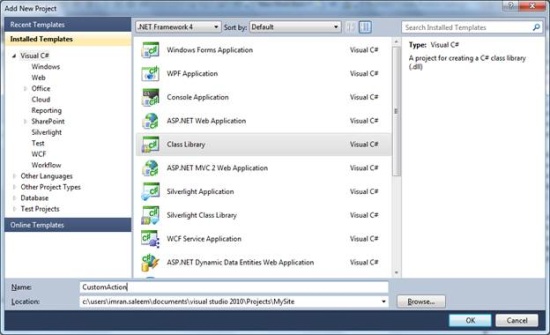
Using this form, you can do whatever you want with full control over all the objects on the form.

## Creating a Custom Action Class Library

1. Right click the solution in Solution Explorer, click the “New Project” sub menu item in the Add menu.



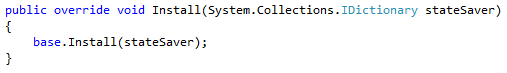
1. Select Class Library from the Visual C# installed templates. This will be our custom action from which we will call our dialog box showDialog() function.



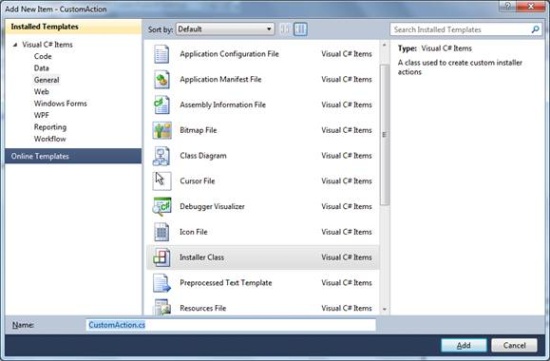
1. Delete the class name Class1.cs.
2. Add System.ComponentModel, System.Windows.Forms, and System.Configuration.Install in namespaces.
3. Add the [RunInstaller(true)] attribute on the class.

http://www.codeproject.com/KB/install/DialogWebSetup/image012.png

1. Add a new class named CustomAction.cs and inherit it from System.Configuration.Install.Installer. You will have to add a reference to the System.Configuration.Install component to your project.
2. You will now override the Install function of the Installer class.



1. You can skip the steps from 9 – 11 by adding the Installer class in the General category in Installed Templates.

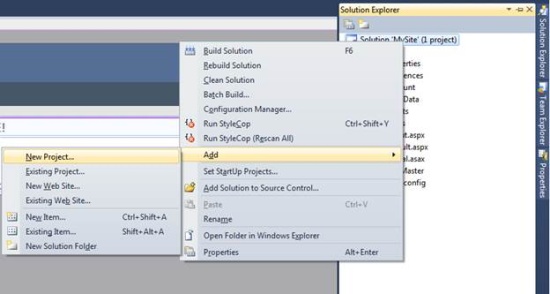


1. Add a reference to the CustomWindow project in the CustomAction project.
2. Add the following lines of code in the override:

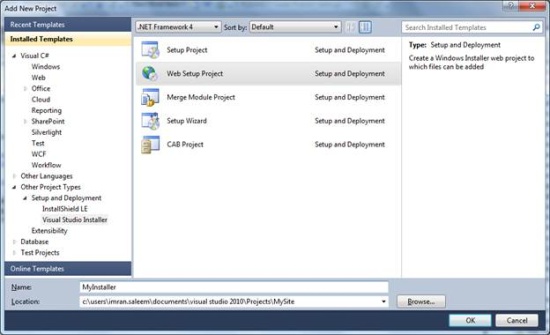
http://www.codeproject.com/KB/install/DialogWebSetup/image014.png

## Creating a Web Setup Project

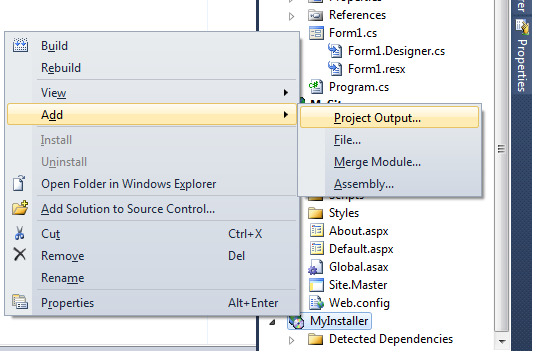
1. Now we will add the Web Setup Project. For that, right click the solution in Solution Explorer, and click the “New Project” sub menu item in the Add menu.



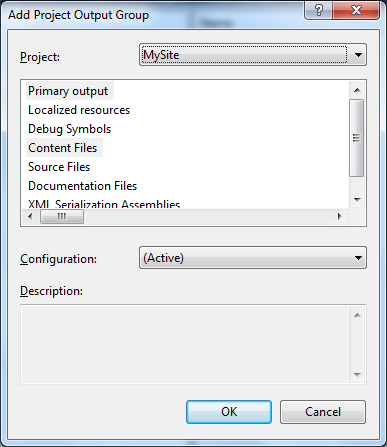
1. Expand the “Other Project Types” node in the Installed Templates pane. Under the “Setup and Deployment” node, click “Visual Studio Installer” and select “Web Setup Project” from the templates pane. Enter “MyInstaller” in the Name field and click OK.



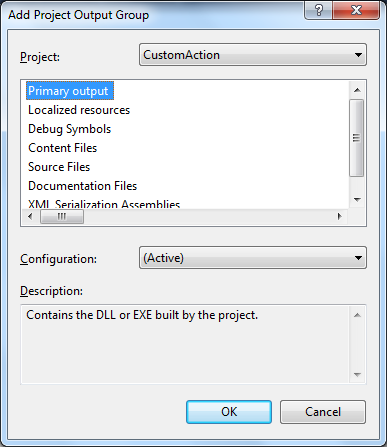
1. Once the project is created, right click the “MyInstaller” project in Solution Explorer, and in the “Add” menu, click “Project Output”.



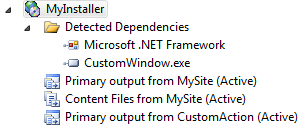
1. The “Add Project Output Group” window will be shown. Select “MySite” from the Project dropdown list and select “Primary output” and “Content Files”, and press OK.



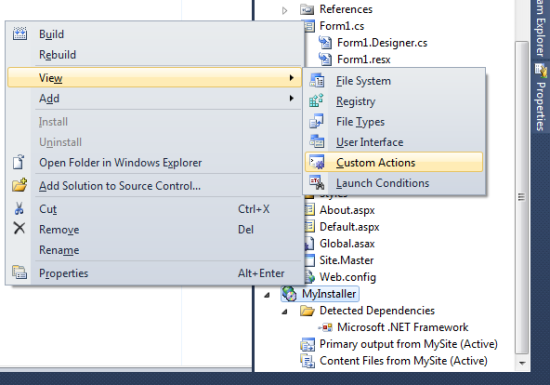
1. Similarly, add the primary output of the CustomAction project in the MyInstaller project.



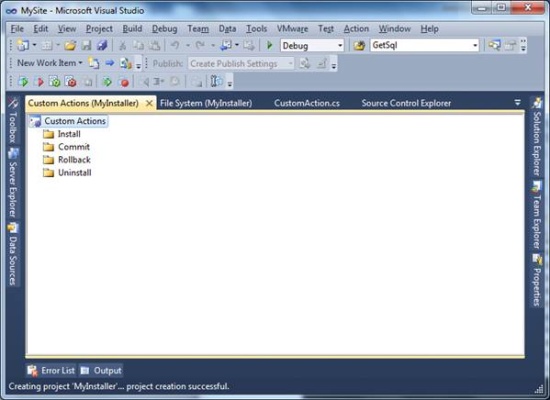
1. The output from the MySite project and CustomAction will be added to the installer. See figure below.



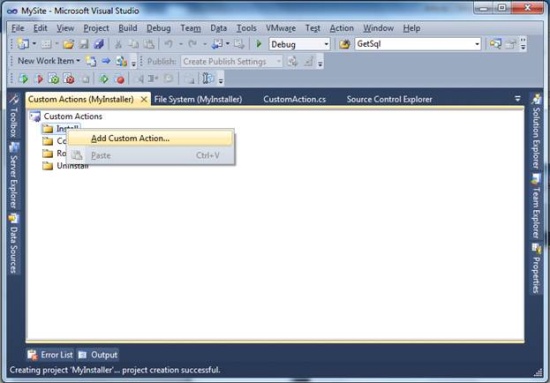
1. To add the custom action to show our dialog box, right click the “MyInstaller” project and in the “View” menu, click “Custom Actions”.



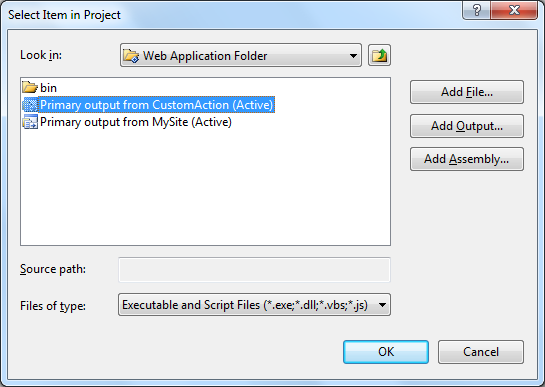
1. The Customer Action tab will be opened:



1. Right click the “Install” folder under CustomActions and click “Add Custom Action…”



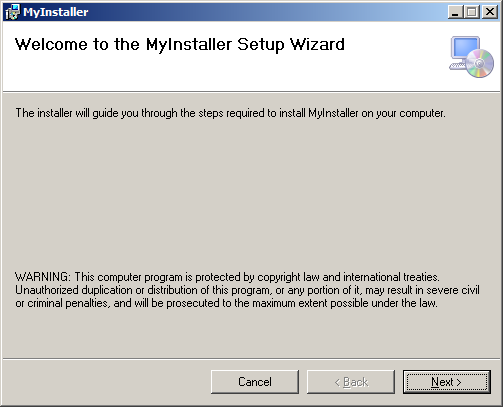
1. A dialog will open and in the Web Application Folder, select “Primary output from CustomAction” and click OK.



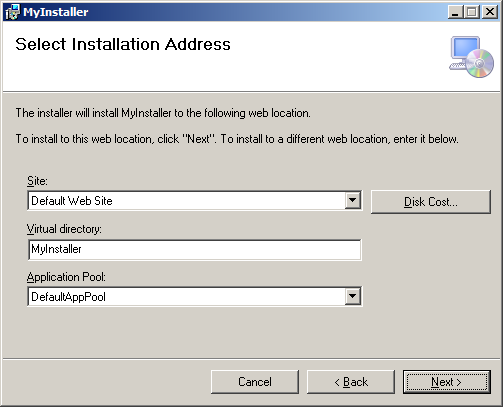
1. And that should be it, phew. Build the solution and our setup will be created in the Debug or Release folder of our project depending on our configuration.

## Running the Setup

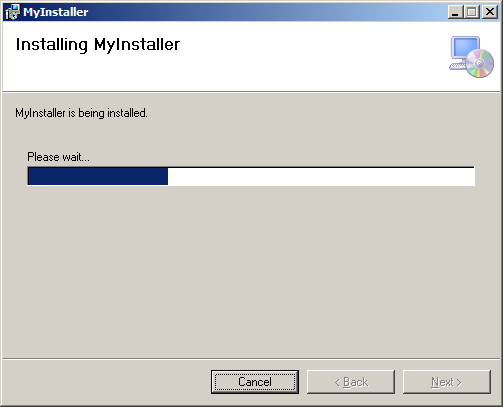
Now we are going to run the setup we just created. Just double click Setup.exe and our setup will start.



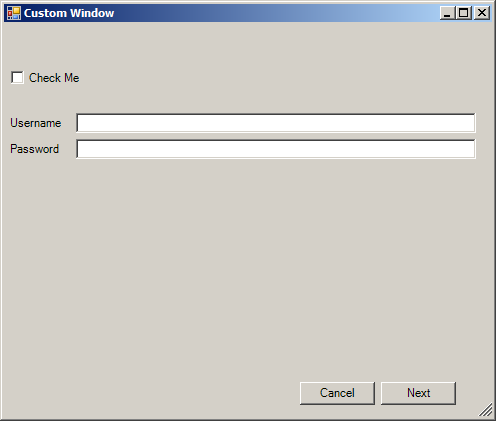
Click Next for the web configuration options:



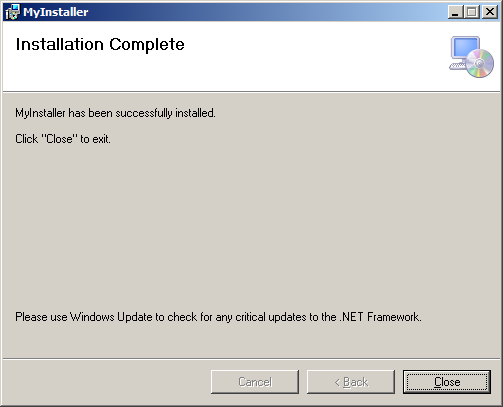
Click Next to confirm installation and click Next to start setup.



You will see the custom action dialog which was added.



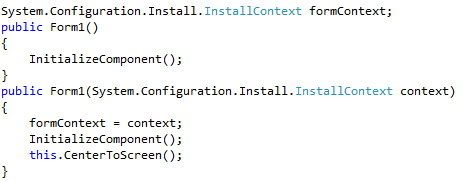
And when you click Next after performing your operation, when the dialog is closed, the control will be given back to the installer and it will complete the installation.



Click Close to complete the installation.

## Passing the Context to a Custom Dialog

In some cases, you would want to pass the context to your custom dialog and use the context variable. You can create a class member of Install and create a constructor for the form class which accepts an InstallContext object and assigns it to our formContext object.



And when creating the instance of our dialog in a custom action, pass it as a single argument for the constructor.

http://www.codeproject.com/KB/install/DialogWebSetup/image033.png

## Conclusion

By using this technique, you can get custom functionality while running your setup with full control over the execution of your custom logic.