Creating an MSI Package for C# Windows Application Using a Visual Studio Setup Project

By **Akhil Mittal**, 27 Aug 2013

Download source - 818.7 KB

Introduction

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

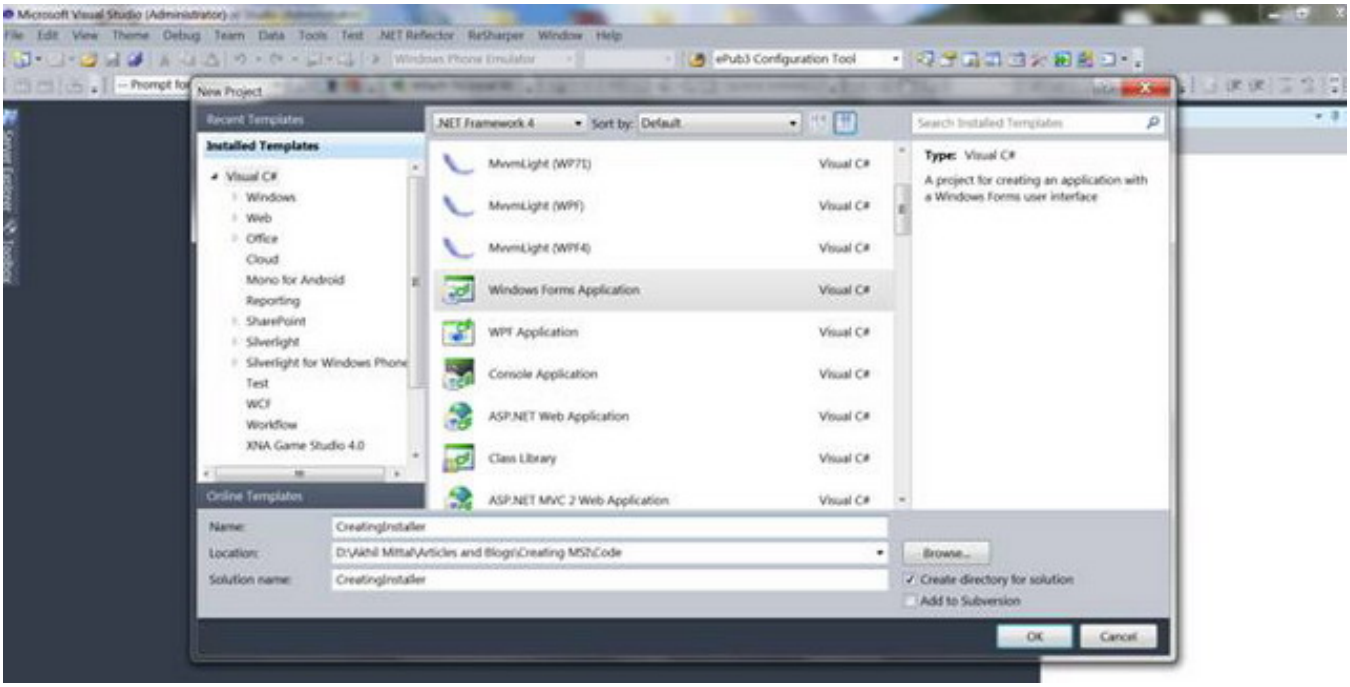
But when I started to create one, I got nothing but queries and confusions of how to start and where to start. There are numerous articles I found explaining to create a setup project, but some did not work, and some did not have a live example to follow.

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

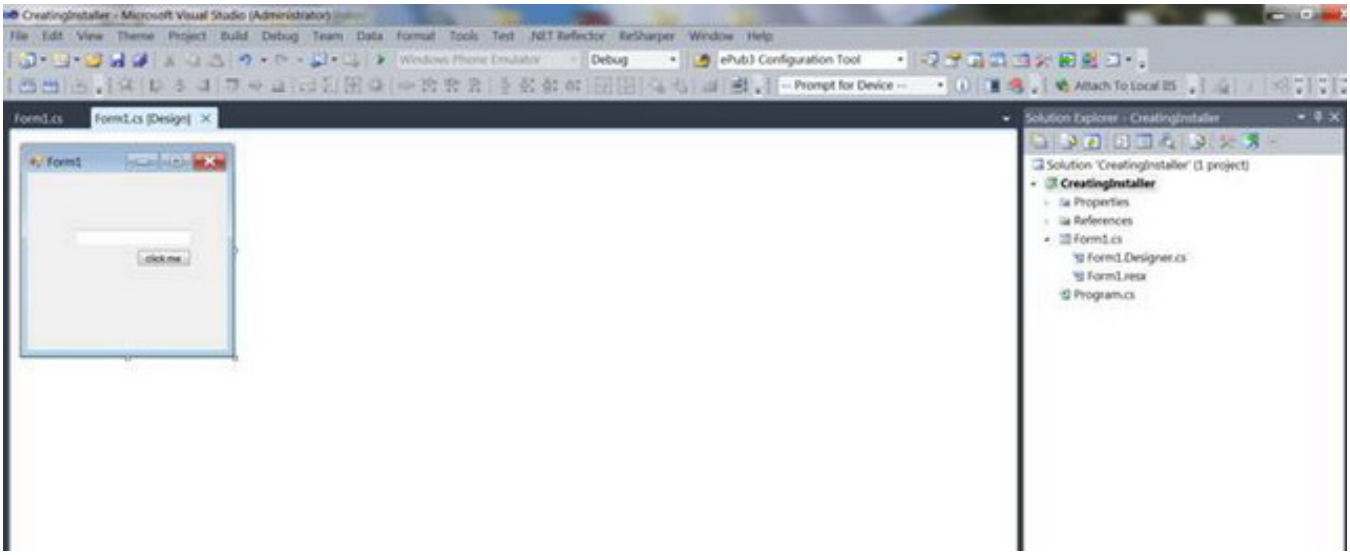
First, let's create a simple one form windows application, with only a text box and a button.



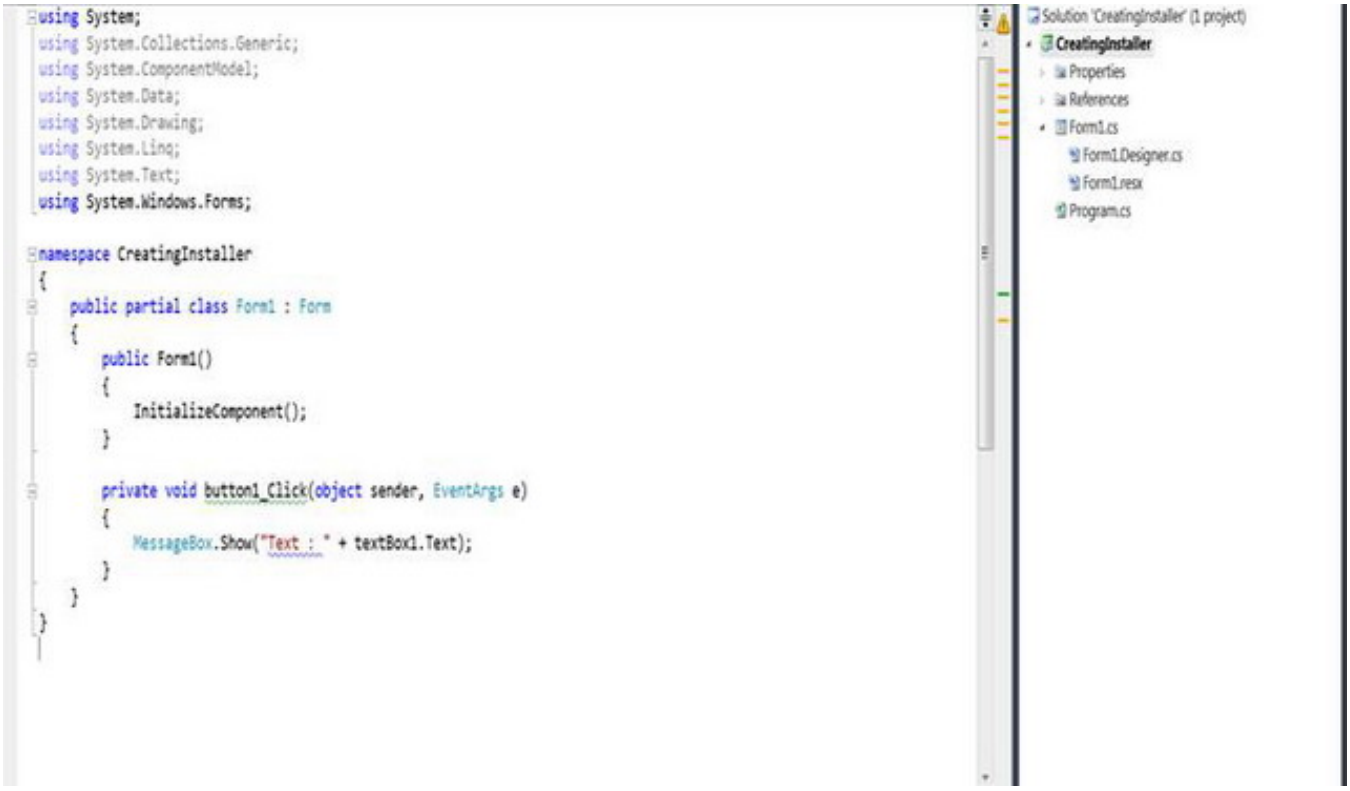
Creating a windows application is just for the sake of having one to install.

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.

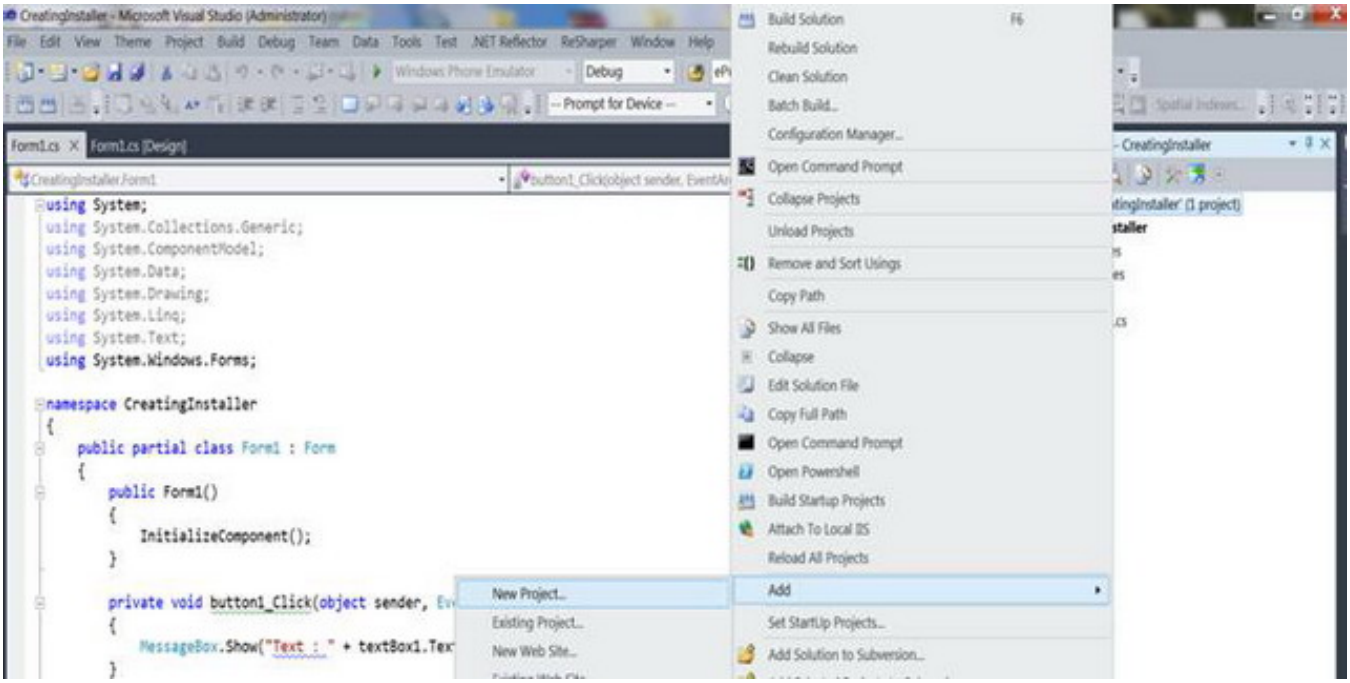


Just wanted to write few lines of code, so I binded the button's click event to show text box's text

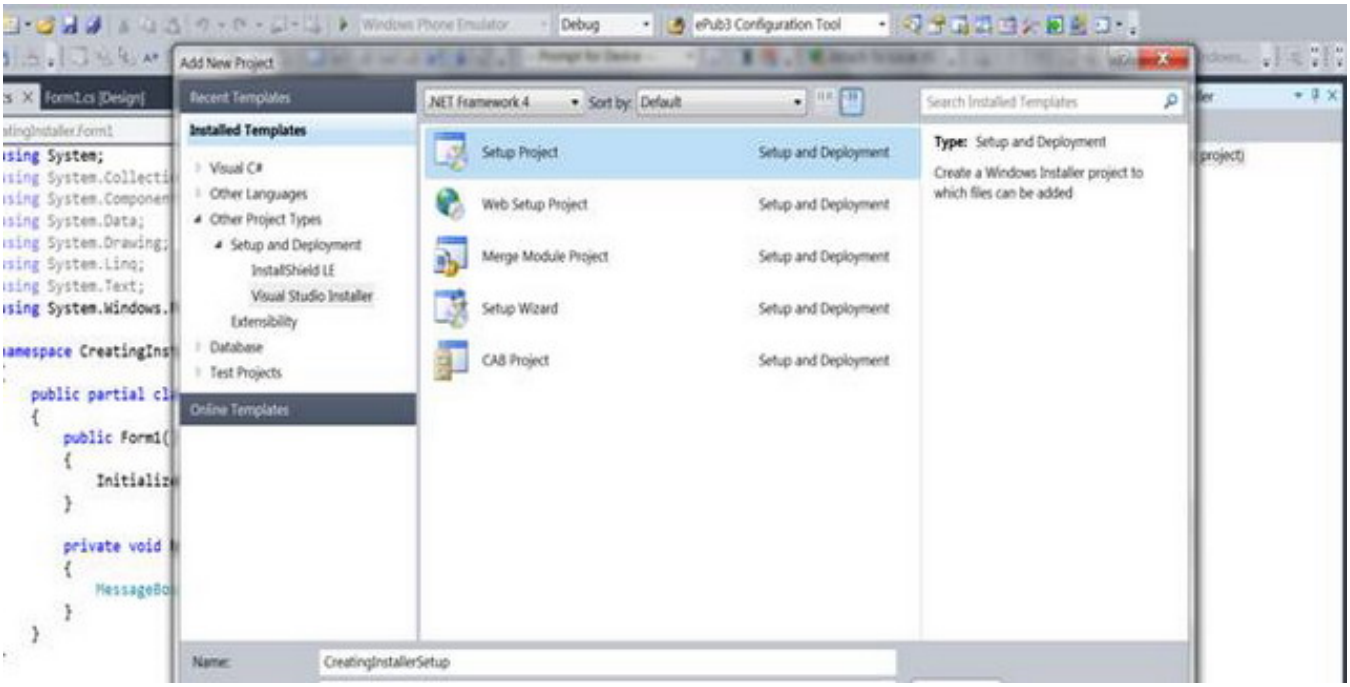


Primary Objective

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:

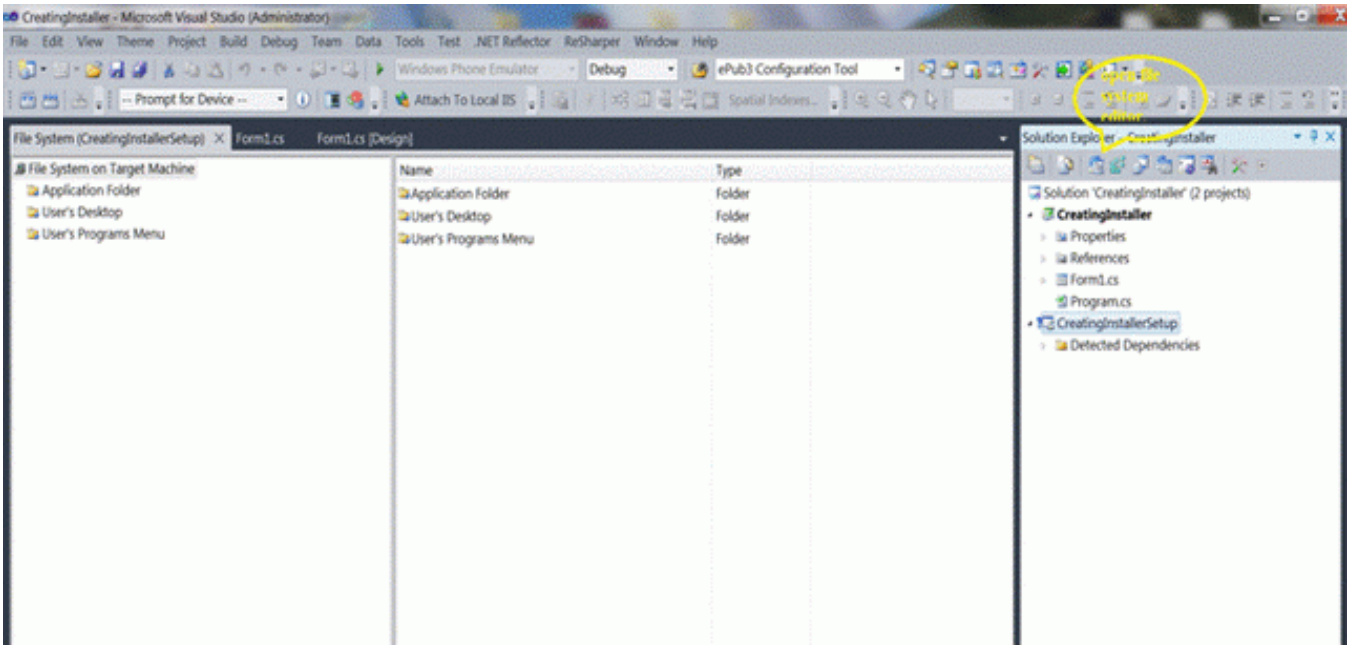


And add a setup project by **Other project Types->Setup and Deployment->Visual Studio Installer**

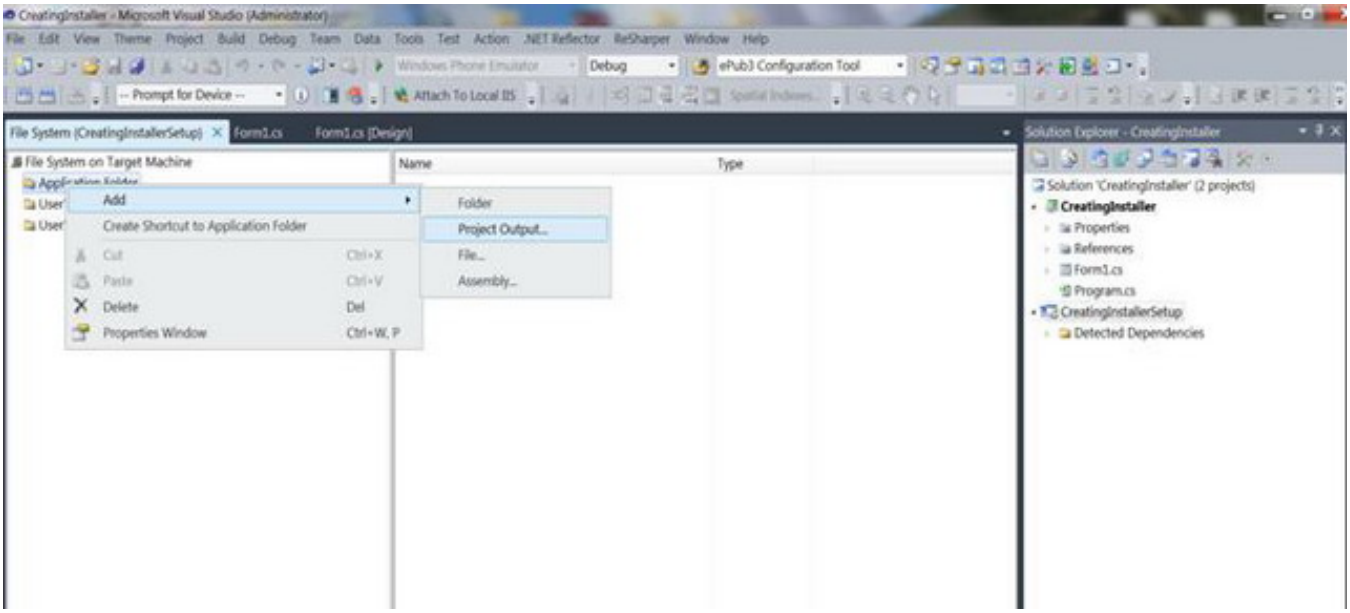


The project will be added to the solution. Now open the file system editor by clicking on the project and select the option to open file system editor, like in below figure:

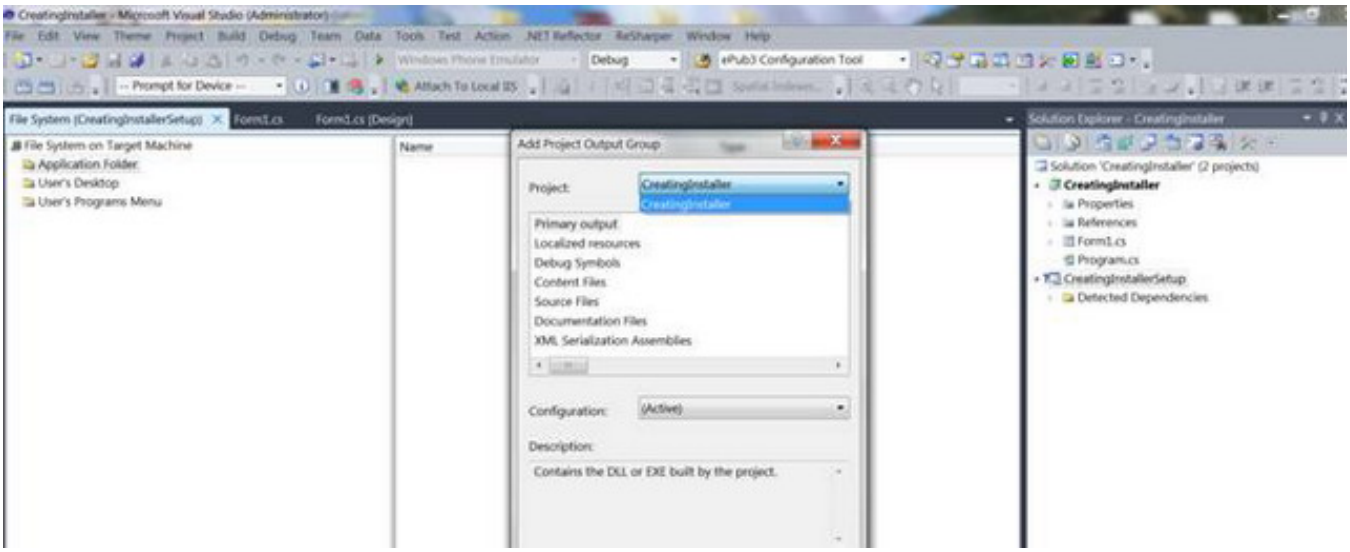
You'll get to see Application Folder, User's Desktop and User's Program Menu.



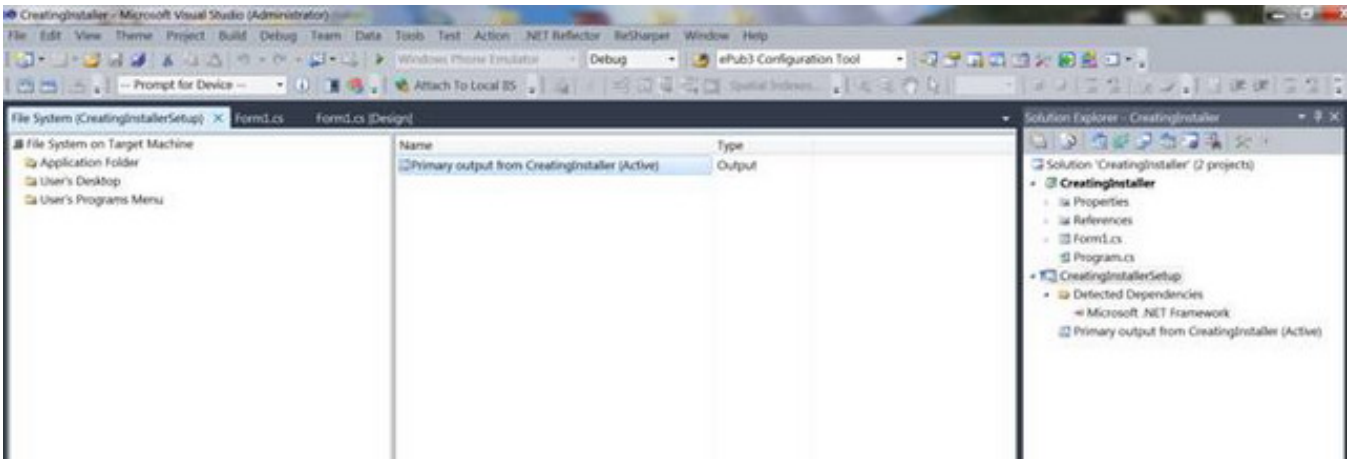
Right click on Application Folder and add an output project. Out project specifies the project we are creating an installer to, like in the following figure:



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.

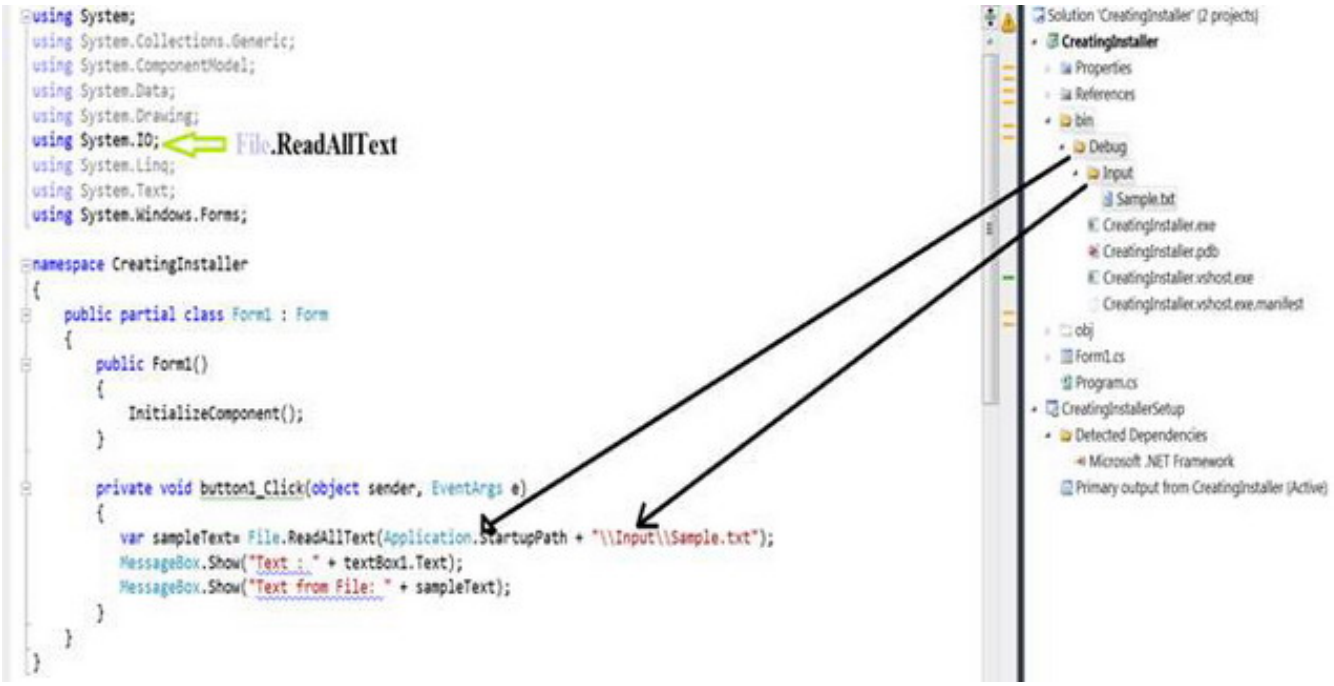


In the meanwhile, let's add some more functionality to our windows application. Let's read a file and show its

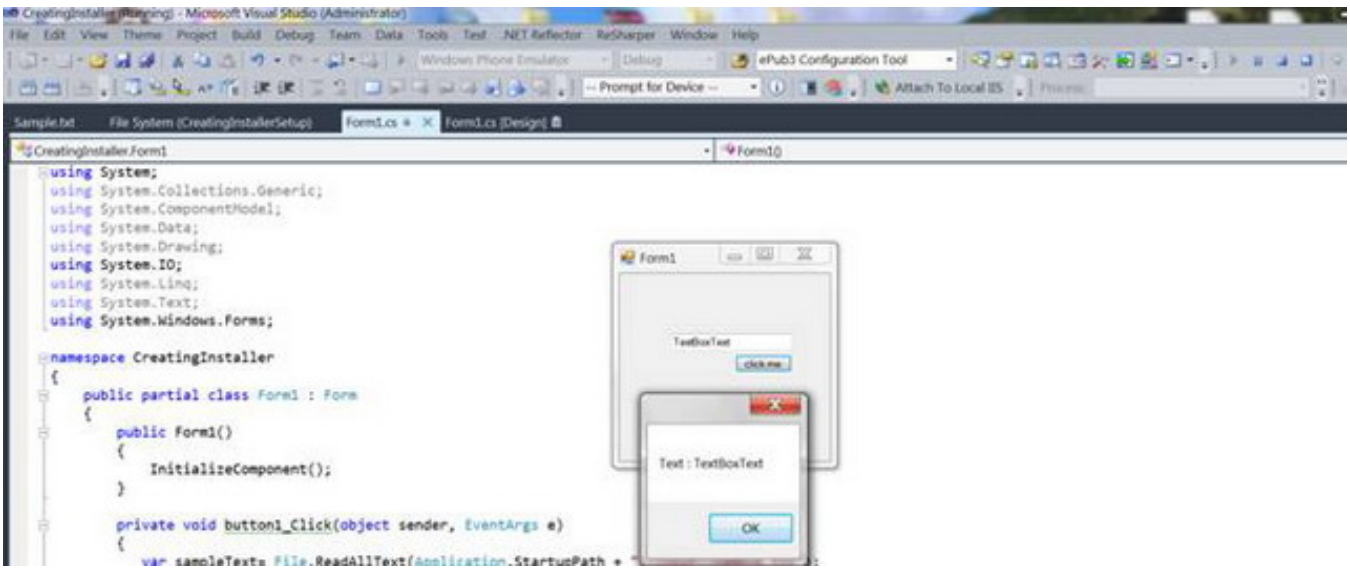
output in a message box upon a button click. Therefore, just add a text file. I called it *Sample.txt* to the *bin\debug\Input* folder, input is the custom folder I created to place my txt file.

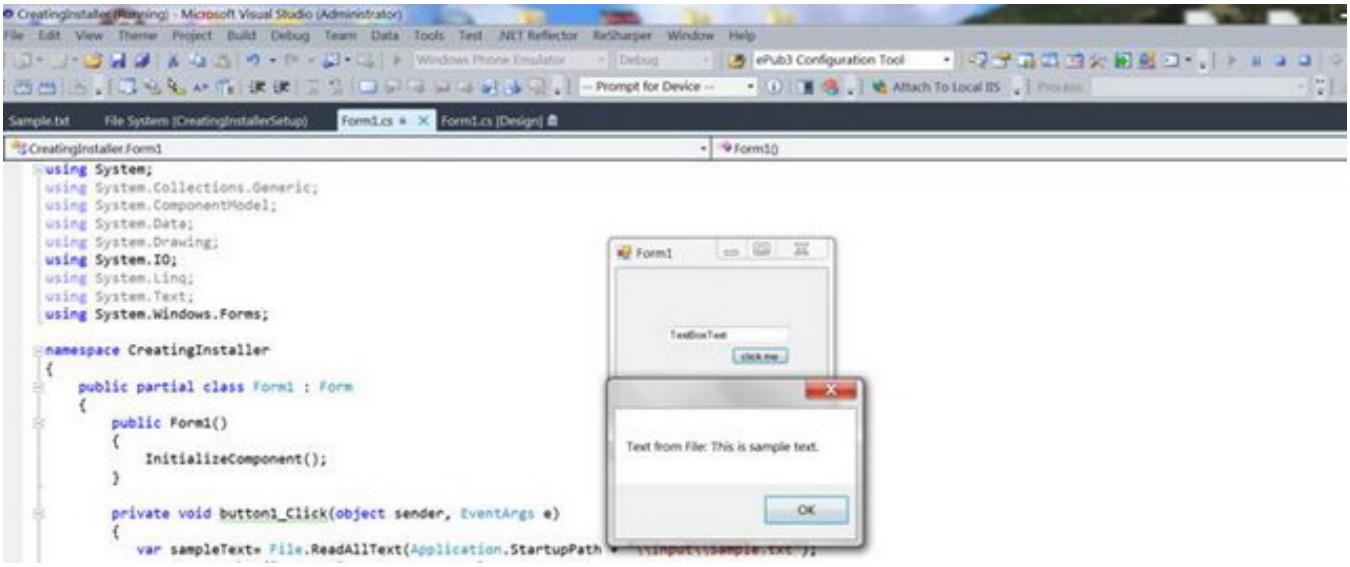
Write a few lines of code just to read the txt file from the Startup path. In my case *bin\debug*, it could also be *bin\release* as per the project build, and specify the file folder name and file name to read the content. I chose to keep my txt file at the startup path so that I could explain how we 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.



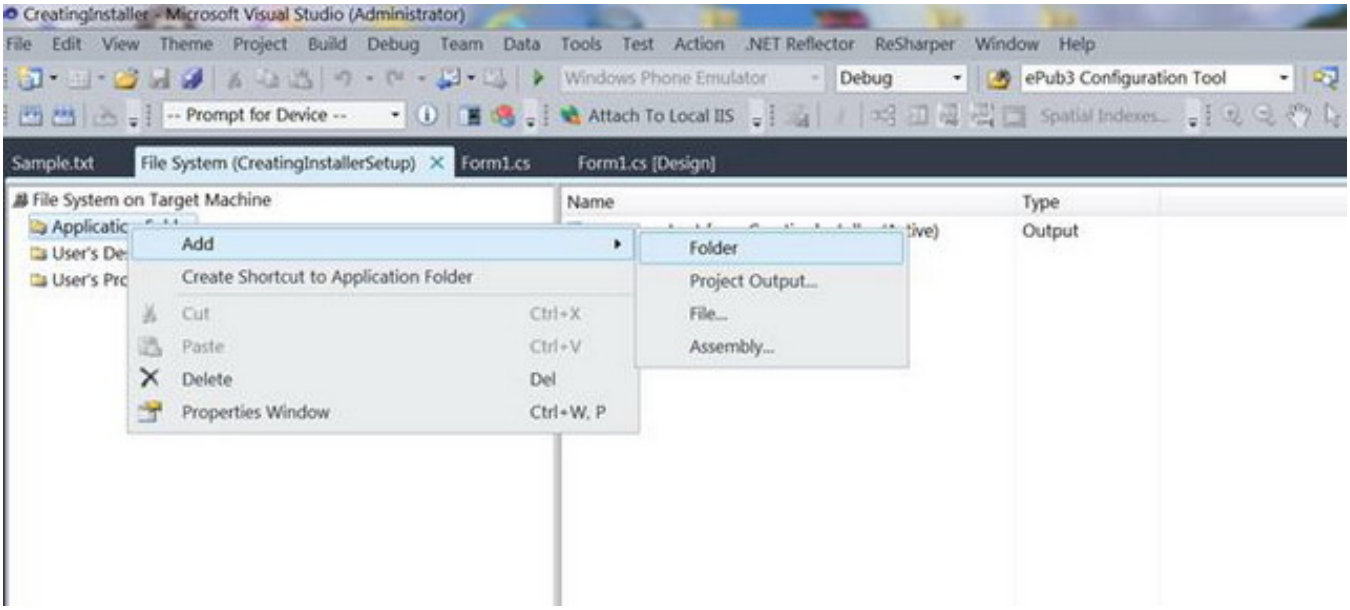
Therefore, running the application will show two message boxes, one after the other showing text box text and text from *Sample.txt* file.



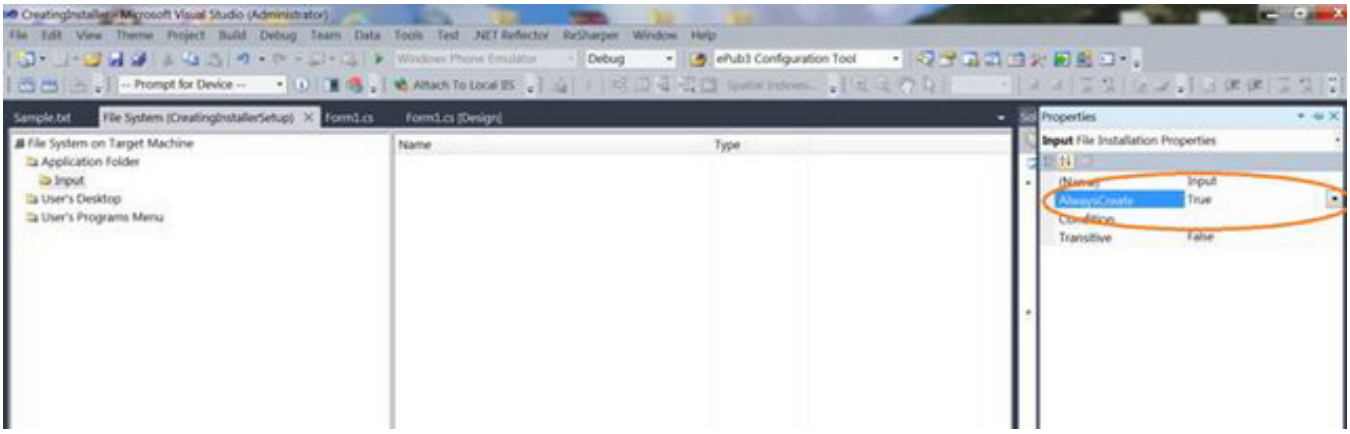


Now this folder creation logic has to be implemented in our setup project, so that when the application installs, it has all the pre-requisites required to run the application, like the Input folder and the *Sample.txt* file.

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**.

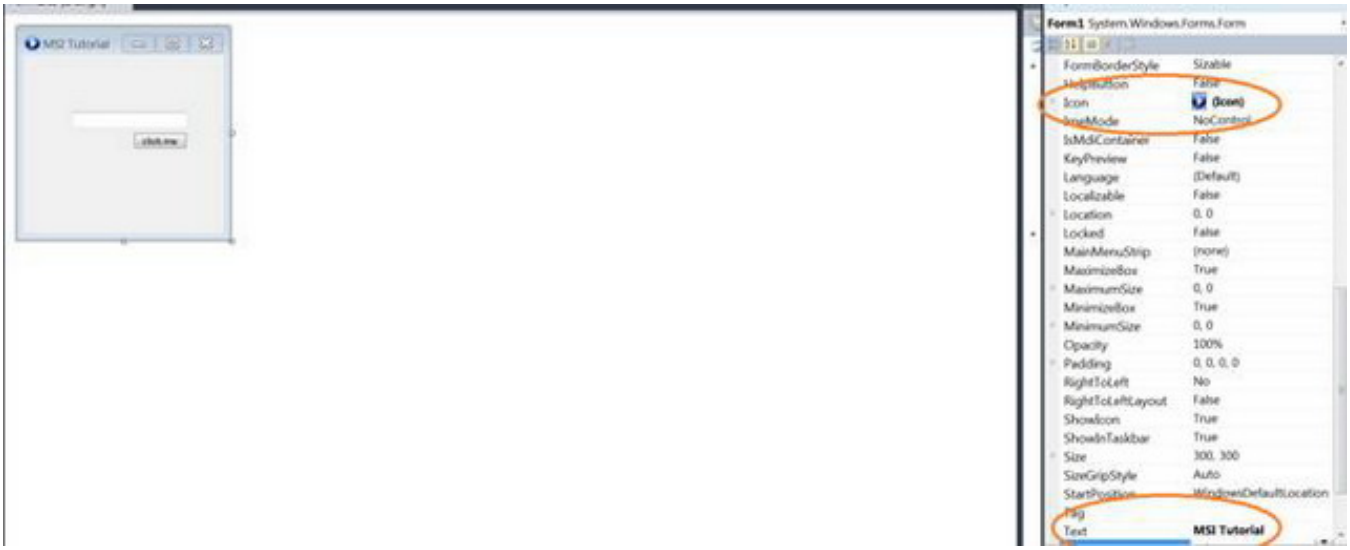


Right-click on folder, select properties, and mark the Always Create property to True. That means the folder will always be created whenever we run the installer, after a fresh build release.

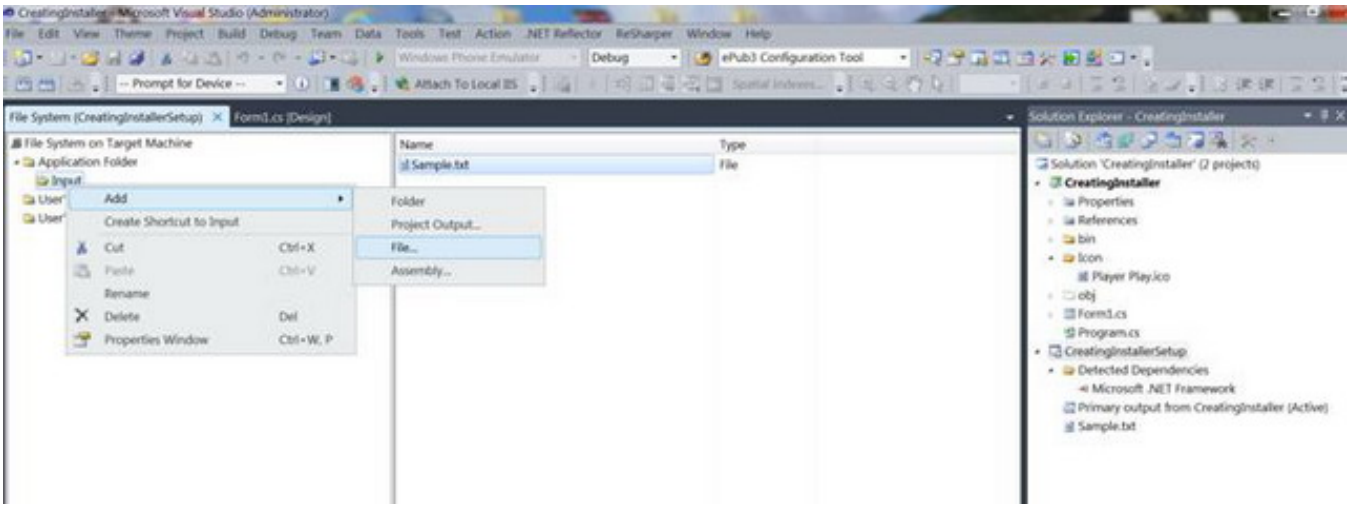


Create Shortcuts

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:

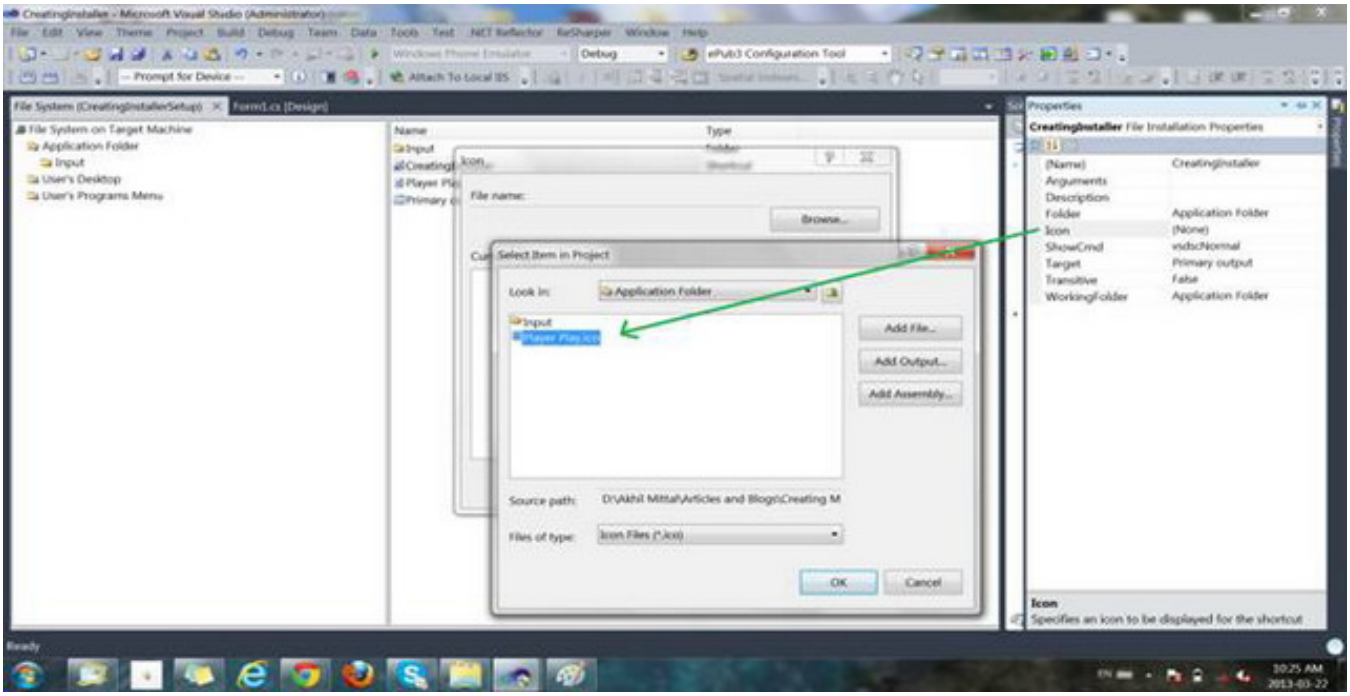


Time to add the *Sample.txt* file. Right click the Input folder created and Add file to it, browse for the *Sample.txt* file in the Windows Application project we created earlier.

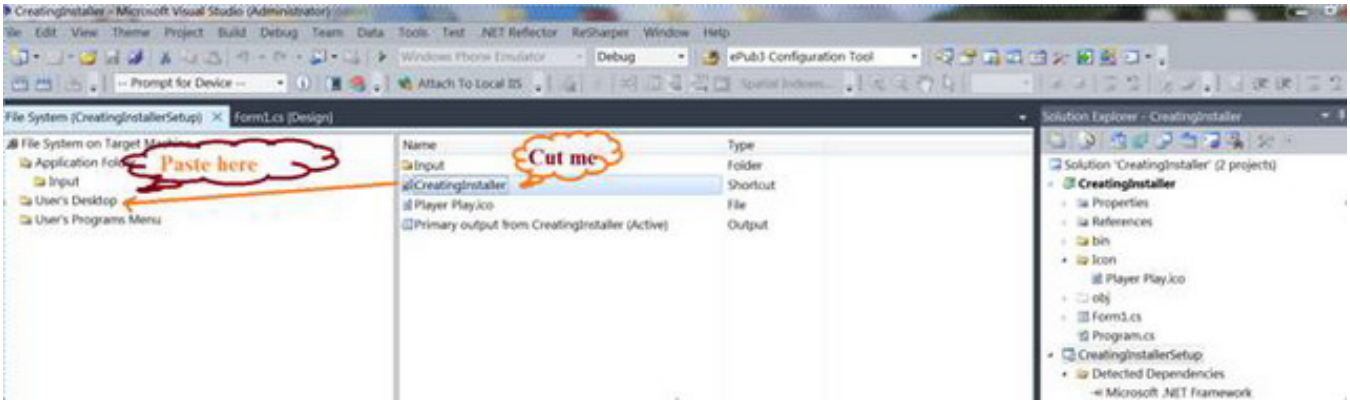


To create a shortcut to the application, right click on Primary output in middle window pane and select Create shortcut to Primary output, name that shortcut as CreatingInstaller.

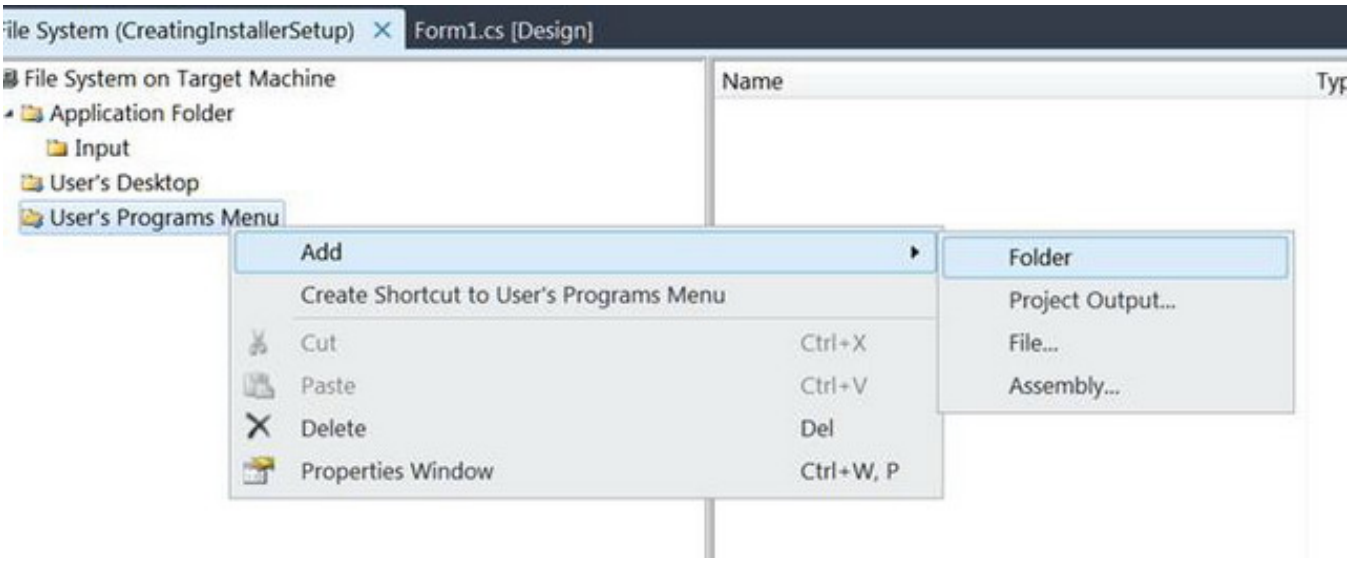
Select the properties of the shortcut, by right clicking it and add an icon to it. This icon will be created on the desktop when the application launches. The below figures explain how to add an icon.



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

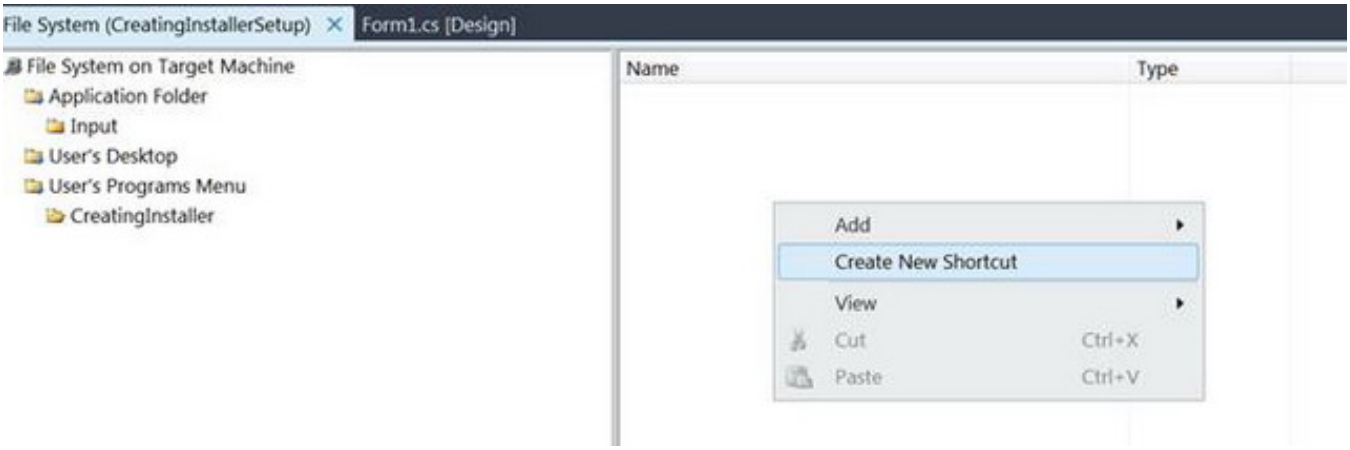


For shortcuts to be created at the User's Program Menu, add a new folder to the User's Program Menu. This will be created at the program's menu location in that folder. Create a new shortcut pointing to the primary output as we did when we created a desktop shortcut. The three images below describe the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



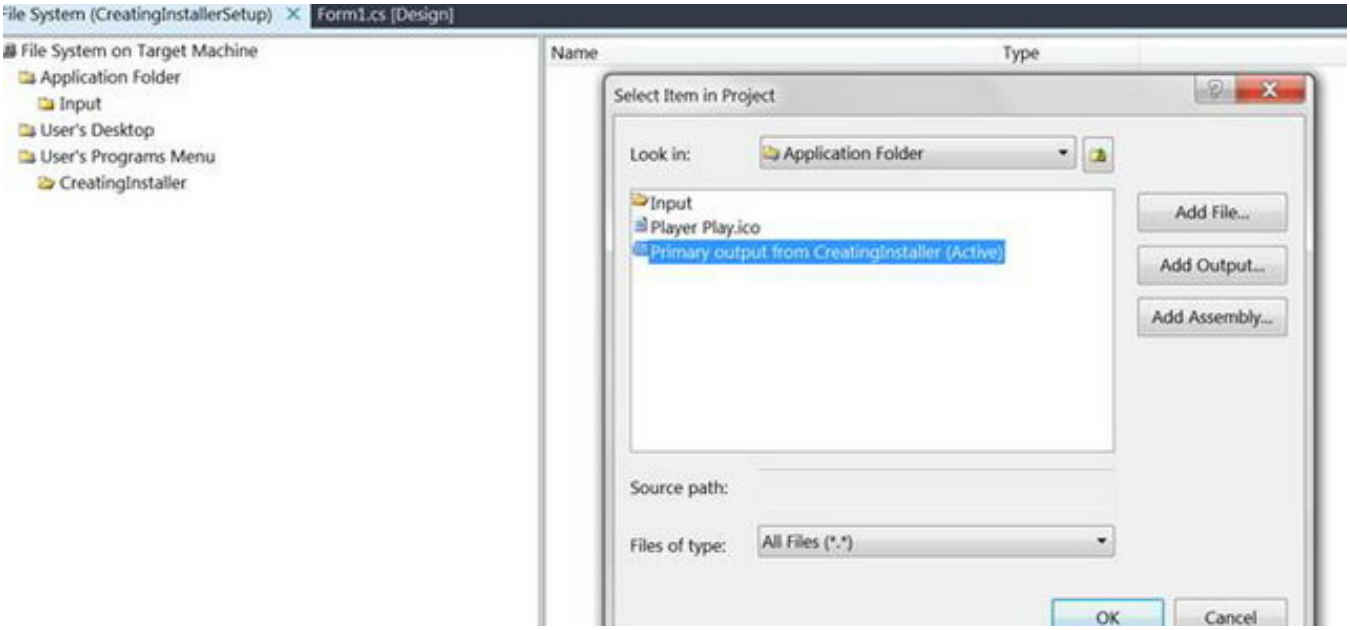
Name the folder CreatingInstaller.

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

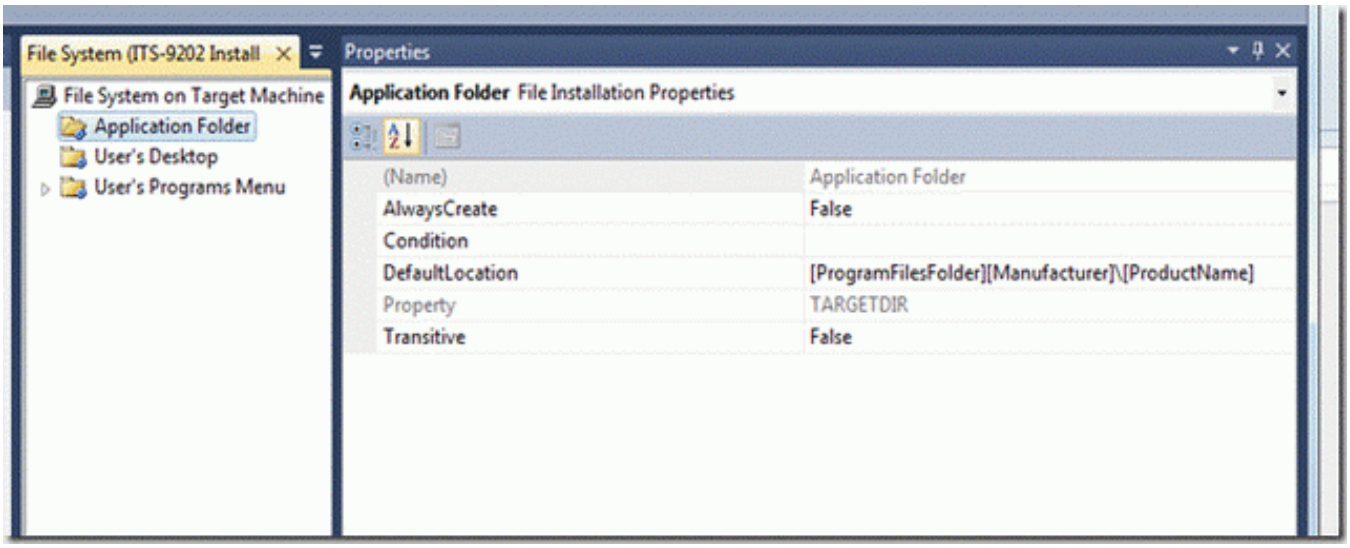


Select shortcut source to primary output selected.

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



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

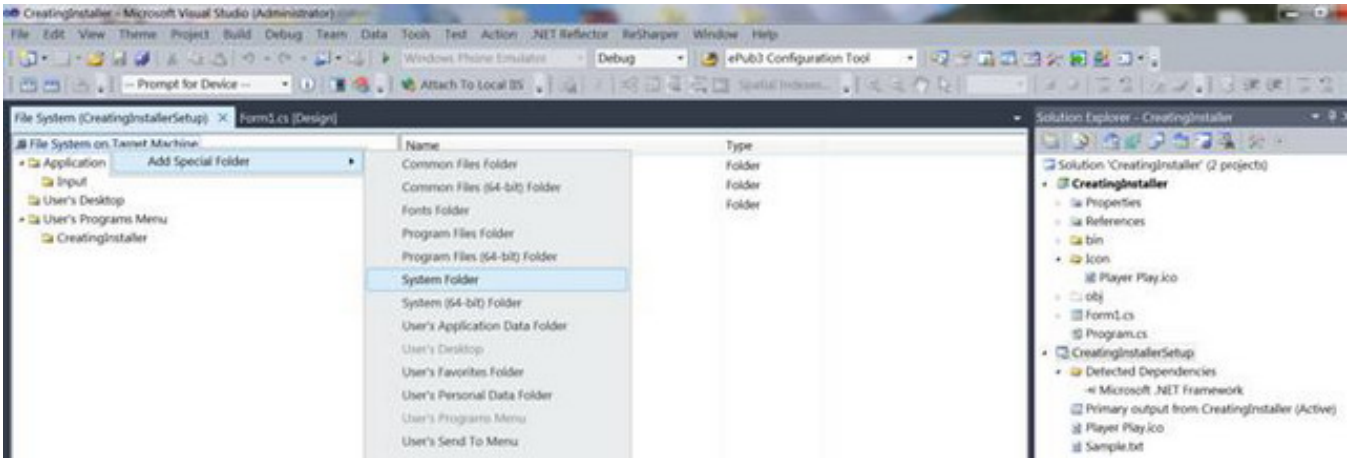


Uninstall

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.

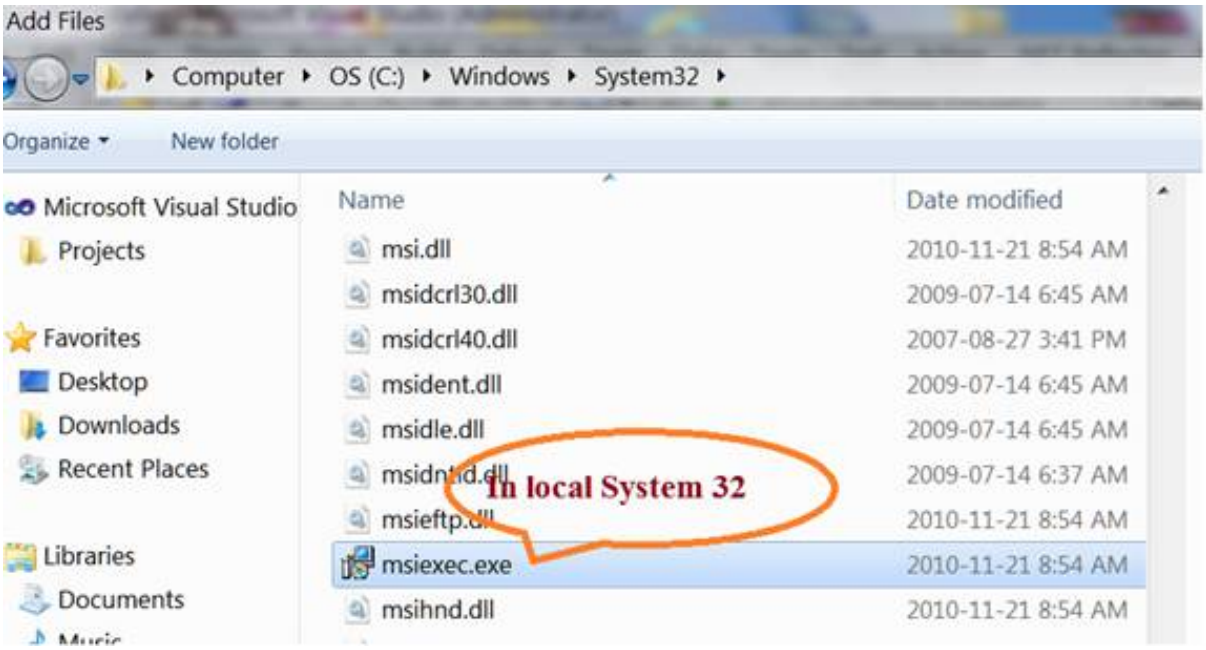
Step 1

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

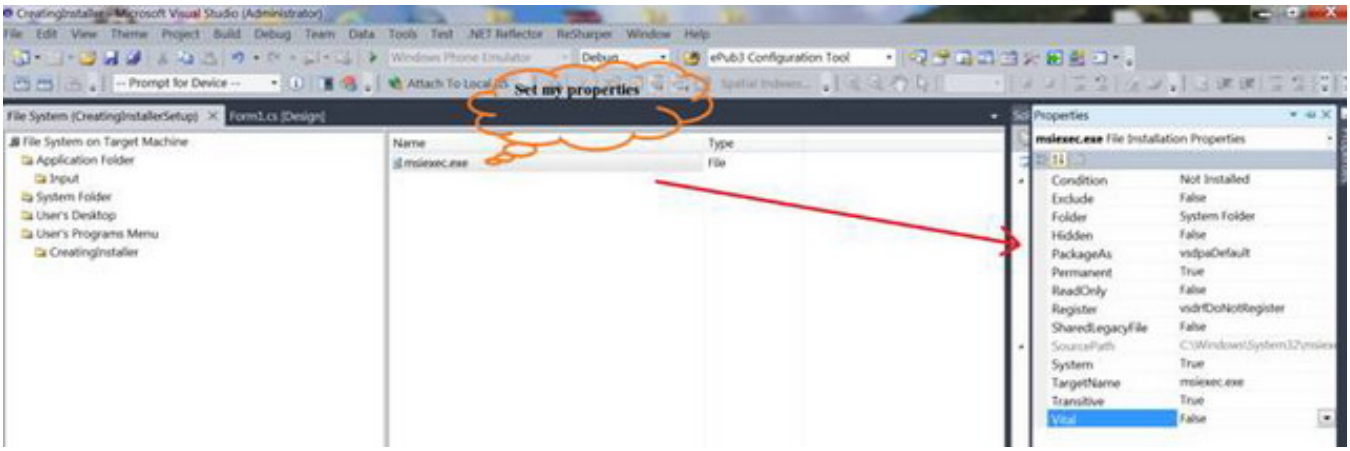


Step 2

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.

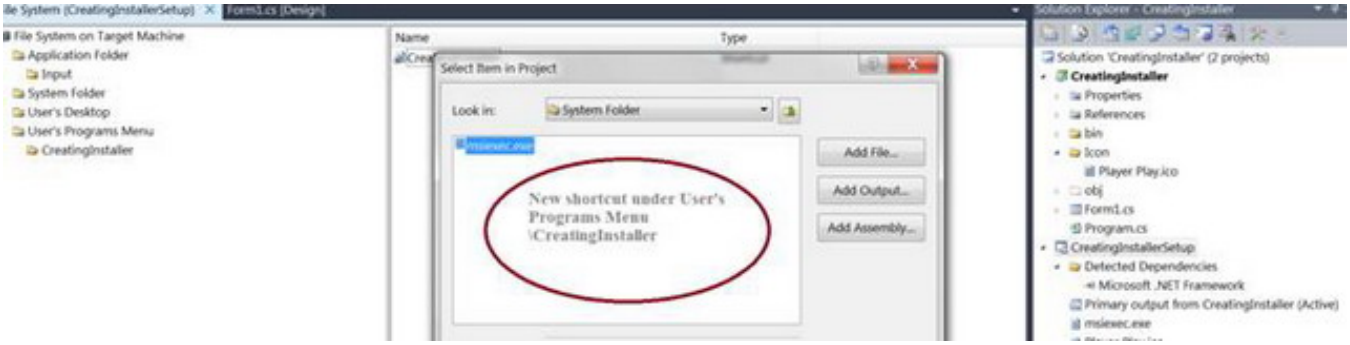


Set the properties of the file exactly as shown in the figure:



Step 3

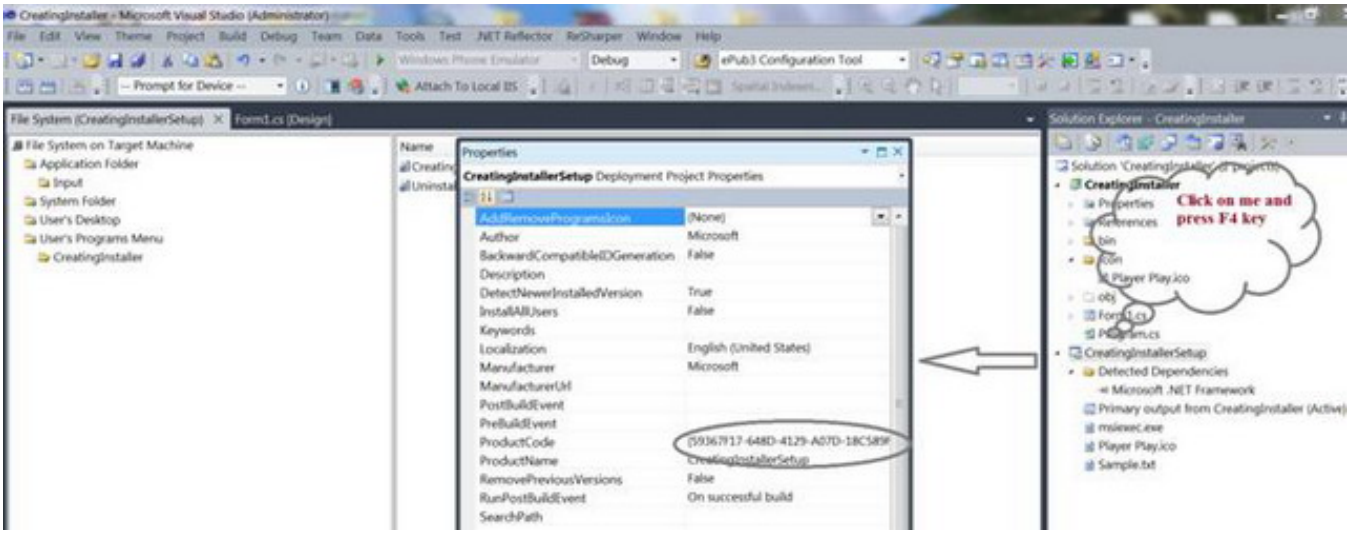
Now create a new shortcut under the User's program Menu and point its source to *msiexec* as shown below. You can add more icons and a name to your shortcut. I have given it the name "Uninstall."



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.

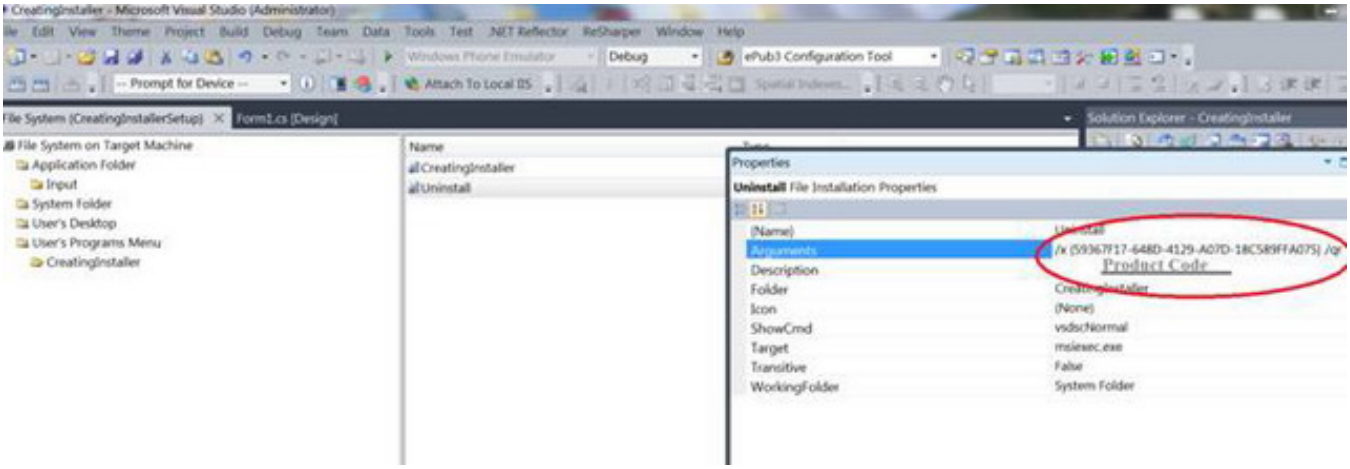


Step 5

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

```
/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. Chose whichever one you like.



Step 6

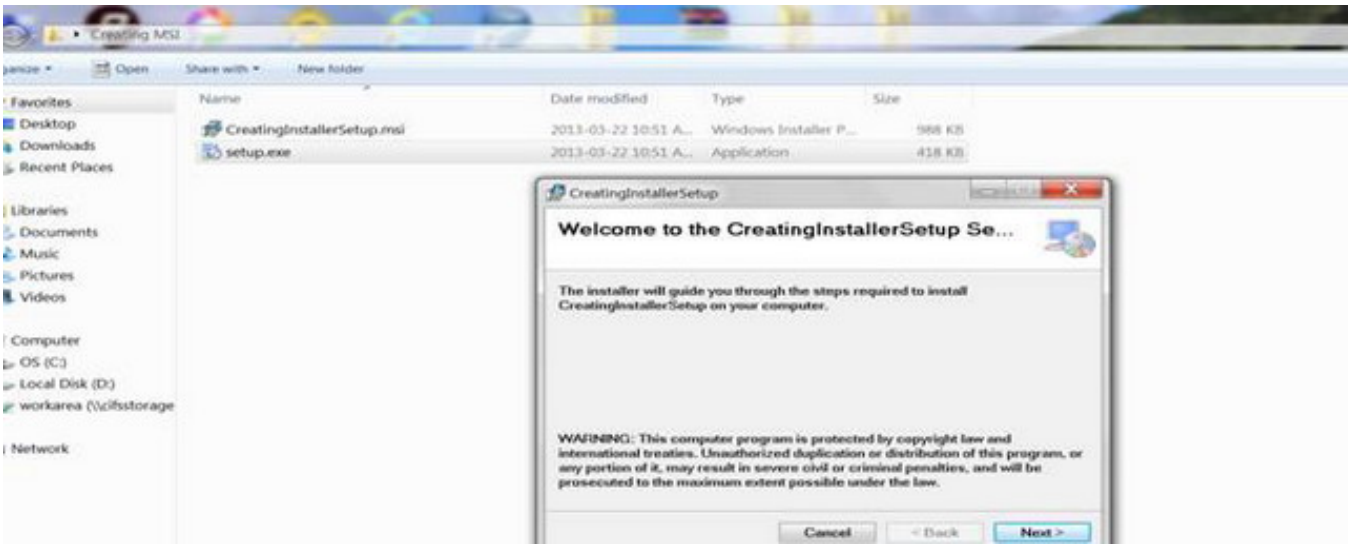
Save all and Rebuild the setup project.

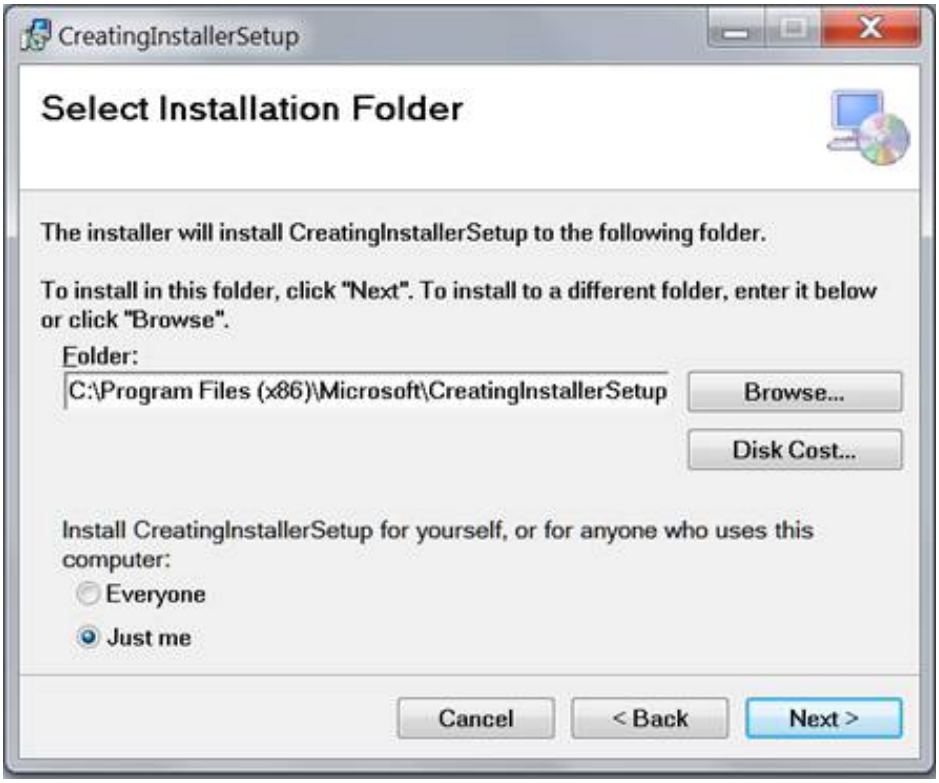
Job Done!

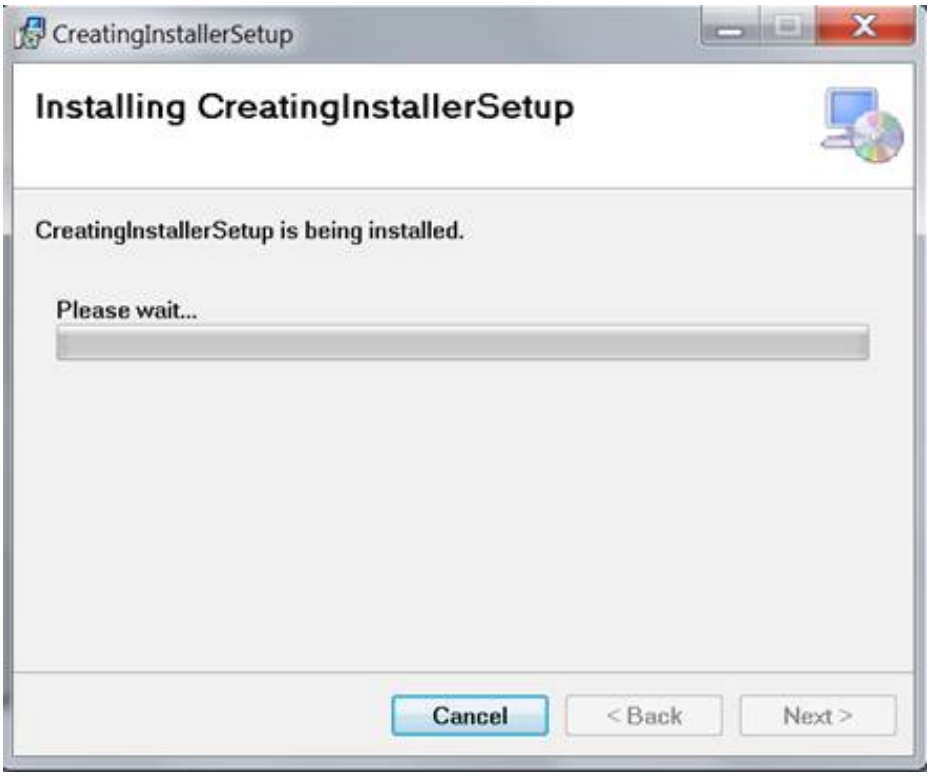
Now our setup is ready to install our windows application.

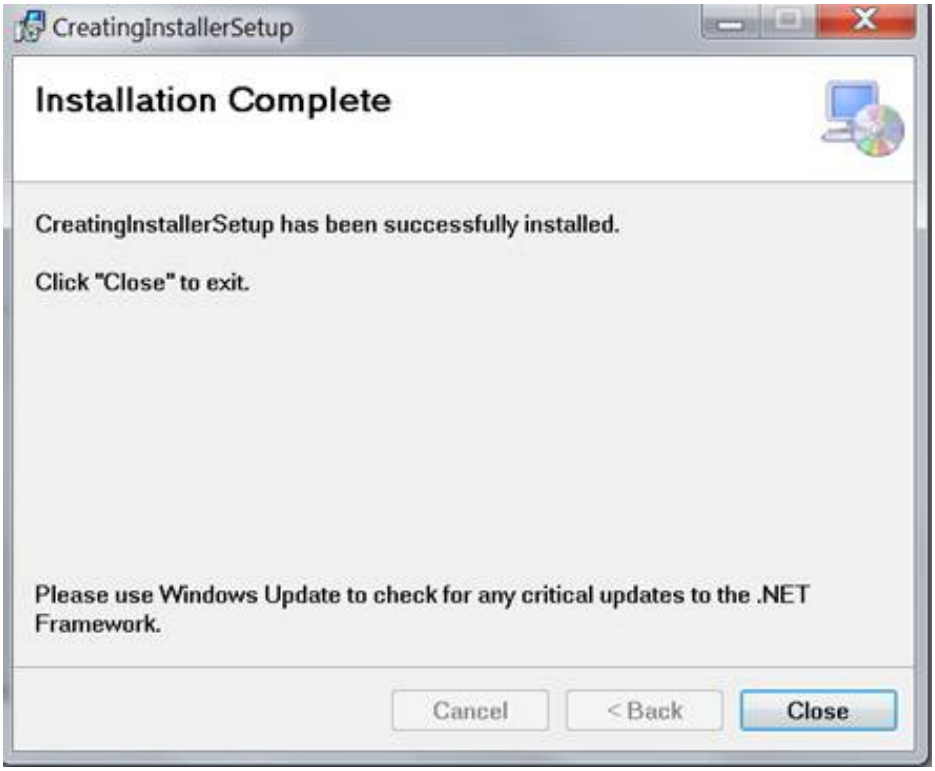
Just browse the debug folder location of Setup project. We find an *msi* and a *setup.exe*. You can run either to initiate setup.

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.)



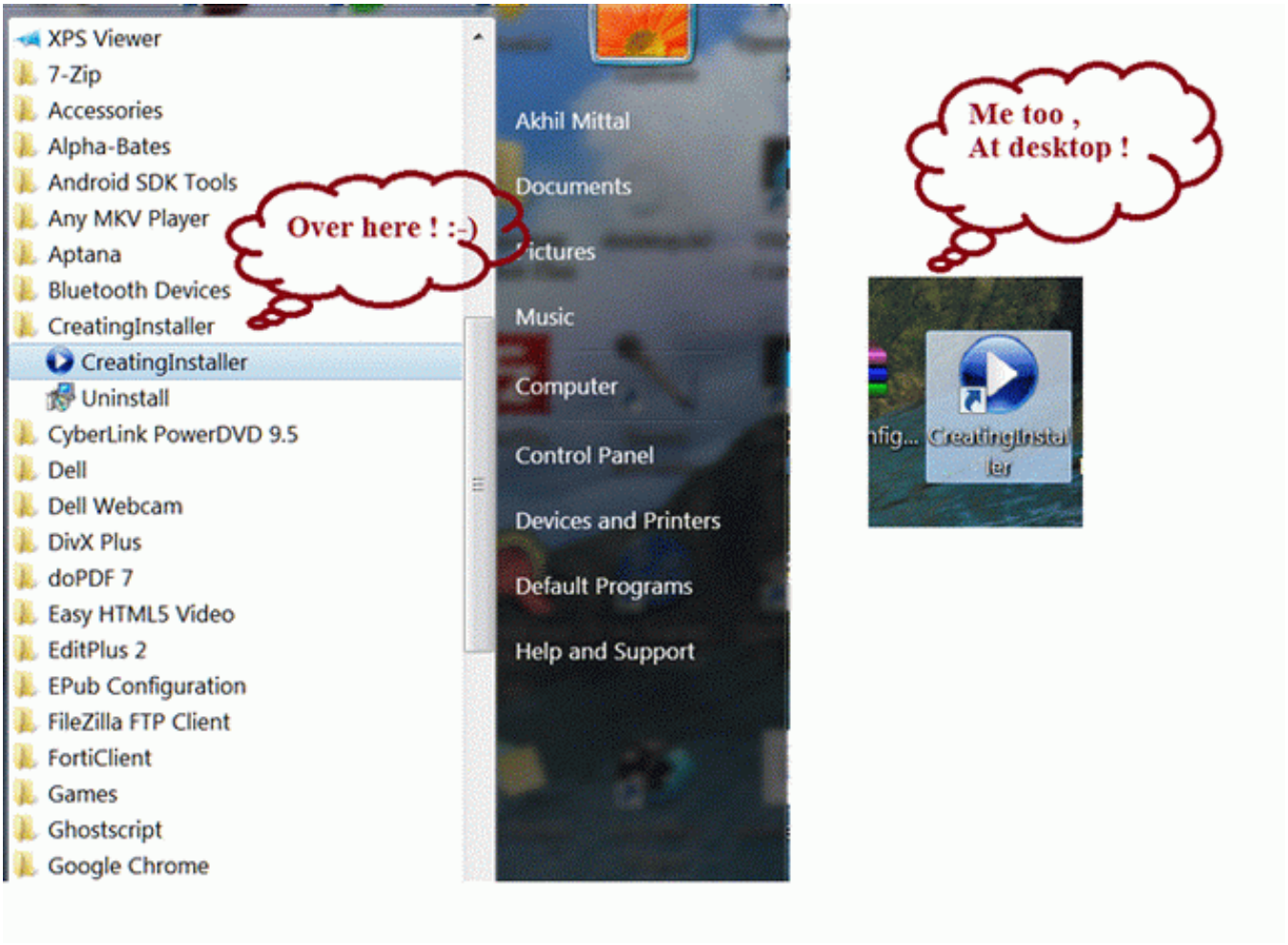

CreatingInstallerSetup
Select Installation Folder
The installer will install CreatingInstallerSetup to the following folder.
To install in this folder, click "Next". To install to a different folder, enter it below or click "Browse".
Folder:
C:\Program Files (x86)\Microsoft\CreatingInstallerSetup
Browse...
Disk Cost...
Install CreatingInstallerSetup for yourself, or for anyone who uses this computer:
Everyone
Just me
Cancel
< Back
Next >


CreatingInstallerSetup
Installing CreatingInstallerSetup
CreatingInstallerSetup is being installed.
Please wait...
Cancel
< Back
Next >



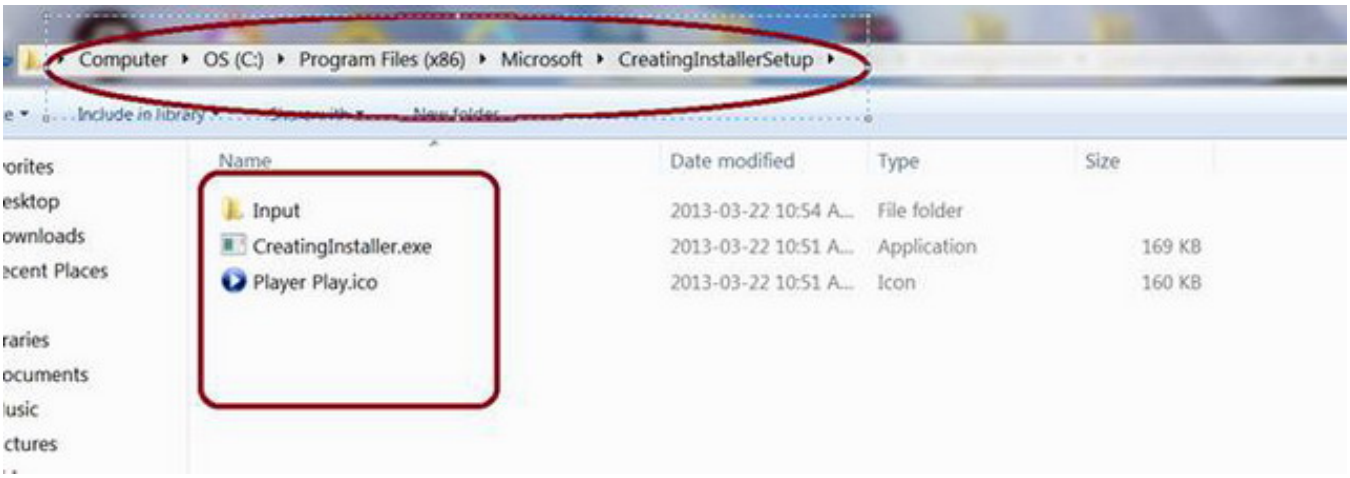
After completing the wizard, click the close button.

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.

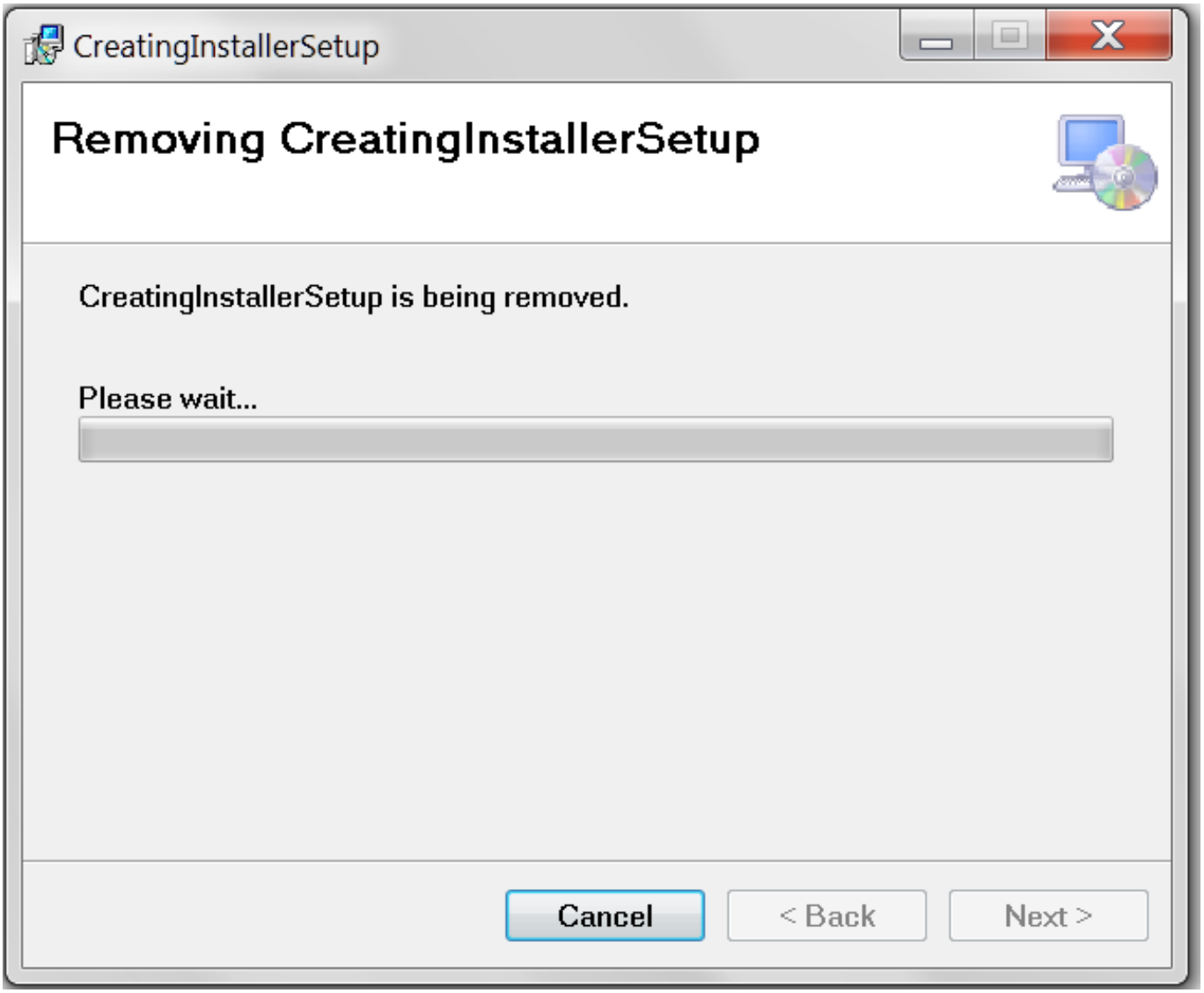


Now if we navigate out to the installation location we can also see the Input folder created and the *Sample.txt* file resting inside it.

Run the application and see the output.



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

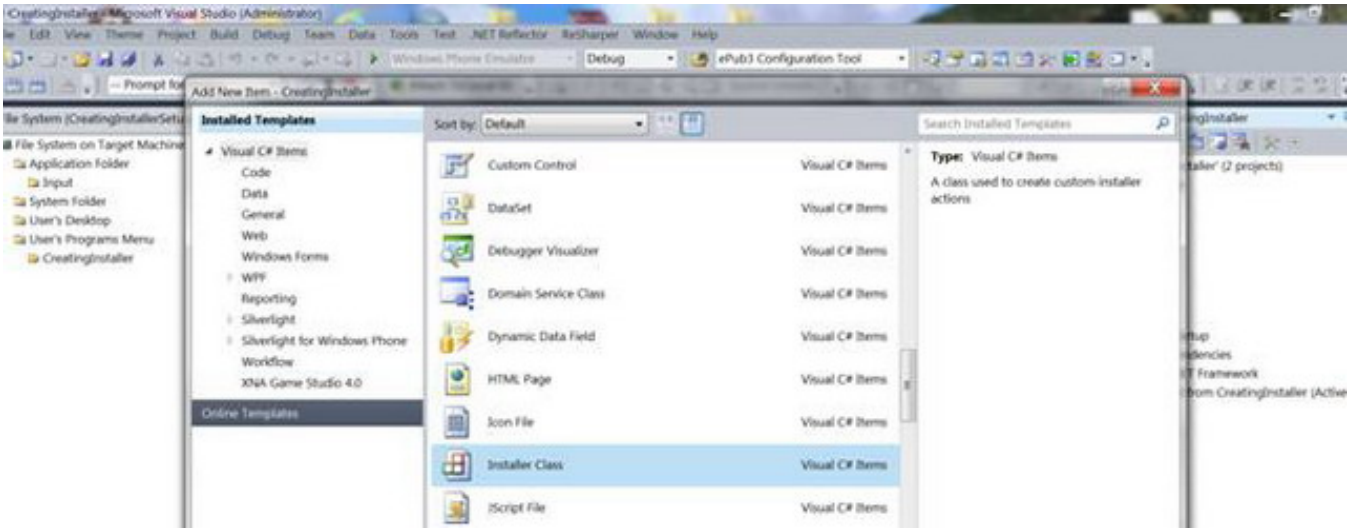


Custom Actions

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.



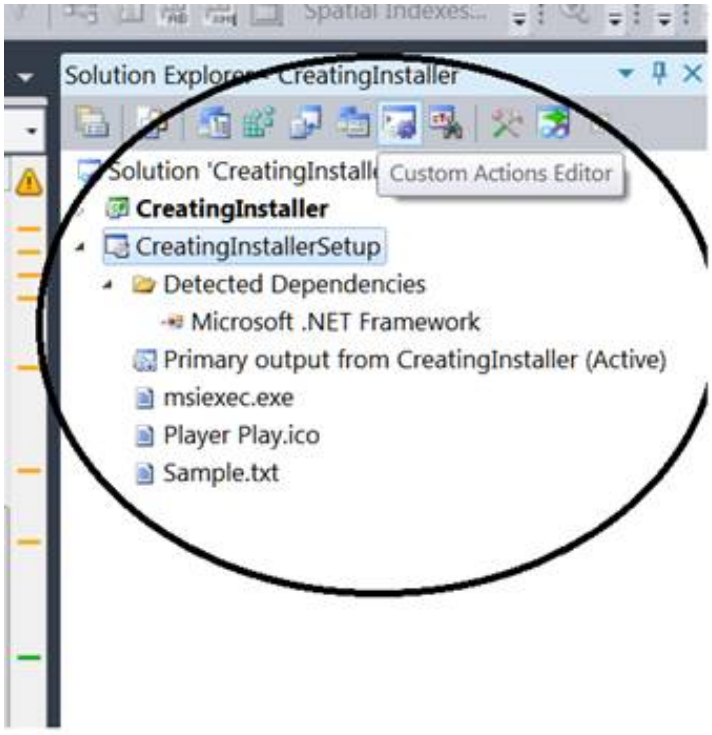
My need was to write code for the uninstallation, so I wrote few lines to fulfill the need.

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.

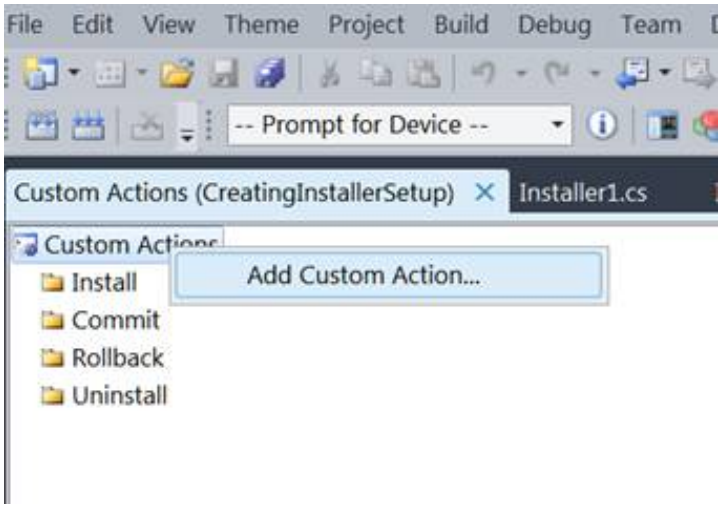
```
using System;
using System.Collections;
using System.Collections.Generic;
using System.ComponentModel;
using System.Configuration.Install;
using System.Diagnostics;
using System.Linq;


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);
            }
        }

    }
}
```

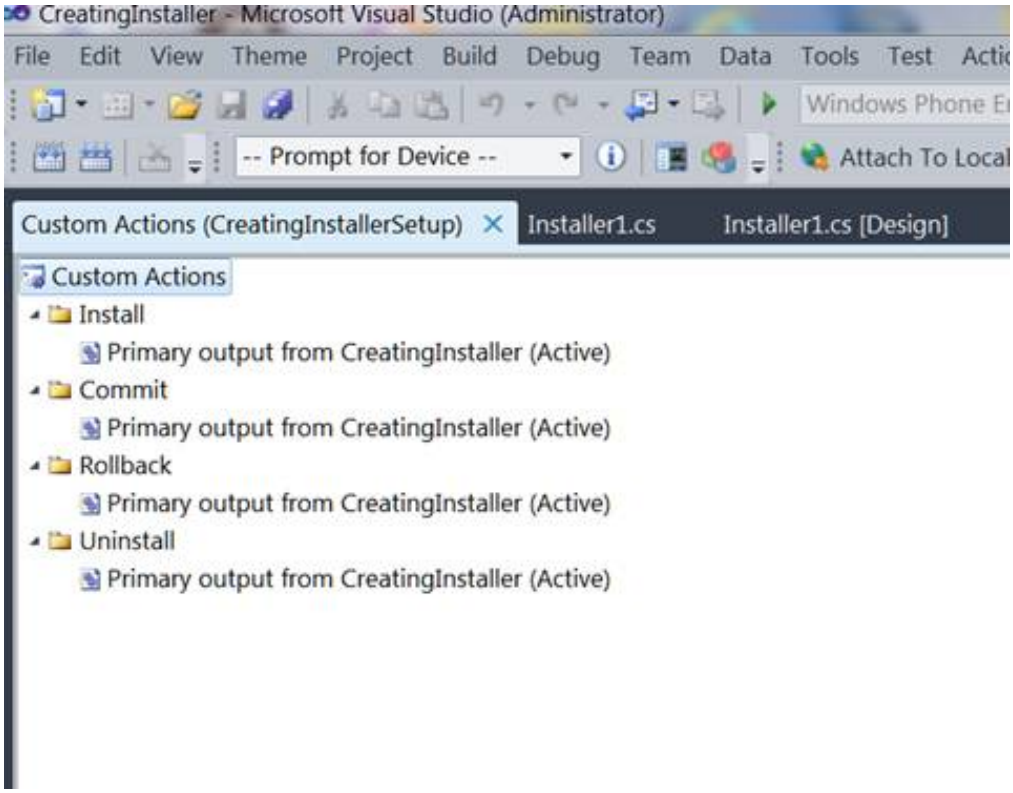
2. Click on the Custom Actions Editor after selecting the **CreatingInstallerSetup** project.



3. 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.



4. 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.



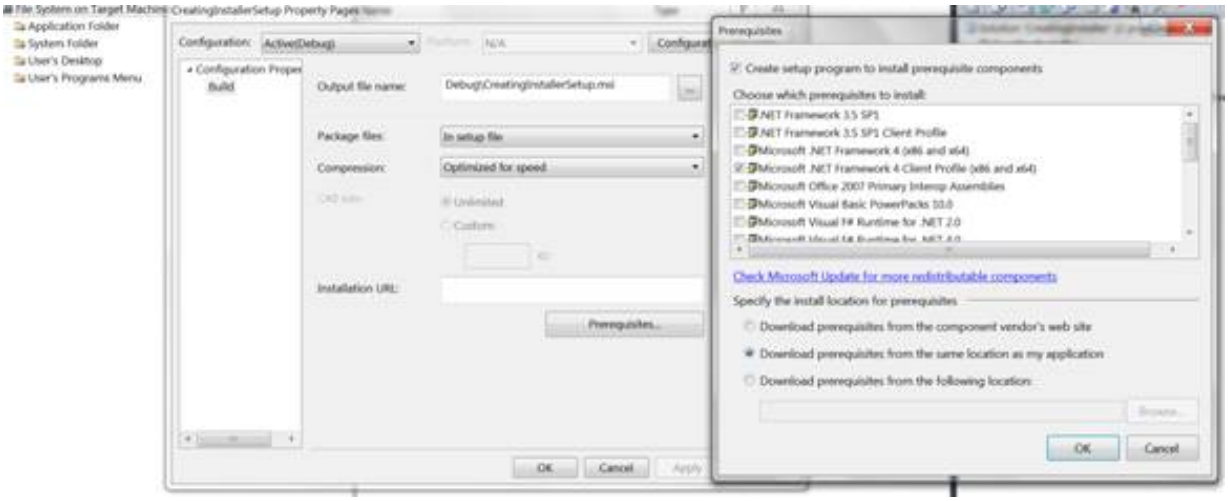
.NET Framework

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.

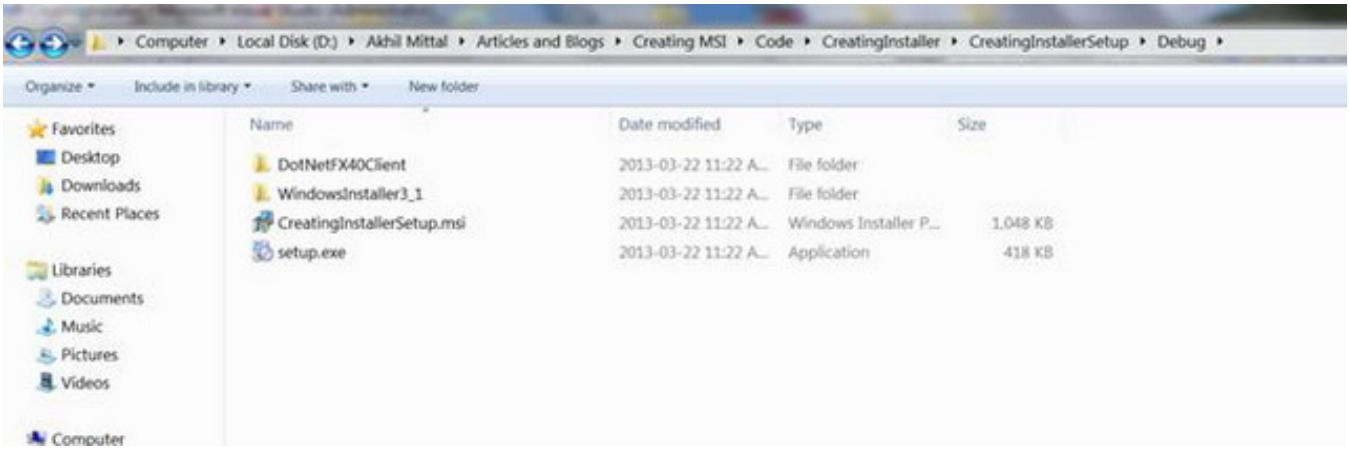
Here we can specify pre-requisites for the application to install. Just click on Prerequisites button and in the opened prerequisites window, select the checkbox for the .NET Framework application that needs to follow, and

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** for more informative articles.

License

About the Author

Akhil Mittal
Technical Lead
India

I am a C# Corner MVP,a Code project MVP,author,blogger and currently working as an Analyst in an MNC and have an experience of more than 6 years in C#.Net. I am a B.Tech in Computer Science and hold a diploma in Information Security and Application Development. My work experience includes Development of Enterprise Applications using C#,.Net and Sql Server,Analysis as well as Research and Development. I am a MCP in Web Applications(MCTS-70-528,MCTS-70-515) and .Net Framework 2.0 (MCTS-70-536). Please visit my blog **A Practical Approach** for more informative articles.

Article of the Day on Microsoft's site http://www.asp.net/community/articles on 16 August 2013.
Article of the Day on Microsoft's site http://www.asp.net/community/articles on 28 August 2013.
Article of the Day on Microsoft's site http://www.asp.net/community/articles on 08 Sept 2013.
Article of the Day on Microsoft's site http://www.asp.net/community/articles on 29 Sept 2013.
Article of the Day on Microsoft's site http://www.asp.net/community/articles on 29 Oct 2013.
Article of the Day on Microsoft's site http://www.asp.net/community/articles on 21 May 2014.
Member of the month for July 2013 on C# Corner
Month Winner for July 2013 on C# Corner
http://www.c-sharpcorner.com/News/3067/july-2013-month-winners-announced.aspx
Month Winner for May 2014 on C# Corner
http://www.c-sharpcorner.com/News/3798/may-2014-month-winners-announced.aspx

Group type: Collaborative Group

137 members

Follow on Twitter Google LinkedIn

Comments and Discussions

34 messages have been posted for this article Visit
http://www.codeproject.com/Articles/568476/Creating-an-MSI-Package-for-Csharp-Windows-Applica
to post and view comments on this article, or click **here** to get a print view with messages.