Creating Visual Studio 2010 ItemTemplate

In this article, I will demonstrate step by step process on how to create your own custom visual studio ItemTemplate and Visual Studio 2010 Installer VSIX for your item template. Before we go into details following are the prerequisites you have to have installed on your machine.

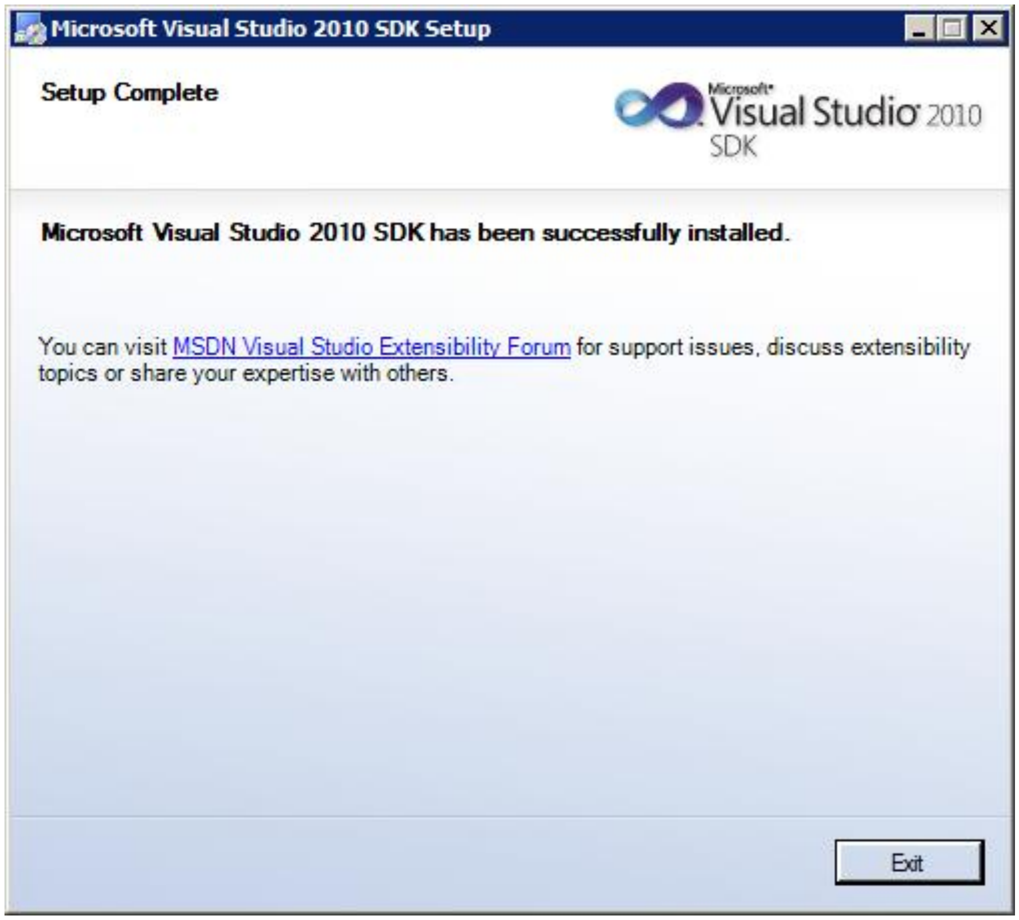
Prerequisite:-

Visual Studio Developer SDK

Visual Studio 2010

Once you download and install this Visual Studio 2010 SDK, follow the steps I described below using series of screen shot and very little explanation wherever necessary.

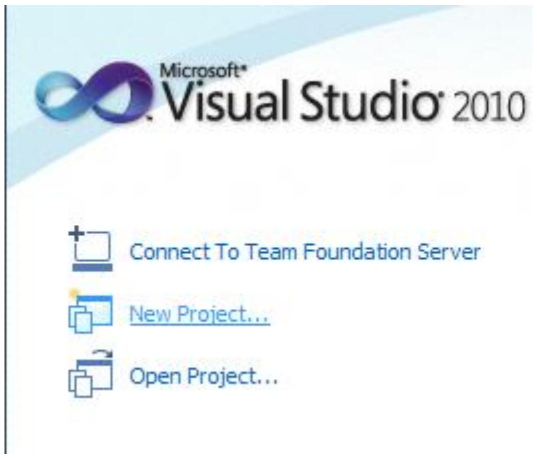
Installing Visual Studio 2010 SDK

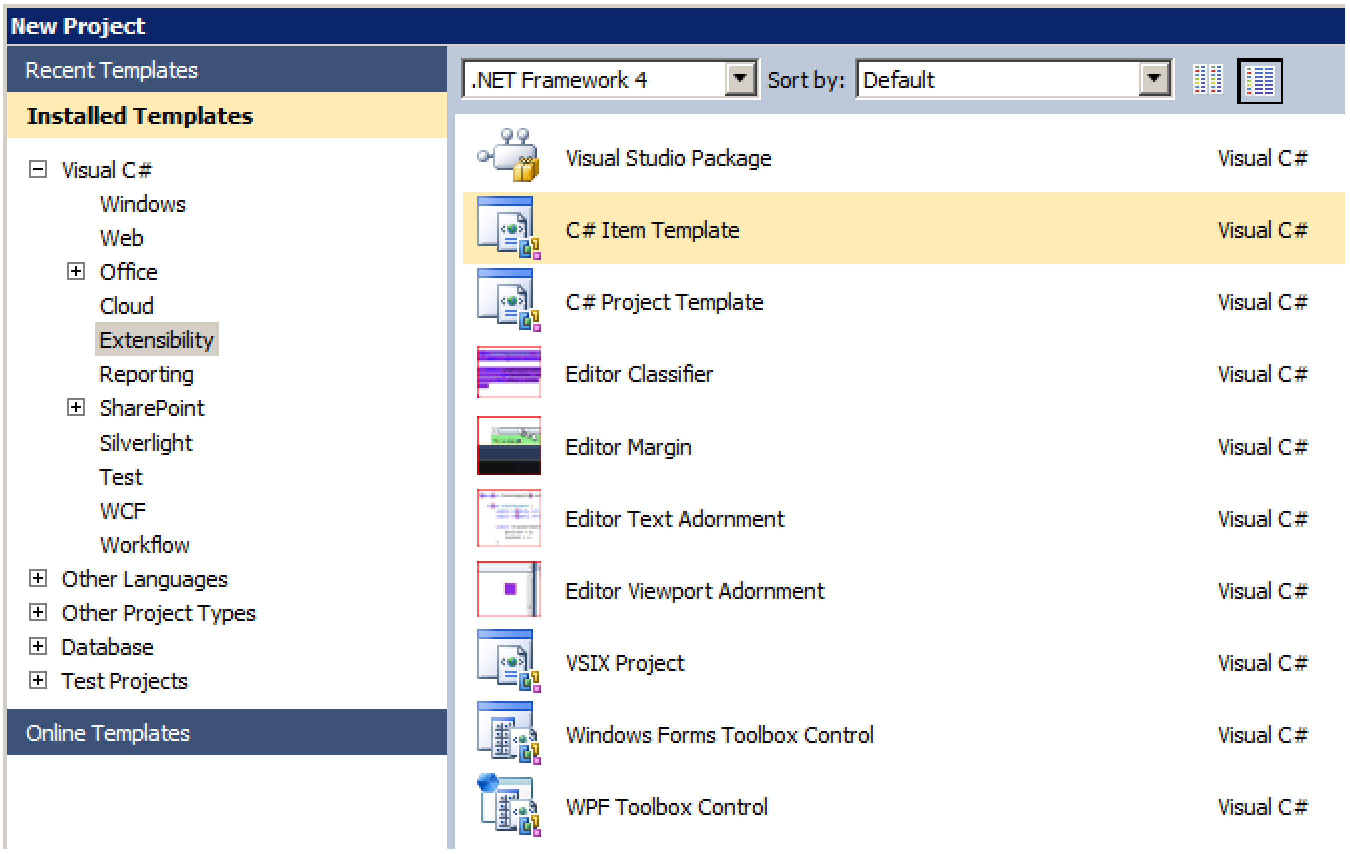

Microsoft Visual Studio 2010 SDK Setup
Setup Complete
Microsoft Visual Studio 2010 SDK
Exit

Microsoft Visual Studio 2010 SDK has been successfully installed.

You can visit MSDN Visual Studio Extensibility Forum for support issues, discuss extensibility topics or share your expertise with others.

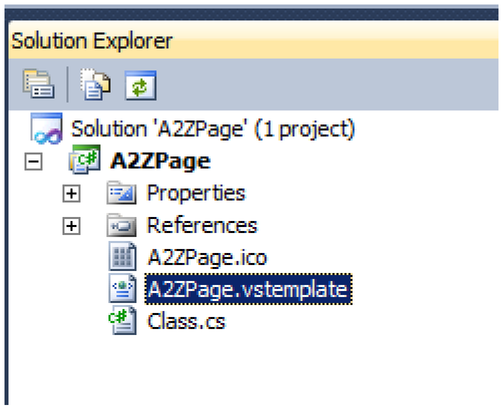
Open Visual studio and create a new ItemTemplate project using New Project Windows and selecting Extensibility templates as shown below.


Microsoft Visual Studio 2010
Connect To Team Foundation Server
New Project...
Open Project...



Visual Studio Package	Visual C#
C# Item Template	Visual C#
C# Project Template	Visual C#
Editor Classifier	Visual C#
Editor Margin	Visual C#
Editor Text Adornment	Visual C#
Editor Viewport Adornment	Visual C#
VSIX Project	Visual C#
Windows Forms Toolbox Control	Visual C#
WPF Toolbox Control	Visual C#

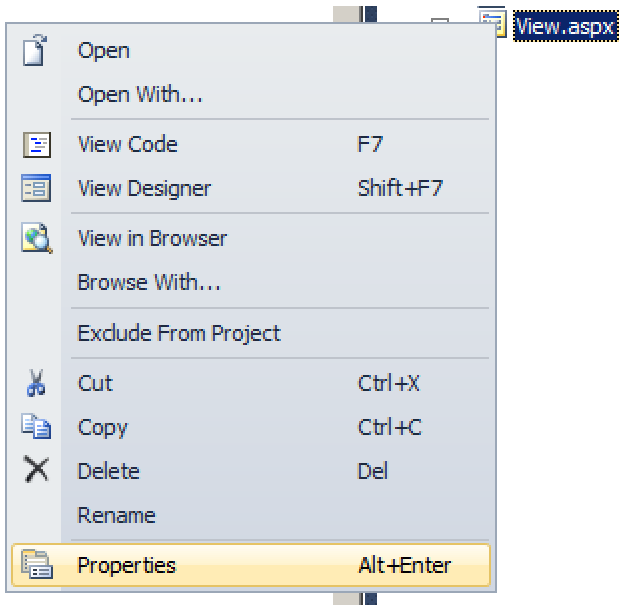
Once the project is created you, you should see the project structure as following. We will remove the class file and add our own .aspx and .aspx.cs files in this solution.

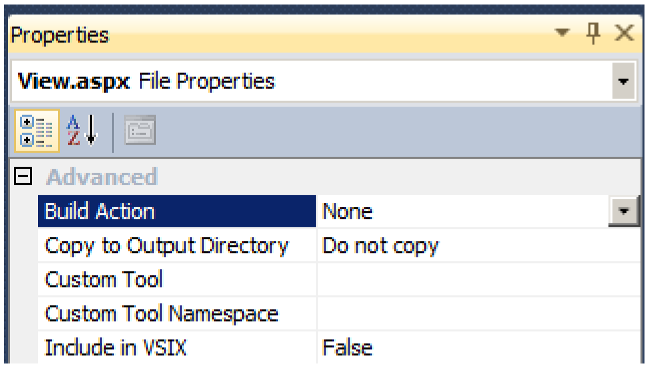


Notice the A2ZPage.vstemplate file created in our solution. That is the file where we will be doing most of our modifications later in this article. Just keep in mind that that is the file that holds all the configuration details for your ItemTemplate.

Now create/copy default.aspx & default.aspx.cs files from the project which you want to convert as a template and paste them in the solution folder of this project.

Then click on the “Show all files” icon on your solution explorer and include those two files into this ItemTemplate solution. Once you include these files into solution, go ahead and change their build action to “None” by right clicking on them, going to properties windows as shown below.





Build Action	None
Copy to Output Directory	Do not copy
Custom Tool	
Custom Tool Namespace	
Include in VSIX	False

At this point we are done with the basic configuration. Now is the time to do the main thing. Replacing our file names and namespaces into place holders so that whenever the file is added to actual project the namespaces and file names are treating accordingly. For this you will have to get yourself familiar with the XML schema of the VS templates. I have covered few basic place holders but depending upon the complexity of your project you might have to look yourself for advanced place holders. Let's examine the content of TemplatePage.aspx/ TemplatePage.aspx.cs & .vstemplate files.

TemplatePage.aspx

```
<%@ Page Title="" Language="C#" MasterPageFile="~/MySiteMasterPage.Master" AutoEventWireup="true"
ValidateRequest="false"
    CodeBehind="$safeitemrootname$.aspx.cs" Inherits="$rootnamespace$.$safeitemrootname$" %>

<?xml:namespace prefix = asp />
<asp:Content id=Content1 runat="server" ContentPlaceHolderID="HeadPlaceHolder">
</asp:Content>
<asp:Content id=Content2 runat="server" ContentPlaceHolderID="LeftPlaceHolder">
</asp:Content>
<asp:Content id=Content3 runat="server" ContentPlaceHolderID="MainPlaceHolder">
</asp:Content>
```

So, as we can see I am deriving this page from some master page and also the appropriate place holder is describe in this template. Interesting parts are CodeBehind and Inherits where we are exploring two placeholders.

$safeitemrootname$ This will be replaced by the actual name given to this file by the user while adding this file to his/her project.

$rootnamespace$ This will be replaced by the root namespace including all the folder hierarchy where your item is added into the application.

TemplatePage.aspx.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using A2ZSolutions.Web.Business;
using A2ZSolutions.Common.Controls;
using A2ZSolutions.Model;


namespace $rootnamespace$
{
    public partial class $safeitemrootname$ : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using A2ZSolutions.Web.Business;
using A2ZSolutions.Common.Controls;
using A2ZSolutions.Model;


namespace $rootnamespace$
{
    public partial class $safeitemrootname$ : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {

        }

    }
}
```

This code snippet is also using the same place holders we have explored in the aspx page. It also includes the default using statements this page should have by default when it is added to the solution are defined. And finally let's look at the .vstemplate file details. Couple of things need to notice in this file.

.vstemplate

```
<?xml version="1.0" encoding="utf-8"?>
<?XML:NAMESPACE PREFIX = [default] http://schemas.microsoft.com/developer/vstemplate/2005
                NS = "http://schemas.microsoft.com/developer/vstemplate/2005" />
<VSTemplate xmlns="http://schemas.microsoft.com/developer/vstemplate/2005"
                Type="Item" Version="3.0.0">
  <TemplateData>
    <DefaultName>A2ZSitePage.aspx</DefaultName>
    <Name>A2ZSitePage</Name>
    <Description>A2Z Site ASPX PAGE TEMPLATE</Description>
    <Icon>A2ZPage.ico</Icon>
    <TemplateID>8a8b1983-d7ab-493b-8f65-20daf25cbb4e</TemplateID>
    <SortOrder>10</SortOrder>
    <ProjectType>CSharp</ProjectType>
  </TemplateData>
  <TemplateContent>
    <References></References>
    <ProjectItem ReplaceParameters="true" TargetFileName="$fileinputname$.aspx"
                SubType="Designer">TemplatePage.aspx</ProjectItem>
    <ProjectItem ReplaceParameters="true" TargetFileName="$fileinputname$.aspx.cs"
                SubType="">TemplatePage.aspx.cs</ProjectItem>
  </TemplateContent>
</VSTemplate>
```

Visual Studio Template Schema

First thing is the <[default] http://schemas.microsoft.com/developer/vstemplate/2005:ProjectItem>tag which defines what files have to be crated into solution when someone uses this template to create new item into the solution. Along with their TargetName and Actual file name we have added into our ItemTemplate project.

Second is the <[default] http://schemas.microsoft.com/developer/vstemplate/2005:DefautName>where you define the default name it has to show up with in the Add New Item window in Visual Studio with your ItemTemplate.

Rest are the minor things like, the icon file which will represent you ItemTemplate in visual studio, the description, Project type and TemplateId which is a GUID.

Once you have these things in place and when you compile this project and go to your bin/debug folder you should see a .zip file which will contain your .vstemplate,.ico,.aspx and .aspx.cs files required to create new menu item in Visual Studio and create actual files when somebody adds a new file using this ItemTemplate. Once you have this template file ready there are couple of ways you can install it.

One

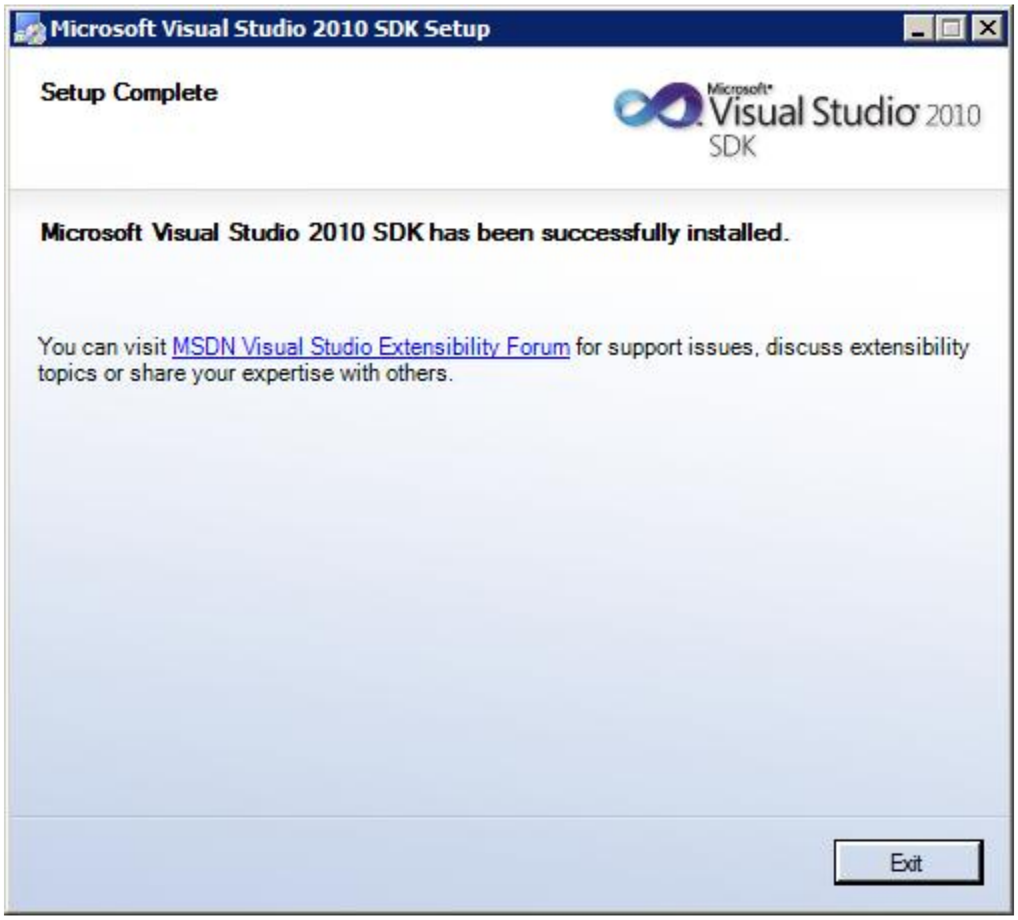
Locate following directory and put your zip file in this directory. C:\Program Files\Microsoft Visual Studio 10\Common7\IDE\ItemTemplates Open Visual Studio Command Prompt and run this command. This is the command for rebuilding Visual Studio Template Cache. I have seen people asking this question frequently on web about “How to rebuild my Visual Studio Template Cache ?”. Following is the command you need to run in order to rebuild VS template cahce. devenv /installvstemplates

Two

We create the VSIX installer package, so that you can easily distribute this VSIX installer and not worry about locating the folder, copying file and rebuilding the Cache. So that is what we are going to do in the next section.

So, in the next article I will walk you through the step by step process of creating a VSIX installer package and adding our Visual Studio ItemTemplate project inside that so that we do not have to manually process these templates. You can just distribute the VSIX package and one can installed template with the help of few mouse clicks.

If possible in this series, I will add a third article that covers how to create your own custom project template. Just to give a brief idea, if you remember the third screen shot in this article, right below your Item template project template you have Project template options as well. If you feel excited enough go ahead and explore this option by your self.


Microsoft Visual Studio 2010 SDK Setup
Microsoft Visual Studio 2010 SDK
Exit

Setup Complete

Microsoft Visual Studio 2010 SDK has been successfully installed.

You can visit MSDN Visual Studio Extensibility Forum for support issues, discuss extensibility topics or share your expertise with others.

Thanks