



Article

Browse Code

Stats

Revisions (2)

Alternatives

Comments & Discussions (21)

Extending Visual Studio Part 3 - Item Templates

By **Dave Kerr**, 14 Apr 2012

★★★★★ 4.93 (25 votes)

[Download ViewModelItemTemplate.zip - 19.5 KB](#)

Extending Visual Studio

This article is part of the series 'Extending Visual Studio'.

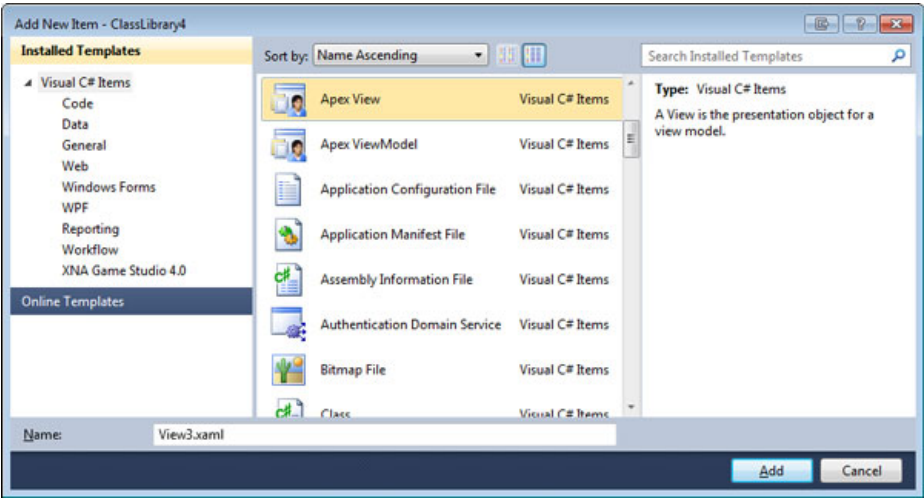
[Part 1 - Creating Code Snippets](#)

[Part 2 - Creating Addins](#)

[Part 3 - Item Templates](#)

Introduction

In part three of my series on Extending Visual Studio, we're going to take a look at Item Templates. Item Templates are what you see when you choose the menu option 'Add New Item':



Here we can see a stack of Item Templates. Some come with Visual Studio, some are my own. It's surprisingly straightforward to add your own custom Item Templates, but the documentation on this is a little scant. In this article we'll put together a band new Item Template for a 'View Model' - a class that implements `INotifyPropertyChanged`.

Hold Your Fire!

This article describes *how* to create Item Templates - don't worry that the template we create isn't enormously functional - it's for illustrative purposes only! Please avoid picking the View Model to death - it's just to give us a better task that 'Hello World'! For those of you who are interested, I *have* created some much more functional Item Templates for MVVM systems, more on that later.

Before We Begin

You will need the Microsoft Visual Studio SDK. I'm using Microsoft Visual Studio 2010 SP1, be aware

About Article

Learn how to extend Visual Studio 2010 by creating custom Item Templates.

Type	Article
Licence	CPOL
First Posted	13 Apr 2012
Views	12,588
Downloads	276
Bookmarked	50 times

[C#](#) [Windows](#)
[Visual-Studio](#) [Dev](#) [+](#)



Top News

The Mathematical Hacker

Get the [Insider News](#) free each morning.

Related Articles

[Creating project item code templates for Visual Studio .NET](#)

[Extending Visual Studio Part 1 - Creating Code Snippets](#)

[Visual Studio 2005 Project And Item Templates](#)

[Customization of Project and Item Templates in Visual Studio](#)

[Code Template Add-In for Visual Studio .NET](#)

[Visual Studio .NET "Add New Item" Custom Template Installer](#)

[Adding Customized Visual Studio Code Generation Templates to the Add new item Dialog Box](#)

[Extending Visual Studio Setup Project](#)

[Write Templates for Visual Studio 2010](#)

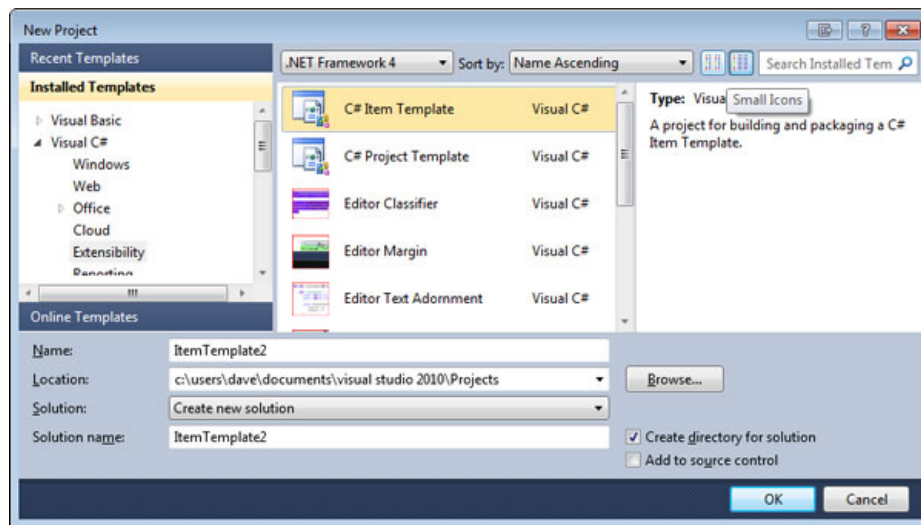
[Customization of Project and Item Templates - Part III](#)

that there are different installers for 2010 and 2010 SP1. Grab the SDK from the links below:

Microsoft Visual Studio 2010 SDK - <http://www.microsoft.com/download/en/details.aspx?id=2680>

Microsoft Visual Studio 2010 SP1 SDK - <http://www.microsoft.com/download/en/details.aspx?id=21835>

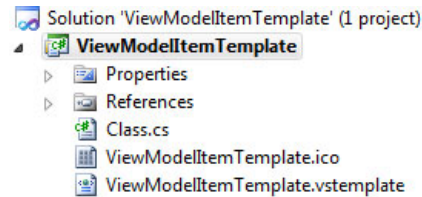
Once you've installed the SDK you'll have a whole host of new Item Templates and Project Templates to play with - filed under 'Visual C# > Extensibility':



To get started, open Visual Studio, choose 'New Project' and select 'C# Item Template' Call the project 'ViewModelItemTemplate'.

Anatomy of an Item Template

Here's a screenshot of how the project will look. There's very little to it.



ViewModelItemTemplate.ico - The icon for the item template.

ViewModelItemTemplate.vstemplate - The template metadata.

Class.cs - The file the template will use as a base.

Firstly let's rename 'Class.cs' to 'ViewModel.cs'. Now we'll open the vstemplate file and update the metadata:

Collapse | Copy Code

```
<?xml version="1.0" encoding="utf-8"?>
<VSTemplate Version="3.0.0" Type="Item"
xmlns="http://schemas.microsoft.com/developer/vstemplate/2005">
  <TemplateData>
    <Name>View Model</Name>
    <Description>Creates a class suitable for use as a View Model, which implements
INotifyPropertyChanged.</Description>
    <Icon>ViewModelItemTemplate.ico</Icon>
    <TemplateID>341358cd-cacb-4f5f-9e5d-be7b3f8a4a97</TemplateID>
    <ProjectType>CSharp</ProjectType>
    <RequiredFrameworkVersion>2.0</RequiredFrameworkVersion>
    <NumberOfParentCategoriesToRollUp>1</NumberOfParentCategoriesToRollUp>
    <DefaultName>ViewModel1.cs</DefaultName>
  </TemplateData>
  <TemplateContent>
    <References>
      <Reference>
        <Assembly>System</Assembly>
      </Reference>
    </References>
    <ProjectItem ReplaceParameters="true">ViewModel1.cs</ProjectItem>
  </TemplateContent>
</VSTemplate>
```

Templates in Visual Studio .NET

Customization of Project and Item Templates - Part II

Visual Studio Add-In for Web Parts Generation

A Heterodox Visual Studio Gadget Template

C# Templates for Visual Studio 2003

Customizing Visual Studio's Code Generation Templates

Add a custom template to Visual Studio .NET 2003

Adding your items to Visual Studio default files

Extending Visual Studio 2008 - What and Why

Building a Web Message Board using Visual Studio 2008 Part II - Posting messages using Microsoft Word

HJAddin for Visual Studio 2005 IDE with code template included

Blow by blow, here's what we're defining with the metadata (the TemplateData item):

- The name displayed to the user is 'View Model'.
- A description for the user, displayed on the right of the screen when they select the template.
- The icon to use.
- A GUID for the template.
- The project type (CSharp, VisualBasic or Web).
- The required framework version is the .NET Framework 2.0.
- The number of parents that will show this item, so if the item is in Category 1 > Category 2 > Category 3 > Category 4 > Item and this is 2, then it'll be shown when you child on Category 2, Category 3 and Category 4.
- The default name - this is what is shown in the file name box.

The TemplateContent is a bit more straightforward in this example. We state that 'System' is an assembly that a reference is required to for this item, and that we're going to take the item 'ViewModel.cs' from this project and replace parameters in it (replacements we'll take a look at in a bit).

Building the Template Itself

Building the template itself is dead easy - you just use a few of the predefined tokens with dollar signs around them. Here's what the ViewModel.cs template looks like:

[Collapse](#) | [Copy Code](#)

```
using System;
using System.Collections.Generic;
#if$ ($targetframeworkversion$ >= 3.5)using System.Linq;
#endifusing System.Text;
using System.ComponentModel;

namespace $rootnamespace$
{
    /// <summary>
    /// The $safeitemrootname$ class.
    /// </summary>
    public class $safeitemrootname$ : INotifyPropertyChanged
    {
        /// <summary>
        /// Occurs when a property value changes.
        /// </summary>
        public event PropertyChangedEventHandler PropertyChanged;

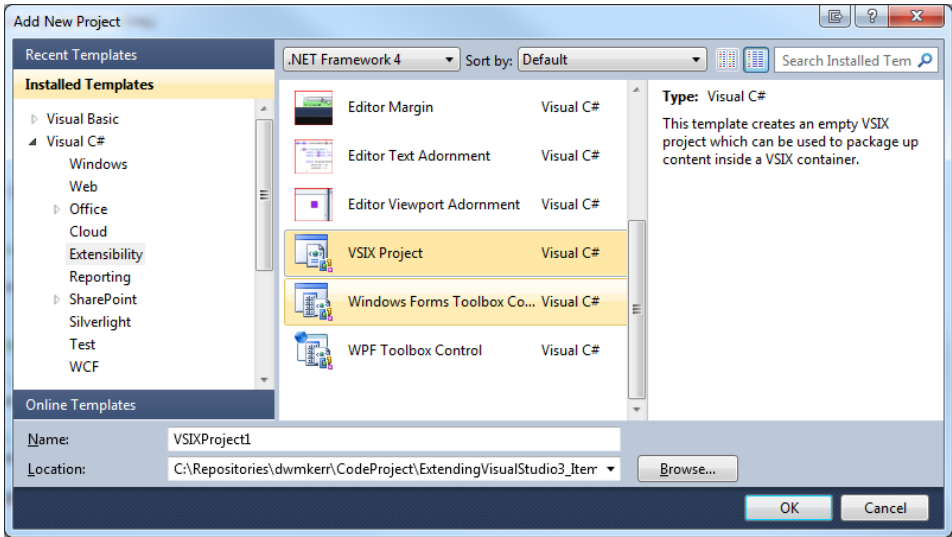
        /// <summary>
        /// Notifies the property changed.
        /// </summary>
        /// <param name="propertyName">Name of the property.</param>
        private void NotifyPropertyChanged(string propertyName)
        {
            // Get and fire the event.
            var theEvent = PropertyChanged;
            if (theEvent != null)
                theEvent(this, new PropertyChangedEventArgs(propertyName));
        }
    }
}
```

Sheesh it's a bit ugly with those odd looking tokens everywhere but you can see what's happening - \$rootnamespace\$ is (you guessed it) the root namespace of the destination project, \$safeitemroot\$ is the file name they've chosen, with things like spaces removed.

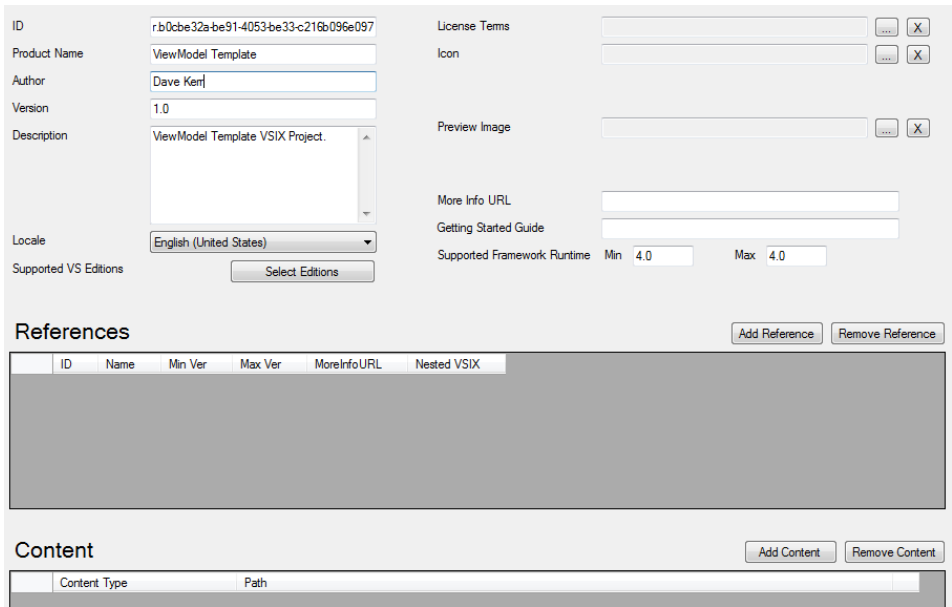
That's all there is to it! We have the item template ready to go. The next thing to do is to create a VSIX package which will contain the template. We can also use this as a way to test and debug it.

Creating the VSIX Package

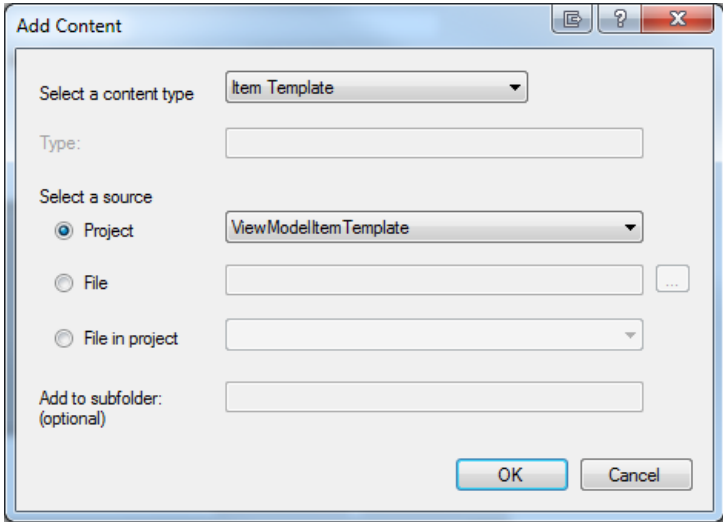
In the solution, add a new project called 'ViewModelTemplateVsix' with the type as VSIX project.



Double click on the 'source.extension.vsixmanifest' file. Here you can edit data about the package itself. We'll set a better title and some descriptions, as below:

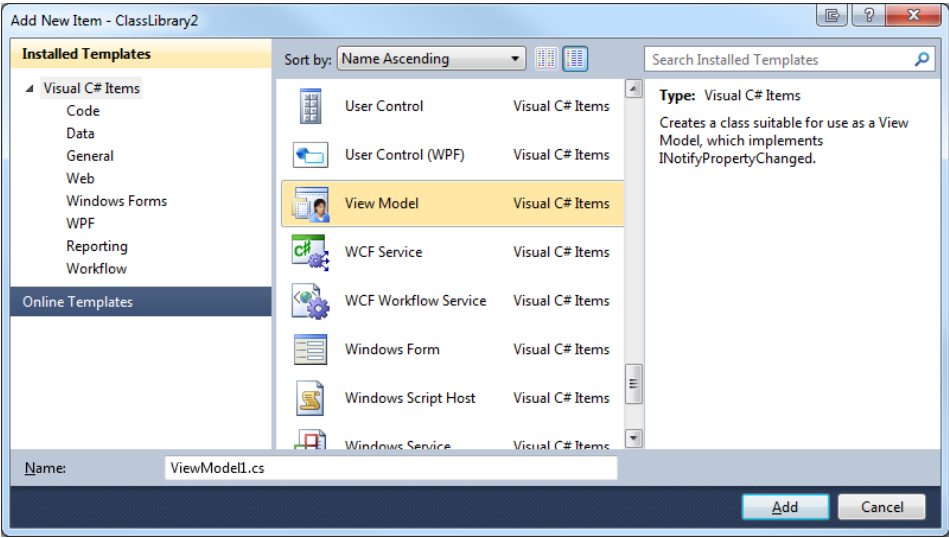


Now we can press 'Add Content' to include our Item Template:



Now that we've done this make sure the VSIX package is the startup project and hit F5. This will start

the Experimental Instance of Visual Studio which is where we can test and debug the extension.
Create a new C# Class Library project and then choose 'Add New Item...' our new ViewModel item template is listed - choose 'Example' as the file name and add the item.



Now this is most impressive - we have got a new class with the correct name that implements INotifyPropertyChanged! Here's the newly generated class:

Collapse | Copy Code

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.ComponentModel;

namespace ClassLibrary2
{
    /// <summary>
    /// The Example class.
    /// </summary>
    public class Example : INotifyPropertyChanged
    {
        /// <summary>
        /// Occurs when a property value changes.
        /// </summary>
        public event PropertyChangedEventHandler PropertyChanged;

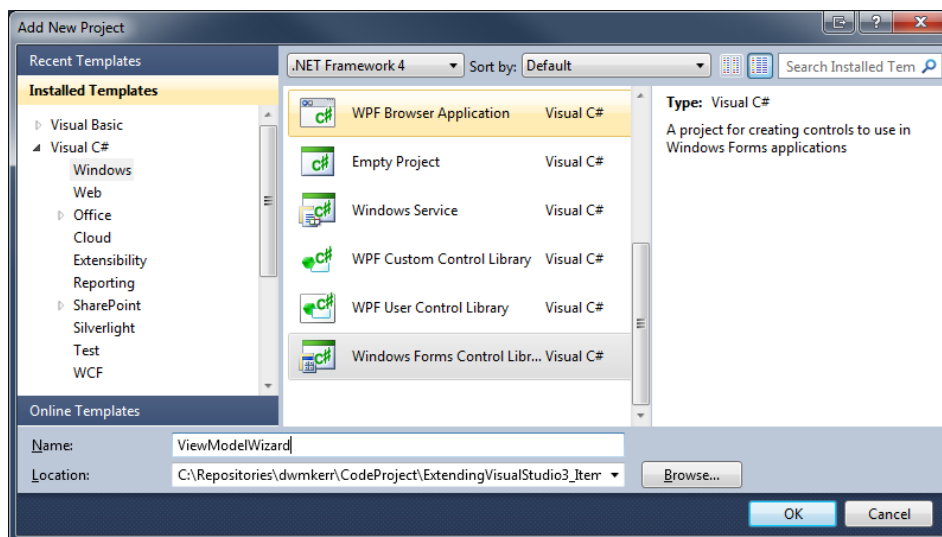
        /// <summary>
        /// Notifies the property changed.
        /// </summary>
        /// <param name="propertyName">Name of the property.</param>
        private void NotifyPropertyChanged(string propertyName)
        {
            // Get and fire the event.
            var theEvent = PropertyChanged;
            if (theEvent != null)
                theEvent(this, new PropertyChangedEventArgs(propertyName));
        }
    }
}
```

Congratulations! At this stage you now know how to build a basic item template. If this is all you need then you're good to go. In the next part of this article we'll look at how we can include a Wizard for the Item Template that'll let us configure some options about how to create it.

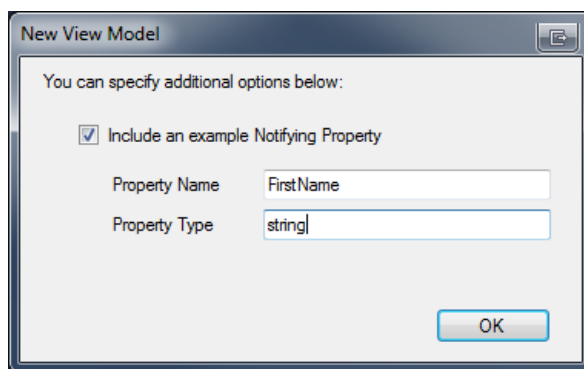
Creating a Wizard for the Template

In many cases, if you're creating an Item Template it's for something a bit more specific and specialised. You may not just want to create a new item with a certain layout, but also allow the user to specify some options. In this case you need to include a Wizard for the Item Template. 'Wizard' is a bit of a misleading name perhaps, all we are in fact going to do is show a single dialog with some options.

I use WPF for my Wizards nowadays but to keep things simple we'll use WinForms. Add a new Windows Forms Control Library to the project, and call it 'ViewModelWizard'.



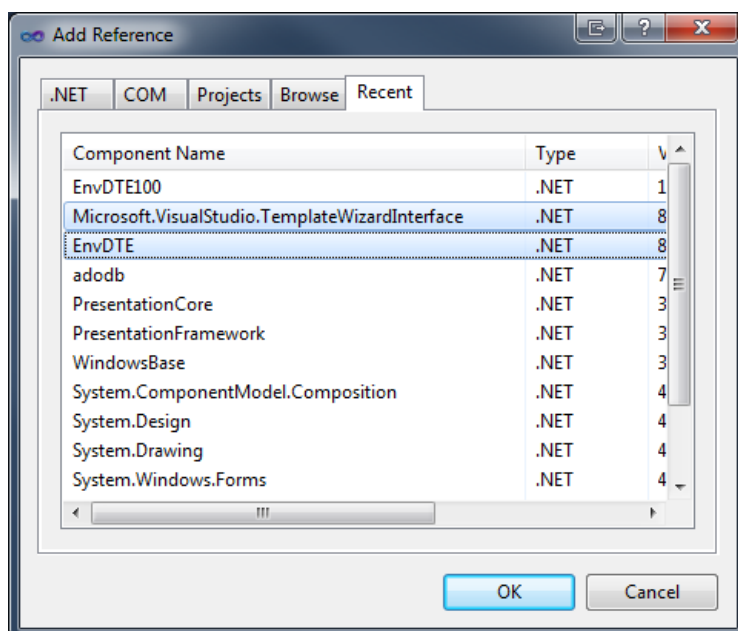
Add a new form to the library and call it NewViewModelForm. We're going to allow the user to optionally include a notifying property with a specified type and name. Lay out the form as below:



I'm not going to go into the details of how the form works, it's very simple - it just exposes the 'Include Example' as a boolean property as well as the Example Name and Example Type as strings. We now need to actually create the Wizard class. Add a new class to the project called 'ViewModelWizard'. At this stage also add references to:

- ENVDT
- Microsoft.VisualStudio.TemplateWizardInterface

As below:



The Wizard class must implement 'IWizard'. Right click on the 'IWizard' line and choose 'Implement Interface'. Your class should now look like this:

[Collapse](#) | [Copy Code](#)

```
public class ViewModelWizard : IWizard
{
    public void BeforeOpeningFile(EnvDTE.ProjectItem projectItem)
    {
        throw new NotImplementedException();
    }

    public void ProjectFinishedGenerating(EnvDTE.Project project)
    {
        throw new NotImplementedException();
    }

    public void ProjectItemFinishedGenerating(EnvDTE.ProjectItem projectItem)
    {
        throw new NotImplementedException();
    }

    public void RunFinished()
    {
        throw new NotImplementedException();
    }

    public void RunStarted(object automationObject,
        Dictionary<string, string> replacementsDictionary, WizardRunKind runKind,
        object[] customParams)
    {
        throw new NotImplementedException();
    }

    public bool ShouldAddProjectItem(string filePath)
    {
        throw new NotImplementedException();
    }
}
```

The only function we care about is 'RunStarted' - in this function we'll show the form and add the values specified to the 'replacementsDictionary'. This will let us use the tokens in our template. Update the class to look like this:

[Collapse](#) | [Copy Code](#)

```
public class ViewModelWizard : IWizard
{
    public void BeforeOpeningFile(EnvDTE.ProjectItem projectItem)
    {
    }

    public void ProjectFinishedGenerating(EnvDTE.Project project)
    {
    }

    public void ProjectItemFinishedGenerating(EnvDTE.ProjectItem projectItem)
    {
    }

    public void RunFinished()
    {
    }

    public void RunStarted(object automationObject,
        Dictionary<string, string> replacementsDictionary, WizardRunKind runKind,
        object[] customParams)
    {
        // Create the form.
        var form = new NewViewModelForm();

        // Show the form.
        form.ShowDialog();

        // Add the options to the replacementsDictionary.
        replacementsDictionary.Add("$IncludeExample$", form.IncludeExample ? "1" :
"0");
        replacementsDictionary.Add("$PropertyName$", form.PropertyName);
        replacementsDictionary.Add("$PropertyType$", form.PropertyType);
    }

    public bool ShouldAddProjectItem(string filePath)
    {
        return true;
    }
}
```

You can see that all we're doing is showing the form and adding the options into the replacements

dictionary. The dictionary can only contain strings - so use "1" or "0" for booleans!

Now we can update the actual template (ViewModel.cs) itself, to use these tokens. We'll set up a conditional block based on \$IncludeExample\$ that will add a property of name \$PropertyName\$ and type \$PropertyType\$ that uses NotifyPropertyChanged. Update ViewModel.cs to the below:

Collapse | Copy Code

```
using System;
using System.Collections.Generic;
#if ($targetframeworkversion$ >= 3.5)using System.Linq;
#endifusing System.Text;
using System.ComponentModel;

namespace $rootnamespace$
{
    /// <summary>
    /// The $safeitemrootname$ class.
    /// </summary>
    public class $safeitemrootname$ : INotifyPropertyChanged
    {
        /// <summary>
        /// Occurs when a property value changes.
        /// </summary>
        public event PropertyChangedEventHandler PropertyChanged;

        /// <summary>
        /// Notifies the property changed.
        /// </summary>
        /// <param name="propertyName">Name of the property.</param>
        private void NotifyPropertyChanged(string propertyName)
        {
            // Get and fire the event.
            var theEvent = PropertyChanged;
            if (theEvent != null)
                theEvent(this, new PropertyChangedEventArgs(propertyName));
        } #if$ ($IncludeExample$ == 1)

        /// <summary>
        /// The value of $PropertyName$.
        /// </summary>
        private $PropertyType$ $PropertyName$Value;

        /// <summary>
        /// Gets or sets $PropertyName$.
        /// </summary>
        public $PropertyType$ $PropertyName$
        {
            get { return $PropertyName$; }
            set
            {
                if( $PropertyName$Value != value)
                {
                    $PropertyName$Value = value;
                    NotifyPropertyChanged($PropertyName$);
                }
            }
        } #endif$
    }
}
```

Before we can test this we need to do three things - sign the Wizard assembly, install the Wizard assembly in the GAC, then update the Item Template to use the Wizard.

To sign the assembly, open the Wizard project settings, choose Signing, check 'Sign the Assembly' and enter a new key - no password is necessary.

To install the assembly into the GAC, add the following to the post build steps of the Wizard project:

32 bit Machine:
"C:\Program Files\Microsoft SDKs\Windows\v7.0A\bin\NETFX 4.0 Tools\gacutil" /if "\$(TargetPath)"
"C:\Program Files\Microsoft SDKs\Windows\v7.0A\bin\sn" -T "\$(TargetPath)"

64 bit Machine:
"C:\Program Files (x86)\Microsoft SDKs\Windows\v7.0A\bin\NETFX 4.0 Tools\gacutil" /if
"\$ (TargetPath)"
"C:\Program Files (x86)\Microsoft SDKs\Windows\v7.0A\bin\sn" -T "\$(TargetPath)"

This just build the project. The post build step will install the wizard to the GAC and show the strong name token:

Collapse | Copy Code

```
Assembly successfully added to the cache
```



```
Microsoft (R) .NET Framework Strong Name Utility Version 3.5.30729.1
Copyright (c) Microsoft Corporation. All rights reserved.

Public key token is e0700af0fcefc5c7
```

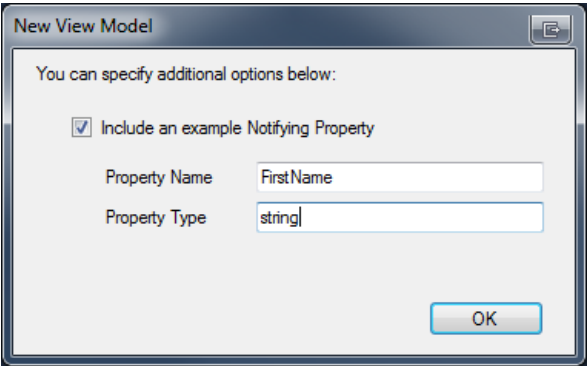
We'll need this token to do the final part of the project. Open the ViewModelItemTemplate.vsitemplate file and after the closing tag of TemplateContent add:

Collapse | Copy Code

```
<!-- Use the appropriate IWizard from the ViewModelWizard assembly. -->
<WizardExtension>
  <Assembly>
    ViewModelWizard, Version=1.0.0.0, Culture=neutral,
    PublicKeyToken=e0700af0fcefc5c7
  </Assembly>
  <FullClassName>ViewModelWizard.ViewModelWizard</FullClassName>
</WizardExtension>
```

Don't forget to use the correct public key token (highlighted in bold).

Now press F5. Add a new View Model to a class library - we get the user interface we created when we add it and can specify an example property:



and we get the result below:

Collapse | Copy Code

```
using System.ComponentModel;

namespace ClassLibrary2
{
    /// <summary>
    /// The Example class.
    /// </summary>
    public class Example : INotifyPropertyChanged
    {
        /// <summary>
        /// Occurs when a property value changes.
        /// </summary>
        public event PropertyChangedEventHandler PropertyChanged;

        /// <summary>
        /// Notifies the property changed.
        /// </summary>
        /// <param name="propertyName">Name of the property.</param>
        private void NotifyPropertyChanged(string propertyName)
        {
            // Get and fire the event.
            var theEvent = PropertyChanged;
            if (theEvent != null)
                theEvent(this, new PropertyChangedEventArgs(propertyName));
        }

        /// <summary>
        /// The value of FirstName.
        /// </summary>
        private string FirstNameValue;

        /// <summary>
        /// Gets or sets FirstName.
        /// </summary>
        public string FirstName
        {
            get { return FirstName; }
            set
            {
                if (FirstNameValue != value)
                {
                    FirstNameValue = value;
                    NotifyPropertyChanged(FirstName);
                }
            }
        }
    }
}
```

```
}
}
}
}
}
```

Final Thoughts

I hope you've enjoyed this article - please feel free to comment on ideas, problems or potentially improvements!

License

This article, along with any associated source code and files, is licensed under [The Code Project Open License \(CPOL\)](#)

About the Author



Dave Kerr
Software Developer 7Layer Solutions

United Kingdom
Member

 Follow on Twitter

Follow my blog at www.dwmkerr.com and find out about my charity at www.childrenshomesnepal.org.

Article Top

Tweet

Sign Up to vote

Poor

Excellent

Vote

Comments and Discussions

You must [Sign In](#) to use this message board.

Search this forum

☐ Profile popups Spacing

Relaxed

 Noise

Medium

 Layout







Normal

 Per page

25

First Prev Next

My vote of 5	John B Oliver	0:16 25 Sep '12
Re: My vote of 5	Dave Kerr	0:55 25 Sep '12
My vote of 5	Mihai MOGA	18:30 12 May '12
Re: My vote of 5	Dave Kerr	0:48 24 Sep '12
My vote of 5	Sergio Andrés Gutiérrez Rojas	18:33 26 Apr '12
Re: My vote of 5	Dave Kerr	0:48 24 Sep '12
From one Visual Studio extender to another	Pete O'Hanlon	21:44 25 Apr '12
My vote of 4	Panos Rontogiannis	21:37 16 Apr '12

 Re: My vote of 4 	 Dave Kerr	22:07 17 Apr '12
 My vote of 5 	 Brian Pendleton	15:36 14 Apr '12
 Re: My vote of 5 	 Dave Kerr	22:07 17 Apr '12
 Nice job 	 Mike Hankey	1:06 14 Apr '12
 Re: Nice job 	 Dave Kerr	2:13 14 Apr '12
 My vote of 5 	 Gareth Barlow (NBNI)	9:31 13 Apr '12
 Re: My vote of 5 	 Dave Kerr	23:56 13 Apr '12
 Another good one there Dave 	 Sacha Barber	2:57 13 Apr '12
 Re: Another good one there Dave 	 Dave Kerr	3:58 13 Apr '12
 Re: Another good one there Dave 	 Gagan1118	6:31 23 Apr '12
 Re: Another good one there Dave 	 Dave Kerr	8:56 23 Apr '12
 Re: Another good one there Dave 	 Pete O'Hanlon	21:40 25 Apr '12
 Re: Another good one there Dave 	 Gagan1118	8:15 7 May '12
Last Visit: 7:58 27 Dec '12 Last Update: 4:31 27 Dec '12 Refresh 1		

 General  News  Suggestion  Question  Bug  Answer  Joke  Rant  Admin

Use Ctrl+Left/Right to switch messages, Ctrl+Up/Down to switch threads, Ctrl+Shift+Left/Right to switch pages.