

# Visual Studio 2010 Tools for SharePoint Development

Steve Fox

SharePoint development has been a bit of a mystery to many developers, who have felt that developing for the platform was cumbersome and out of their reach. The developer community has also been split over what tool set to use. For example, some developers have used a combination of class libraries, manual project folders with XML configuration files, and post-build output events to generate features and solutions for SharePoint. Other developers have used STSDEV, a community tool, or Visual Studio Extensions for Windows SharePoint Services (VSeWSS) to build different applications and solutions and deploy them to SharePoint. In other words, developers could follow numerous paths to deploying features and solution packages to SharePoint. Challenges notwithstanding, the SharePoint developer community has grown to a significant number—roughly 600,000 developers—and continues to grow. Looking forward, Visual Studio 2010 will offer developers a great entry into SharePoint development with the new SharePoint tools that will ship in the box.

SharePoint 2010 is a major step forward as a development platform, not only because of the rich set of features the platform supports, but also because significant investments have been made in the suite of tools designed to make the development process more productive and more accessible to developers of all skill levels. The two core developer tools for SharePoint 2010 are SharePoint Designer 2010 and Visual Studio 2010. (The companion tool set for designers is the Expression suite.) This article provides a first look at SharePoint 2010 development, introducing you to SharePoint tooling in Visual Studio 2010 (including a glimpse at the new project templates) and illustrating how to create and deploy a sample visual Web Part.

## SharePoint Tools in Visual Studio 2010

A number of areas for SharePoint developers in Visual Studio 2010 are worth mentioning. First, you get SharePoint project templates in the box, so you can start right away on solution development. Second, tooling has standardized on the Windows SharePoint Package (WSP) packaging standard, so when you import or deploy a solution to SharePoint, Visual Studio treats it as a solution package. Third, some great deployment and packaging features, such as solution retraction and custom deployment configurations, ship with the SharePoint tools in Visual Studio 2010. And last, the new SharePoint Explorer provides a view into native and custom artifacts (for example, lists and workflows) that exist on your SharePoint server. This is, of course, a short list of the features that represent a major extension of a Visual Studio tool set designed to reach into a community and make it easier for SharePoint developers to get up and running.

Also worth mentioning are a few of the SharePoint 2010 enhancements, which can definitely be used in the context of Visual Studio 2010. For example, the new client object model enables you to access SharePoint objects through a referenced DLL as opposed to Web service calls. (In SharePoint 2007, you access SharePoint list data, for example, by using an ASP.NET Web service.) Also, LINQ for SharePoint brings the power of LINQ to SharePoint, letting you treat lists, for example, as strongly typed objects. Further, Silverlight (especially in combination with the client object model) is supported natively in SharePoint 2010—no more messing around with the web.config to get started with this development. And sandboxed solutions also offer a way to build SharePoint Web Parts and deploy them to a site without needing administrative intervention—that is, you can deploy a Web Part to a SharePoint site and have it run in the context of that site either in an on-premises instance of SharePoint or in the cloud using

# Visual Studio 2010 Tools for SharePoint Development

Steve Fox

the hosted version of SharePoint. Finally, external data lists make interacting with line-of-business systems a read/write process, and while seemingly small, this is a huge leap forward given the tools support that enables you to build line-of-business integrations quickly and efficiently. For each of these innovations in SharePoint 2010, Visual Studio 2010 provides some measure of support, whether through project templates or APIs, for professional developers. If there's a time to pick up SharePoint development, it's now.

## Developing a Visual Web Part Project

One of the most common artifacts that developers build and deploy in SharePoint is the Web Part. This makes sense given that Web Parts are one of the core building blocks for SharePoint. Because SharePoint is built on top of ASP.NET, the Web Part inherits key features from the ASP.NET Web Part architecture.

One of the new project templates in Visual Studio 2010 is the Visual Web Part project template, which enables developers to visually design a Web Part that can be deployed to SharePoint. If you're new to SharePoint, this is a great way to get started building custom applications for SharePoint 2010. The visual Web Part I'll demonstrate has some self-contained code that calculates product costs and lists information in a simple Web Part UI.

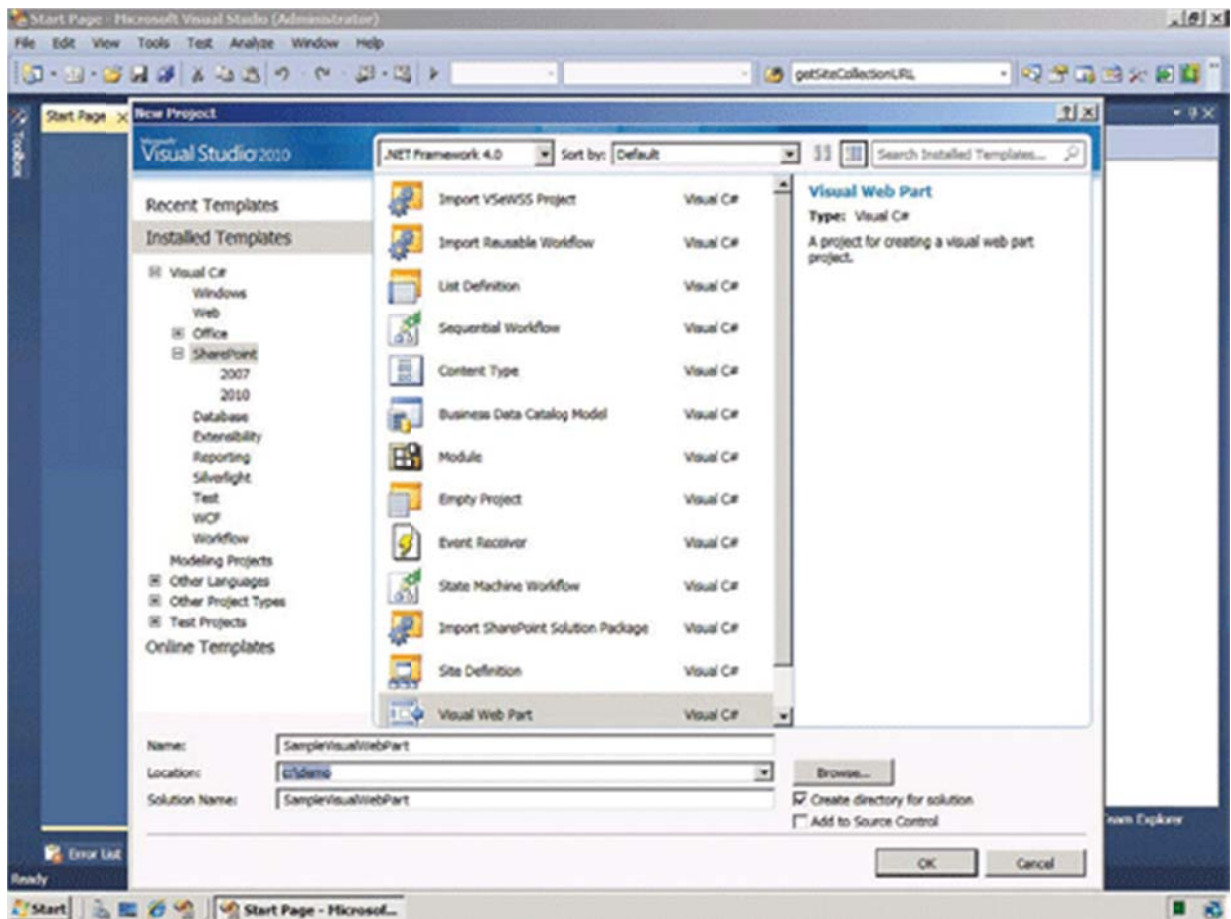
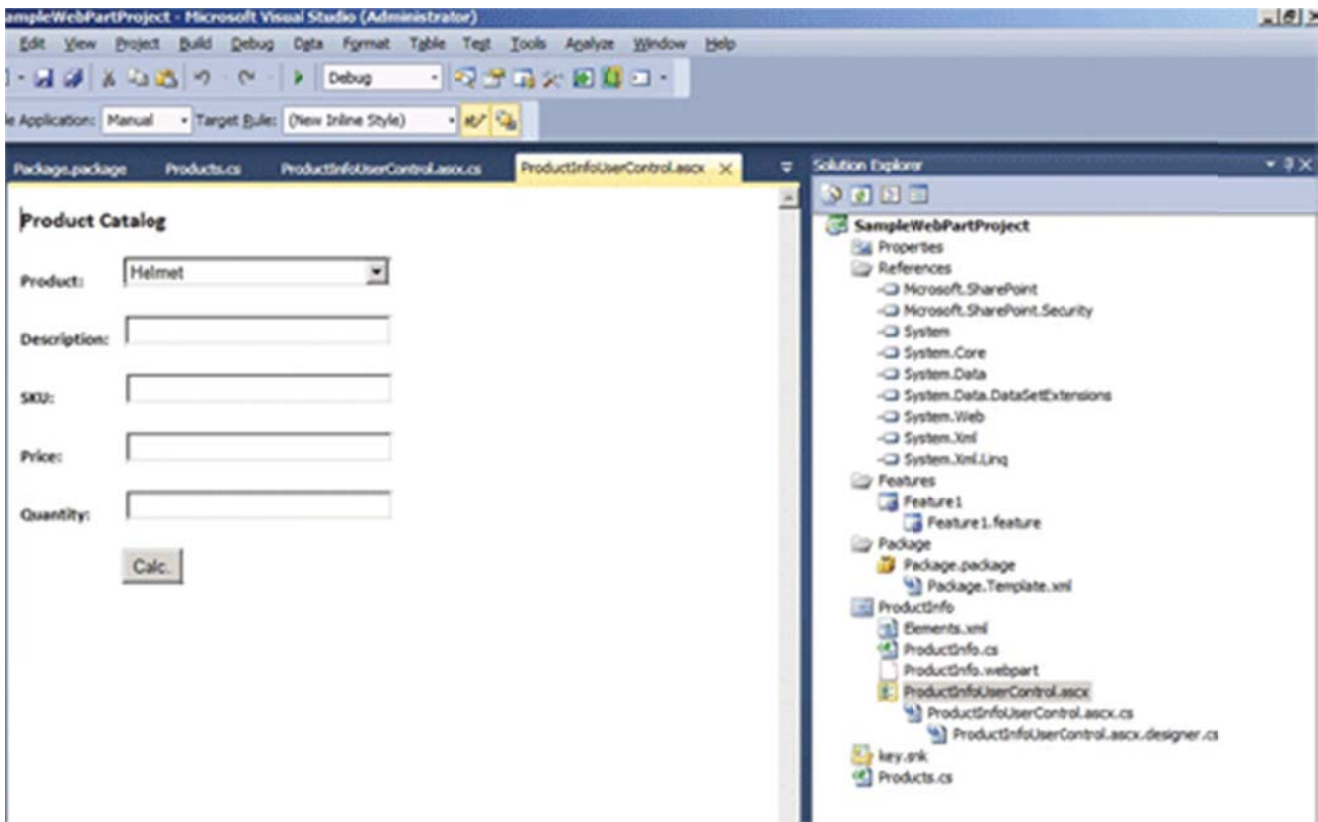


Figure 1: New SharePoint Project Templates

# Visual Studio 2010 Tools for SharePoint Development

Steve Fox



**Figure 2: Designer View for a Visual Web Part**

Make sure you have the beta 2 of Visual Studio 2010 and the beta 2 of SharePoint 2010 installed on 64-bit Windows Server 2008. Open Visual Studio 2010, click File, New Project, and then navigate to the SharePoint node in the Installed Templates section. Figure 1 shows the different types of project templates that are available. For example, the Import VSeWSS Project template provides an upgrade path from your current VSeWSS projects; the workflow templates enable you to create and deploy workflow projects to SharePoint; the Site Definition template provides site-level infrastructure that you can build out and deploy; and the Import SharePoint Solution Package is the template that enables you to import WSPs for redeployment to a local server instance. For this walk-through, select the Visual Web Part project template, provide a name (for example, SampleWebPartProject) and location for your project, and then click OK.

After you create a project, Visual Studio 2010 creates a number of default files. Expand the project nodes in the Solution Explorer to see the files. The key files you'll work with in this article are in the SampleWebPartProject node. Note that the default visual Web Part is called VisualWebPart1. To change this, right-click the VisualWebPart1 node in the Solution Explorer, select Rename and then enter a new name for your Web Part.

Also note in the Solution Explorer the presence of the Features and Package nodes. These are new infrastructural parts to Visual Studio 2010 that package a SharePoint solution using a SharePoint feature. For developers new to SharePoint, a feature organizes your application in a way that SharePoint understands. Features can be deployed to SharePoint at the site or Web level, for example. A feature is structured through a set of XML configuration files, and it also references (depending on the level of trust

# Visual Studio 2010 Tools for SharePoint Development

Steve Fox

for your application) the assembly from the global assembly cache (GAC). Specifically, each feature has its own folder in the SharePoint folder hierarchy, and the configuration files live within that folder and provide the necessary metadata for the feature. The package contains features and other assets and is used when you deploy solutions to SharePoint. The package is also where the assembly deployment location is determined. Visual Studio 2010 introduces a package designer, which makes viewing and managing packages much easier. If you double-click the Package node, the designer opens. The designer provides the ability for you to add and remove features from your deployable package. This designer represents a significant step forward in helping developers shape their SharePoint solutions through the addition of features.

Switch back to the Solution Explorer view, right-click the ProductInfoUserControl.ascx file, and then choose View in Designer. This opens a view in which you can drag and drop controls from the toolbox onto the Web Part designer surface. You'll notice three views: Design, Split, and Code. In this example, I added (by typing) a title and some controls, including text boxes and a button to calculate the cost of the product. I also typed in labels for the controls that were added to the page (see **Figure 2**).

After you complete your visual Web Part layout, you can add event handlers for the button. But before we do that, let's quickly take a look at the source code for the visual Web Part. As you can see from the code excerpt in **Figure 3**, Visual Studio adds some automatic styling to the UI in the form of CSS syntax. You can also see the actual controls (and in the case of the drop-down list, the collection of items) that make up the UI. Note that for brevity, I've removed the directives that are autogenerated and included at the top of the source.

To add event handlers to the Web Part, double-click the button. This takes you to the code behind. It also adds an `onClick` event to the ASCX control design. For example, in **Figure 3** note the `onclick="btnCalcPrice_Click"` event that is included within `btnCalcPrice`. The code behind, which is listed in **Figure 4**, contains some simple code that enables you to calculate the price of the product that is selected in the list box. Key parts of the code are the class-level variables (the doubles), which represent the longhand way I used to calculate the product cost; the List of Products collection (which holds a number of Products objects that are added to the list box); and the `btnCalcPrice_Click` event. When the page is loaded in SharePoint, the code calls the `generateProductList` method, which populates the list box. The `btnCalcPrice_Click` event then calculates the cost of a specific product—depending on what the user selected—and displays the information in the list box in the UI.

**Figure 3: Source Code for SalaryCalcWebPartUserControl.ascx**

```
<style type="text/css">
.style1
{
font-family: Calibri;
font-size: medium;
font-weight: bold;
}
.style2
{
```



# Visual Studio 2010 Tools for SharePoint Development

Steve Fox

create an ASP user control for our Web Part that includes a skin and code behind, the project structure still has the actual Web Part that must surface this control. To do this, Visual Studio creates a string called `_ascxPath`, which represents the path to the ASCX user control that is located within the SharePoint 2010 folder hierarchy. Notice also that in the `CreateChildControls` method, an instance of a control is created and set to the path to the user control (using the `LoadControl` method). It is then added to the `Controls` collection by using the `Add` method. This allows the Web Part to surface the ASP user control within the Web Part in SharePoint. **Figure 5** shows the code.

**Figure 4: Source Code for ProductInfoUserControl.ascx.cs**

```
using System;
using System;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Collections.Generic;
using System.Data;
namespace SampleWebPartProject.ProductInfo
{
public partial class ProductInfoUserControl : UserControl
{
double tax = .11;
double totalCost = 0.0;
List<Products> lstOfProducts = new List<Products>();
protected void Page_Load(object sender, EventArgs e)
{
generateProductList();
}
private void generateProductList()
{
lstOfProducts.Add(new Products()
{ strName = "Helmet", strDescr = "Hockey helmet.", strSKU =
"KLSONHELMT1224", dblPrice = 59.00, intQuantity = 28 });
lstOfProducts.Add(new Products()
{ strName = "Skates", strDescr = "Hockey skates.", strSKU =
"SKATWOKSH0965", dblPrice = 438.00, intQuantity = 88 });
lstOfProducts.Add(new Products()
{ strName = "Stick", strDescr = "Composite hockey stick.",
strSKU = "STIK82910JJKS", dblPrice = 189.99, intQuantity =
35 });
lstOfProducts.Add(new Products()
{ strName = "Elbow Pads", strDescr = "Hockey elbow pads.",
strSKU = "ELBOP563215NN", dblPrice = 34.00, intQuantity =
12 });
lstOfProducts.Add(new Products()
{ strName = "Knee Pads", strDescr = "Hockey knee pads.",
strSKU = "KPDS7827NNJS1", dblPrice = 47.99, intQuantity =
```

# Visual Studio 2010 Tools for SharePoint Development

Steve Fox

```
44 });
}
protected void btnCalcPrice_Click(object sender, EventArgs e)
{
    double dblCost = 0;
    string strPrice = "";
    if (dropdownProducts.SelectedValue == "Helmet")
    {
        dblCost = lstOfProducts[0].dblPrice;
        totalCost = dblCost + (dblCost * tax);
        System.Math.Round(totalCost, 2);
        strPrice = "$" + totalCost.ToString();
        txtbxDescription.Text = lstOfProducts[0].strDescr.
        ToString();
        txtbxSKU.Text = lstOfProducts[0].strSKU.ToString();
        txtbxPrice.Text = strPrice;
        txtbxQuantity.Text = lstOfProducts[0].intQuantity.
        ToString();
    }
    else if (dropdownProducts.SelectedValue == "Skates")
    {
        dblCost = lstOfProducts[1].dblPrice;
        totalCost = dblCost + (dblCost * tax);
        System.Math.Round(totalCost, 2);
        strPrice = "$" + totalCost.ToString();
        txtbxDescription.Text = lstOfProducts[1].strDescr.
        ToString();
        txtbxSKU.Text = lstOfProducts[1].strSKU.ToString();
        txtbxPrice.Text = strPrice;
        txtbxQuantity.Text = lstOfProducts[1].intQuantity.
        ToString();
    }
    else if (dropdownProducts.SelectedValue == "Stick")
    {
        dblCost = lstOfProducts[2].dblPrice;
        totalCost = dblCost + (dblCost * tax);
        System.Math.Round(totalCost, 2);
        strPrice = "$" + totalCost.ToString();
        txtbxDescription.Text = lstOfProducts[2].strDescr.
        ToString();
        txtbxSKU.Text = lstOfProducts[2].strSKU.ToString();
        txtbxPrice.Text = strPrice;
        txtbxQuantity.Text = lstOfProducts[2].intQuantity.
        ToString();
    }
    else if (dropdownProducts.SelectedValue == "Elbow Pads")
    {
```

# Visual Studio 2010 Tools for SharePoint Development

Steve Fox

```
dblCost = lstOfProducts[3].dblPrice;
totalCost = dblCost + (dblCost * tax);
System.Math.Round(totalCost, 2);
strPrice = "$" + totalCost.ToString();
txtbxDescription.Text = lstOfProducts[3].strDescr.
ToString();
txtbxSKU.Text = lstOfProducts[3].strSKU.ToString();
txtbxPrice.Text = strPrice;
txtbxQuantity.Text = lstOfProducts[3].intQuantity.
ToString();
}
else if (dropdownProducts.SelectedValue == "Knee Pads")
{
dblCost = lstOfProducts[4].dblPrice;
totalCost = dblCost + (dblCost * tax);
System.Math.Round(totalCost, 2);
strPrice = "$" + totalCost.ToString();
txtbxDescription.Text = lstOfProducts[4].strDescr.
ToString();
txtbxSKU.Text = lstOfProducts[4].strSKU.ToString();
txtbxPrice.Text = strPrice;
txtbxQuantity.Text = lstOfProducts[4].intQuantity.
ToString();
}
}
}
}
```

**Figure 5: Source Code for ProductInfo.cs**

```
using System;
using System.ComponentModel;
using System.Runtime.InteropServices;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using Microsoft.SharePoint;
using Microsoft.SharePoint.WebControls;
namespace SampleWebPartProject.ProductInfo
{
public class ProductInfo : WebPart
{
private const string _ascxPath =
@"~/CONTROLTEMPLATES/SampleWebPartProject/ProductInfo/" +
@"ProductInfoUserControl.ascx";
public ProductInfo()
{
}
}
```

# Visual Studio 2010 Tools for SharePoint Development

Steve Fox

```
protected override void CreateChildControls()
{
Control control = this.Page.LoadControl(_ascxPath);
Controls.Add(control);
base.CreateChildControls();
}
protected override void Render(HtmlTextWriter writer)
{
base.RenderContents(writer);
}
}
}
```

**Figure 6: ProductInfo.webpart XML File**

```
<?xml version="1.0" encoding="utf-8"?>
<webParts>
<webPart xmlns="http://schemas.microsoft.com/WebPart/v3">
<metaData>
<type name="SampleWebPartProject.ProductInfo.ProductInfo,
SampleWebPartProject, Version=1.0.0.0, Culture=neutral,
PublicKeyToken=db3a9f914308c42a" />
<importErrorMessage>
$Resources:core,ImportErrorMessage;
</importErrorMessage>
</metaData>
<data>
<properties>
<property name="Title" type="string">
Product Info Web Part</property>
<property name="Description" type="string">Provides some
information about hockey products.</property>
</properties>
</data>
</webPart>
</webParts>
```

Now that you've built the visual Web Part, you can deploy it to your SharePoint server. When you created the project, you configured it to be associated with a particular server instance. The implication here is that some programmatic stitch work exists that brings together the code you've just written with the SharePoint server. If you review the files in the Solution Explorer, you'll see a number of XML files that aid in this integration. For example, the Feature.xml file (see the following code) provides a definition of the feature. You can see in the XML that the file references a couple of other XML files that also provide specific information about the Web Part. Here you can see that Elements.xml and ProductInfo.webpart are referenced:

```
<?xml version="1.0" encoding="utf-8"?>
<Feature xmlns="http://schemas.microsoft.com/sharepoint/"
Revised December 17, 2011
```

# Visual Studio 2010 Tools for SharePoint Development

Steve Fox

```
Id="416172c1-cfa7-4d7a-93ba-fe093b037fab"
```

```
ImageUrl="" Scope="Site" Title="SampleWebPartProject Feature1">
```

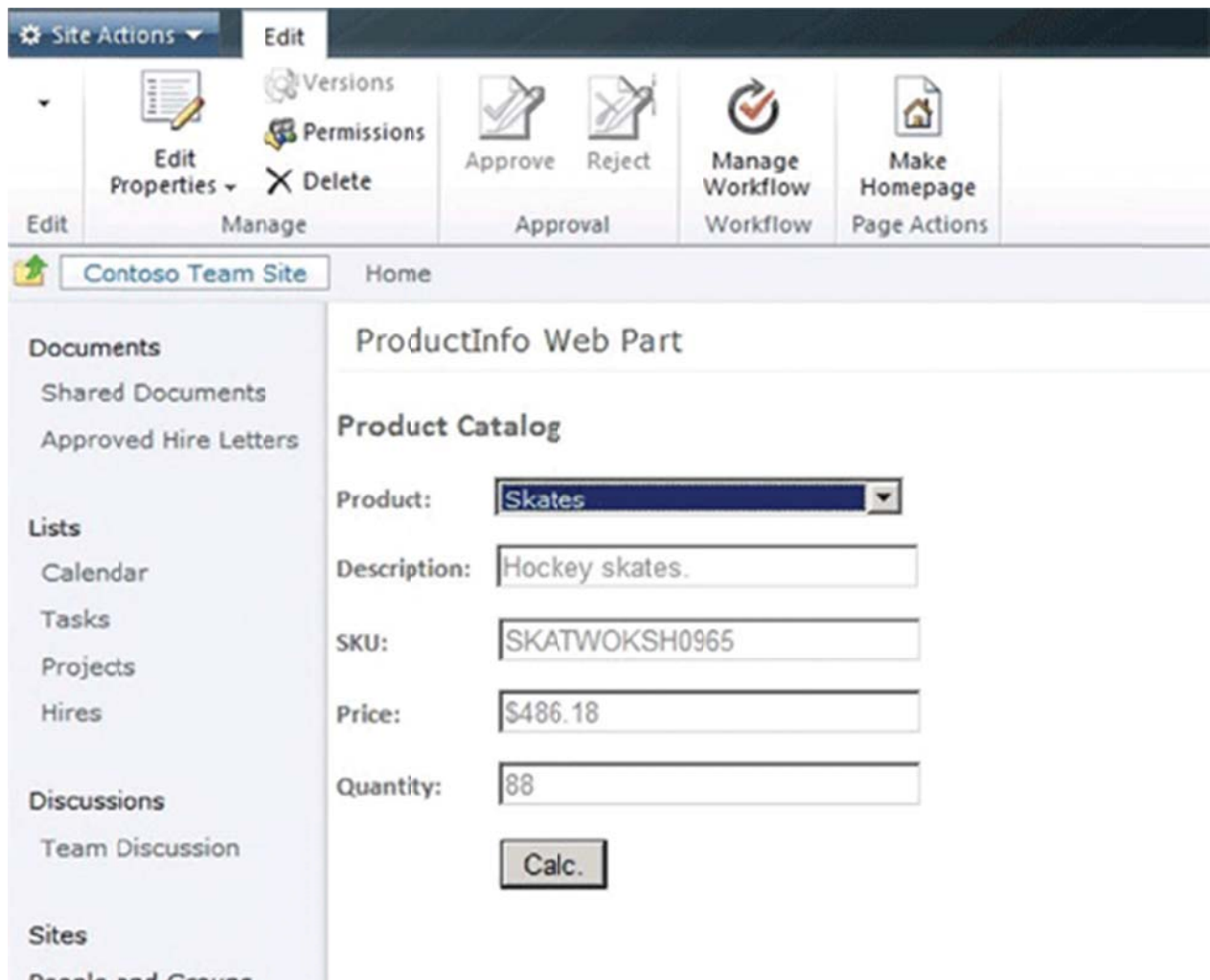
```
<ElementManifests>
```

```
<ElementManifest Location="ProductInfo\Elements.xml" />
```

```
<ElementFile Location="ProductInfo\ProductInfo.webpart" />
```

```
</ElementManifests>
```

Elements.xml provides information about the core assemblies that are included in the feature, and ProductInfo.webpart defines metadata about the Web Part, such as its title and description. For example, **Figure 6** shows the default Title and Description properties. You can update these properties to ensure that the Web Part metadata exposed in the Web Part Gallery is intuitive and meaningful. In the case of this Web Part, you'd likely want to amend the title to be Product Information Web Part and the description to read something like, "Web Part that provides calculated product pricing and information."



**Figure 7: Web Part on Web Part Page**

There are other XML configuration files, and if you're new to SharePoint development, I would encourage you to review each one in the project to better understand their purpose. Now let's move on to deploying the Web Part to your SharePoint server.

# Visual Studio 2010 Tools for SharePoint Development

Steve Fox

## Deploying the Visual Web Part Project

Prior to SharePoint 2010, Stsadm, a command-line-driven administration tool, was commonly used to deploy applications to SharePoint. This need goes away with Visual Studio 2010 (and with the introduction of Window PowerShell, but this is a topic worthy of its own article). Because your project already has a relationship with your SharePoint server, and the association has a defined level of trust, you need only to right-click the project and select Build, make sure the solution builds, and then right-click and select Deploy. Of course, using the F5 key will also work when debugging your SharePoint solutions. By doing this, the debug experience includes steps such as attaching to the appropriate process and resetting IIS.

Once you've successfully deployed the Web Part, you need to open your SharePoint site and create a new Web Part page. If you clicked F5 to debug your application, the Create Web Part page is invoked by default. Otherwise, click View All Site Content, and then click Create. Click the Web Part Page option, and then provide the information requested about that particular Web Part page. For example, provide a name and layout template for the page. After you've entered this information, click Create, and SharePoint creates your Web Part page.

Now you need to add the visual Web Part you created and deployed to the server. To do this, navigate to the Web Part page, click Site Actions, and then click Edit Page. Click the Web Part zone in which you want to place the visual Web Part, click the Insert tab, and then click Web Part on the Insert tab.

After you've done this, SharePoint exposes a number of Web Part categories that you can browse to select a specific Web Part to add to the Web Part zone you selected on the page. Navigate to the Custom category, and in the Web Parts pane, you'll see the visual Web Part you created and deployed. If you followed along with the code in this article, click the ProductInfo Web Part, and then click the Add button.

The Web Part is now added to the Web Part zone on the Web Part page, shown in **Figure 7**. At this point, you can configure the Web Part options through the Tools pane, or you can simply accept the default options and click Stop Editing.

## Engaging SharePoint Development

For SharePoint developers, Visual Studio 2010 provides not only a suite of native tools, but also an amazing opportunity to engage in SharePoint development. I would encourage you to check out these tools. There are some great options for developers who like to have control over their code and for those who like the design experience to build and deploy great solutions in SharePoint.