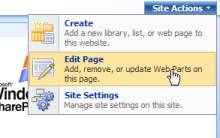
|  |  |
| --- | --- |
| |  | | --- | |  |   **Write Custom WebParts for SharePoint 2007** By [Sahil Malik](http://www.developer.com/feedback.php/http:/www.developer.com/net/asp/article.php/3620316) July 17, 2006  The popularity of SharePoint Portal Server 2003 led Microsoft to tightly integrate the next SharePoint version, Microsoft Office SharePoint Server (MOSS) 2007, with its ASP.NET 2.0 WebPart framework. This tight integration enables the ASP.NET and SharePoint developer to do a number of things that previously weren't possible. For instance, you can plug in your own Membership provider and implement custom authentication. MOSS 2007 also resolves one of SharePoint Portal Server 2003's main problems: It relies on an ISAPI filter to route its requests. Microsoft replaced the ISAPI filter with ASP.NET's HttpHandlers and HttpModules in the new version.  However, rather than attempt to cover all the new possibilities, this article focuses on the process of writing custom WebParts for MOSS 2007.  **Note:** The instructions are written for Microsoft Office SharePoint Server 2007, Beta 2)  **A Tale of Two Base Classes**  You don't need to look very hard to find evidence of MOSS 2007's integration with the ASP.NET 2.0 WebPart framework. The following is a high-level overview of the obvious integration points:   1. A SharePoint Web page is a combination of an ASPX file and a precompiled DLL. If you have customized the page ("unghosted" it), the changes are stored in a database. So, the final ASPX is a combination of the uncustomized ASPX on the file system and the customization changes in the database. You usually find the precompiled class in the Microsoft.SharePoint.ApplicationPages namespace. Various SharePoint pages contain a complicated inheritance structure, but eventually they all end up inheriting from System.Web.UI.Page, which is part of the ASP.NET 2.0 framework. So, in short, all SharePoint ASPX pages are really ASP.NET 2.0 WebPages. 2. SharePoint comes with a number of Web controls—not WebParts, just Web controls, that also inherit from System.Web.UI.Control. 3. The SharePoint WebPartManager is a sealed class called SPWebPartManager. It inherits from System.Web.UI.WebControls.WebParts.WebPartManager, which is the ASP.NET 2.0 WebPartManager. 4. A SharePoint WebPartZone is a public sealed class called Microsoft.SharePoint.WebPartPages.WebPartZone, which inherits from System.Web.UI.WebControls.WebParts.WebPartZone. 5. A SharePoint WebPart inherits from an abstract base class called Microsoft.SharePoint.WebPartPages.WebPart, which inherits from System.Web.UI.WebParts.WebPart.   But wait a minute; doesn't this mean you have two WebPart base classes? Writing a WebPart usually is rather simple. It just needs to inherit from the "WebPart" class. However, you have two WebPart classes here: an ASP.NET 2.0 one at System.Web.UI.WebControls.WebParts.WebPart, and a SharePoint one at Microsoft.SharePoint.WebPartPages.WebPart. So, which one do you use, and when?  For most scenarios, you simply can inherit from the ASP.NET 2.0 WebPart class. The Microsoft.SharePoint.WebPartPages.WebPart class just facilitates backward compatibility with SharePoint 2003. When you move a SharePoint 2003 WebPart over to MOSS 2007, you are simply rebasing your class to now inherit from Microsoft.SharePoint.WebPartPages.WebPart. Because a Microsoft.SharePoint.WebPartPages.WebPart simply inherits from a System.Web.UI.WebControls.WebParts.WebPart, it can live inside a SharePoint page. In addition, you may want to use the SharePoint WebPart when you need cross-page connections (connections between WebParts that are outside of a WebPartZone, a client-side connection, or the data-caching infrastructure). For the most part, however, you should use ASP.NET 2.0 WebParts instead.  **Build and Deploy Your WebPart in MOSS 2007**  For demonstration, put the following WebPart code in a class library:  public class SimpleWebPart : WebPart  {  private string displayText = "Hello World!";  [WebBrowsable(true), Personalizable(true)]  public string DisplayText  {  get { return displayText; }  set { displayText = value; }  }  protected override void Render(System.Web.UI.HtmlTextWriter writer)  {  writer.Write(displayText);  }  }  Next, use the following code snippet to add the AllowPartiallyTrustedCallers attribute (located under the System.Security namespace) to the code's AssemblyInfo.cs file:  [assembly: AllowPartiallyTrustedCallers()]  The reasons for adding this attribute will become clear shortly.  Your WebPart is now ready to use. The next challenge, of course, is deploying it. The following steps will enable you to do just that:   1. Give your assembly a strong name. Just go into the Project Properties and then go to the "Signing" tab (see Figure 1) to do this. Alternatively, you can use the sn.exe command line tool to generate strong names for you.   http://www.developer.com/img/2006/07/WebParts01.jpg  **Figure 1:** Give Your Assembly a Strong Name   1. Find the public key token for your assembly. An easy way to this is to use a standard tool such as [Reflector](http://www.aisto.com/roeder/dotnet/). Once the DLL is compiled and built, drag and drop it in Reflector. It should show you the public key token, along with the details shown in Figure 2.   http://www.developer.com/img/2006/07/WebParts02.jpg  **Figure 2:** Details of Your Public Key Token   1. Locate the SharePoint Web application in which you want to deploy this WebPart. You can do this via the IIS Manager under administrative tools. The default ones would be virtual directories that look like GUIDs under **C:\Inetpub\wwroot\wss\<<GUID>>**. In my case (yours may be different), the directory was **C:\Inetpub\wwwroot\wss\VirtualDirectories\850832ce-d9d4-4a58-b4c9-b06e40870062\\_app\_bin**. 2. Drop the DLL in the \_app\_bin directory in the virtual directory. By doing this, you effectively allow ASP.NET to access the DLL. You still need to take a few more steps for SharePoint to use that assembly.   **Note:** By dropping the WebPart in the bin directory, you are effectively giving it partial trust. If you were writing a plain vanilla ASP.NET 2.0 WebPart, you would need to decorate it with the AllowPartialTrustedCallersAttribute at the assembly level. You also could create a new trust policy for your WebPart (recommended), or raise the trust level in the web.config file (default is WSS\_Minimal). Alternatively, you could just drop the WebPart in the GAC, which again requires strong naming.   1. Under the same virtual directory, find Web.Config and add the following to the SafeControls section: 2. <SafeControl Assembly="SPWebPart, 3. Version=1.0.0.0, Culture=neutral, 4. PublicKeyToken=d81b2da0f7305ec3" 5. Namespace="SPWebPart" 6. TypeName="SimpleWebPart" Safe="True" /> 7. With Web.Config modified, you have to add the WebPart to the gallery. Through your browser, go to the Web application whose virtual directory you have been using. Once on that site, go to Site Actions -> Site Settings -> Modify All Site Settings (see Figure 3).   http://www.developer.com/img/2006/07/WebParts03.jpg  **Figure 3:** Browse to "Modify All Site Settings" Option   1. Under that, click on "Web Parts" under Galleries as shown here:   http://www.developer.com/img/2006/07/WebParts04.jpg   1. Click on "New" in the toolbar, and find the SPWebPart.SimpleWebPart as shown below:   http://www.developer.com/img/2006/07/WebParts05.jpg  Note that SharePoint puts a .webpart extension on the import file being created. If you were to export a WebPart out of an ASP.NET 2.0 page as a .webpart, you easily could use that file to simplify your deployment process.   1. Check the checkbox, go to the top, and click on "Populate Gallery". You should see the SimpleWebPart.webpart file ready to use in your site as shown here:   http://www.developer.com/img/2006/07/WebParts06.jpg  That's it. Your WebPart is now ready to use in a SharePoint ASPX page! |

**Using the WebPart**

Go to the site of your choice. I created a site called "TestSite" using the Blank Template included in SharePoint. Once at that site, click on Site Actions --> Edit Page as shown below:

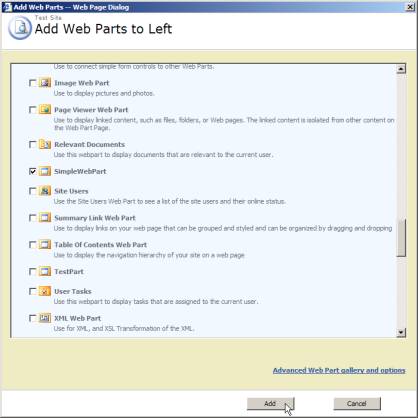


Your page will prompt you to add new WebParts as shown below:

[[](http://www.developer.com/img/2006/07/WebParts08.jpg)  
*Click here for a larger image.*](http://www.developer.com/img/2006/07/WebParts08.jpg)

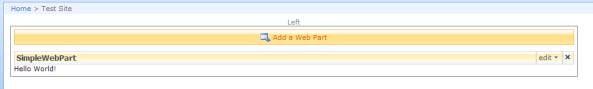
In my case, I have two WebPartZones. Depending upon the template you selected, your exact UI may look different. Click the "Add a WebPart" button on any one of the WebPart zones. You will see the "Add Web Parts" dialog, which simply shows you a list of possible WebParts you could add to your page.

Under Miscellaneous, you should see the SimpleWebPart. Check the checkbox, and click "Add" (see Figure 4).

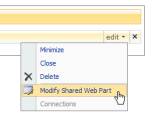


**Figure 4:** Add SimpleWebPart

You now should see the WebPart you wrote running under SharePoint, as shown below:

[[](http://www.developer.com/img/2006/07/WebParts10.jpg)  
*Click here for a larger image.*](http://www.developer.com/img/2006/07/WebParts10.jpg)

Now, you can play with this WebPart a bit. Click the "Edit" menu, and select "Modify Shared Web Part" as shown here:



You will notice that an "EditorZone" pops up. Modify the values as shown below:

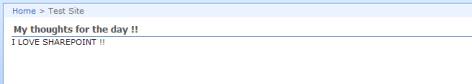


**Figure 5:** Modify the SimpleWebPart Values

Click OK, and then click "Exit Edit Mode" on the top, right-hand side of the page (this may be specific to the template your site is based on) as shown here:

http://www.developer.com/img/2006/07/WebParts13.jpg

You should see your fully customized WebPart running inside MOSS 2007. The following is an action shot of all your hard work:



Congratulations! You have just written a WebPart for MOSS 2007, deployed it, and used it successfully on a SharePoint Web page.

**Microsoft Office and SharePoint Server 2007 Integration**

This tutorial demonstrated how to write a simple WebPart and then host and use it in MOSS 2007. Your real world applications will probably require more complicated WebParts, but to keep the WebPart simple, the demonstration left out some details. For instance, SharePoint by default comes in a very locked-down, secure mode. When writing WebParts for SharePoint, you need to have a good understanding of how code-access security works.