Building WebParts with NetAdvantage for ASP.NET

A Developer's Guide

WebParts are the building blocks of Microsoft Office SharePoint Server pages. By using a simple pattern, you can use NetAdvantage for ASP.NET to target this powerful platform. In this guide, you will learn the key steps necessary to get started in building your own custom web part.

Anthony Lombardo
6/21/2007

Contents

Table of Figures

An Introduction to SharePoint

Microsoft Office SharePoint Server (MOSS) is a powerful web-based portal package that promotes communication and collaboration with-in an organization. MOSS can be used to quickly build web applications, through the Web Parts Framework. Web Parts themselves, are packaged components that when added to a page can provide functional pieces of an application such as data visualization, or data entry.

SharePoint Requirements

Microsoft Office SharePoint Server WebPart pages require a component to derive from a common WebPart base class. The NetAdvantage for ASP.NET controls derive from the core ASP.NET base class, WebControl. Because of this incongruity, NetAdvantage for ASP.NET controls cannot be added directly to a WebParts page.

Getting Started with NetAdvantage for ASP.NET and SharePoint

Wrapping a WebControl into a composite WebPart enables you to bring your typical ASP.NET controls into the world of SharePoint WebParts pages. Composite WebParts, which are typically simple shells which host child WebControls, bring the functionality and reusability of standard WebControls into the modular and personalizable WebPart framework. In this manner, it is quite a simple exercise to create a WebPart shell around one or more NetAdvantage for ASP.NET web controls. Based on your needs, you can make your wrapper as simple or advanced as necessary. On the more advanced end, public properties can be added to the WebPart to enable end user customization. Taking it one step further, you can even add connection points to enable data binding or drill down scenarios. For our example we are going to stick with the basics – a simple shell.

Implementation

To start off, you must determine which base class you want to derive from. In prior versions of Microsoft Office SharePoint, WebParts could only derive from the SharePoint WebPart class, defined in the SharePoint assembly. Because Windows SharePoint Services 3.0 (WSS) is built on top of the .NET Framework 2.0, it is now possible to derive your class from the .NET Framework's WebPart class. Though it is recommended that you use the .NET Framework WebPart base class, there are some valid reasons you may want to consider using the SharePoint WebPart class. Take a look at the following MSDN article for details: http://msdn2.microsoft.com/en-us/library/ms367238.aspx. For our example, we will be using the SharePoint-defined WebPart class, to allow backwards compatibility with prior versions of WSS.

Once you have determined which base class to use, create a new C# or VB.NET class library project, using Visual Studio 2005. You can use the following MSDN topics to get started.

Walkthrough: Creating a Basic Web Part [*System.Web.UI.WebControls.WebParts.WebPart*]
http://msdn2.microsoft.com/en-us/library/ms415817.aspx

Walkthrough: Crating a Basic SharePoint WebPart [*Microsoft.SharePoint.WebPartPages.WebPart*]
http://msdn2.microsoft.com/en-us/library/ms452873.aspx

In the above "Hello World" style articles, overriding the Render method of the control was used as the extensibility model. When creating a composite part, you can leave the brunt of the work to the child controls, including the rendering. Because of this, it is not necessary to override the Render method. Instead, we will rely on the composition model created by the WebPart's Controls collection. By adding our full fledged ASP.NET controls to the WebPart's Controls collection, the controls will automatically render their output with no extra work required as in **Figure 1**.

```
using igGrid = Infragistics.WebUI.UltraWebGrid;
using igGauge = Infragistics.WebUI.UltraWebGauge;
...
private igGrid.UltraWebGrid _Grid;
private igGrid.UltraWebGrid Grid
{
    get
    {
        this.EnsureChildControls();
        return this._Grid;
    }
}

private igGauge.UltraGauge _Gauge;
private igGauge.UltraGauge Gauge
{
    get
    {
        this.EnsureChildControls();
        return this._Gauge;
    }
}
protected override void CreateChildControls()
{
    base.CreateChildControls();
    //create our composites
    this._Gauge = new igGauge.UltraGauge();
    this._Gauge.ID = "Gauge";
    this._Grid = new igGrid.UltraWebGrid("Grid");
    //add composites to controls collection
    //this is the most important piece
    this.Controls.Add(this._Gauge);
    this.Controls.Add(this._Grid);
}
```

Figure 1 - Adding NetAdvantage for ASP.NET to the Controls Collection on a WebPart

In the code above, we are creating a WebGrid and a WebChart, and adding them to the Controls collection of our custom part. CreateChildControls is a method which is automatically called by the underlying control framework, and is guaranteed only to be called once by the EnsureChildControls method. It is recommended that you call EnsureChildControls from inside of the OnInit method of your WebPart class to ensure that any events triggered from child controls are fired properly.

Compiling

To avoid security complications during deployment, it is best to sign your assembly with a strong name key. You can generate a key using sn.exe, a utility included with Visual Studio.

The syntax of the command is as follows.

sn –k [path]

Replace [path] with a file path and name, where the Strong Name Key should be placed.

Use the Project Properties to specify your key file, which will be used to sign the assembly.

Deploying

You will need to copy any Infragistics assemblies used as references, along with your custom assembly, to your Microsoft Office SharePoint Server. It is best to install the assemblies into the GAC to avoid Code Access Security exceptions. Because NetAdvantage for ASP.NET [CLR2] assemblies contain the necessary JavaScript files as embedded resources, creating a virtual directory for script files is not required; however, should you want to use custom images or CSS style sheets, you will want to set up a directory for those resources. You will also need to inform WSS of the directory and mark it for "Wildcard Inclusion." You can use IIS to add a virtual directory, which will contain all of your resources, to your SharePoint site. Once you have added the virtual directory, use the SharePoint Central Administration page to include the directory.

Defining Managed Paths

To define a managed path, first open the SharePoint Central Administration page. Click on the Application Management tab, and under the "SharePoint WebApplication Management" section, click on "Define Managed Paths." Enter the appropriate relative URL for your virtual directory and select the "Wildcard inclusion" option from the "Type:" dropdown box. Finally, click OK.

Using Your WebPart

Now that you have built and deployed your custom WebPart, it is time to add it to a page. WebParts are imported by using a WebPart Definition file (.dwp). Use the same MSDN Walkthrough links from above to create a .dwp file for your WebPart and to register your WebPart as a SafeControl.

Once your part is registered and you have a .dwp file created, use the Import menu item from your SharePoint WebPart page, to import your custom Part. Upon successfully importing the part, you can now add your own custom WebPart to any zone on the page.

An Example WebPart

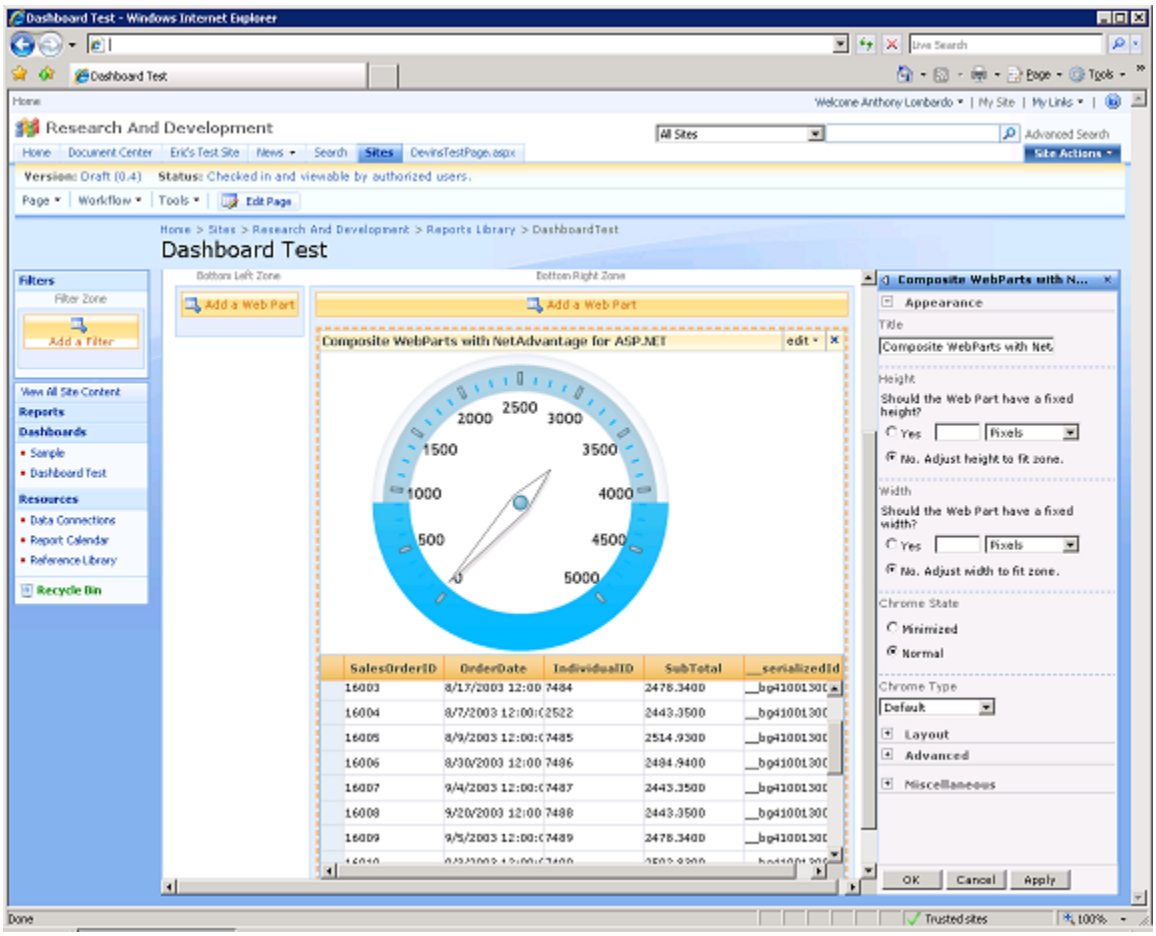

Figure 2 – A WebGrid and WebGauge Bound to AdventureWorks Inside of a Composite WebPart

Conclusion

Although a standard WebControl cannot be added directly to a WebParts page, a simple wrapper can be used to quickly enable this scenario. We have created a WebPart class and added two NetAdvantage for ASP.NET controls to it – namely the WebGrid and the WebGauge. The WebPart shell enables the control to be inserted into a WebParts page, and the underlying WebChart and WebGrid act as the hidden workhorses under the WebPart cover. You can use this same pattern with any NetAdvantage for ASP.NET control, making developing for Microsoft Office SharePoint Server an attractive and quickly attainable goal.