

Making skinned custom controls

Posted on: Friday, January 31, 2003 by Thomas Johansen

Overview

This article will show you how you can easily make your custom controls skinned by deriving from a new control class that we will call SkinControl. SkinControl is an abstract class and derives from Control class, and introduces a new property and a few methods. These methods and the property is what will actually build the skin – one of the methods, Initialize, needs to be overridden by each control you want to be skinned. This must be done because it is in this method where you initialize all the HTML- and web controls that you want to control and that are part of the skin.

The skins will be user controls, in other words, .ASCX files. This makes it a very simple process to update the skins for your controls as you don't need to re-compile anything – all you need to do is to alter the HTML in the skin file and the update will be instantaneous (unless you of course choose to cache the skin files).

Note: the technique seen in this article is based upon the skinning technique as seen in ASP.NET Forums code.

The base class - SkinControl

Now let's get started on the actual code... First up is the base class all your skinned controls will derive from; the abstract class SkinControl.

```
using System;
using System.Web.UI;

namespace MySite.Controls
{
    public abstract class SkinControl : Control, INamingContainer
    {
        // *****
        // VARIABLES
        // *****
        private Control skinControl = null;
        private string skinVirtualPath = null;

        // *****
        // PROPERTIES
        // *****
        public string SkinVirtualPath
        {
            get { return skinVirtualPath; }
            set { skinVirtualPath = value; }
        }

        // *****
        // METHODS
        // *****
        protected override void CreateChildControls()
        {
            Controls.Clear();
            base.CreateChildControls();

            try
            {
                // Try to load the skin
                PreInitialize();

                // Add the skin to the ControlCollection
                Controls.Add( skinControl );

                // Initialize the skin
                Initialize( skinControl );
            }
            catch {} // Prevent application from breaking
        }

        private void PreInitialize()
        {
            skinControl = Page.LoadControl( skinVirtualPath );
        }

        protected abstract void Initialize(Control skinControl);
    }
}
```

As you can see our base class derives from Control class and implements the interface INamingContainer. The reason we implement INamingContainer is that it makes sure our controls are named in a hierarchical, correct manner.

The first thing after declaring the namespace and class is to add 2 variables which we will use to store the actual skin control with and the virtual path to the skin file. After this is taken care of we expose a public property called SkinVirtualPath. This property will allow page developers to get and set the virtual path to the skin file - each control will need to have this set in order for the control to work.

Next on the agenda is the part where we override CreateChildControls method and do a little action - first off we clear the control collection by performing a call to Controls.Clear() method, and then we do a call to the base class' CreateChildControls method.

Once this is over with we need to add our actual skinning methods, namely PreInitialize and Initialize and we need to add the skin control to the control collection. PreInitialize in its current state does nothing more than instantiating skinControl variable by using the method Page.LoadControl. This method takes 1 parameter - a string that point to the virtual path where the user control resides.

The Initialize method is abstract because it needs to be specifically overridden by each control that derives from SkinControl. This is because each skin uses its own set of html- and web controls and thus trying to find controls that might not exist make no sense...

A simple sample control

Ok, now let's test out our base class by creating a simple sample control. The control will do nothing more than display some text, using a label control and a repeater control.

```
using System;
using System.Web.UI.WebControls;

namespace MySite.Controls
{
    public class SampleControl : SkinControl
    {
        // *****
        // VARIABLES
        // *****
        private string skinVirtualPath = "SampleSkin.ascx";

        // *****
        // CONSTRUCTOR
        // *****
        public SampleControl()
        {
            if (SkinVirtualPath == null)
                SkinVirtualPath = skinVirtualPath;
        }

        // *****
        // METHODS
        // *****
        protected override void Initialize(System.Web.UI.Control skinControl)
        {
            Label label1 = (Label) skinControl.FindControl("Label1");
            Repeater repeater1 = (Repeater) skinControl.FindControl("Repeater1");

            label1.Text = "A few primes";
            repeater1.DataSource = new string[] { "1", "3", "5", "7", "11" };
            repeater1.DataBind();
        }
    }
}
```

The first thing we do besides of course declaring the class is to hard-code the default virtual path to the skin file for this specific control. In this case SampleSkin.ascx. Once this is done we add the constructor for this class and inside the constructor we need to check whether a virtual path to a skin file is set, if not we use the default path.

The most important thing comes next, and that is overriding the Initialize method to customize it for the control. In the method we first of all locate all the controls in the skin we want to control – in this case a label control and a repeater control. The retrieval process is done by taking advantage of the FindControl method. For this sample the controls in the skin are named Label1 and Repeater1...

After we have found our controls we can customize the controls like we see fit.

Using the sample control

In order to use the sample control on a page all you need to do is add a @Register directive to the assembly:

```
<%@ Register TagPrefix="MySite" Namespace="MySite.Controls" Assembly="MySite" %>
```

And to add the actual control to the page:

```
<MySite:SampleControl SkinVirtualPath="SampleSkin.ascx" runat="server"/>
```

The property SkinVirtualPath is optional, but I thought I'd just show you how you can easily override the default skin for a control.

And of course you need to create a skin - here is an example:

```
<%@ Control Language="C#" %>
<asp:label id="Label1" cssclass="header" runat="server"/><br/>
<asp:repeater id="Repeater1" runat="server">
  <itemtemplate><%# Container.DataItem %></itemtemplate>
  <separatortemplate>, </separatortemplate>
</asp:repeater>
```

Well that's it basically – I hope you've enjoyed reading this...