



# Custom Controls

Fritz Onion  
DevelopMentor  
onion@develop.com



## Controls and Pages

- Every ASP.NET Page is constituted by a collection of controls
  - **System.Web.UI.Page** is itself the top-level control
  - **System.Web.UI.HTMLControls** define server-side equivalents of HTML elements
  - **System.Web.UI.WebControls** define server-side controls that generate HTML in an intuitive, standard way
  - **System.Web.UI.LiteralControl** is the catch-all control used to represent all literal HTML text on a Page
  - **System.Web.UI.Control** is the base class for all of these controls, and can be extended to build custom controls



Figure 7.1: An .ASPX page prior to compilation

```
<%@ Page language='C#' %>
<html><head>
<script runat='server'>
protected void OnEnterName(object src, EventArgs e)
{
    m_Message.Text =
string.Format("Hi, {0}, how are you?", m_Name.Text);
}
</script></head>
<body>
<form runat='server'>
<h2>Enter your name:
    <asp:TextBox id='m_Name' runat='server' /></h2>
<asp:Button Text='Enter'
            OnClick='OnEnterName' runat='server' />
<br /><asp:Label Id='m_Message' runat='server' />
</form>
</body></html>
```



Figure 7.2: In-memory representation of an .ASPX page

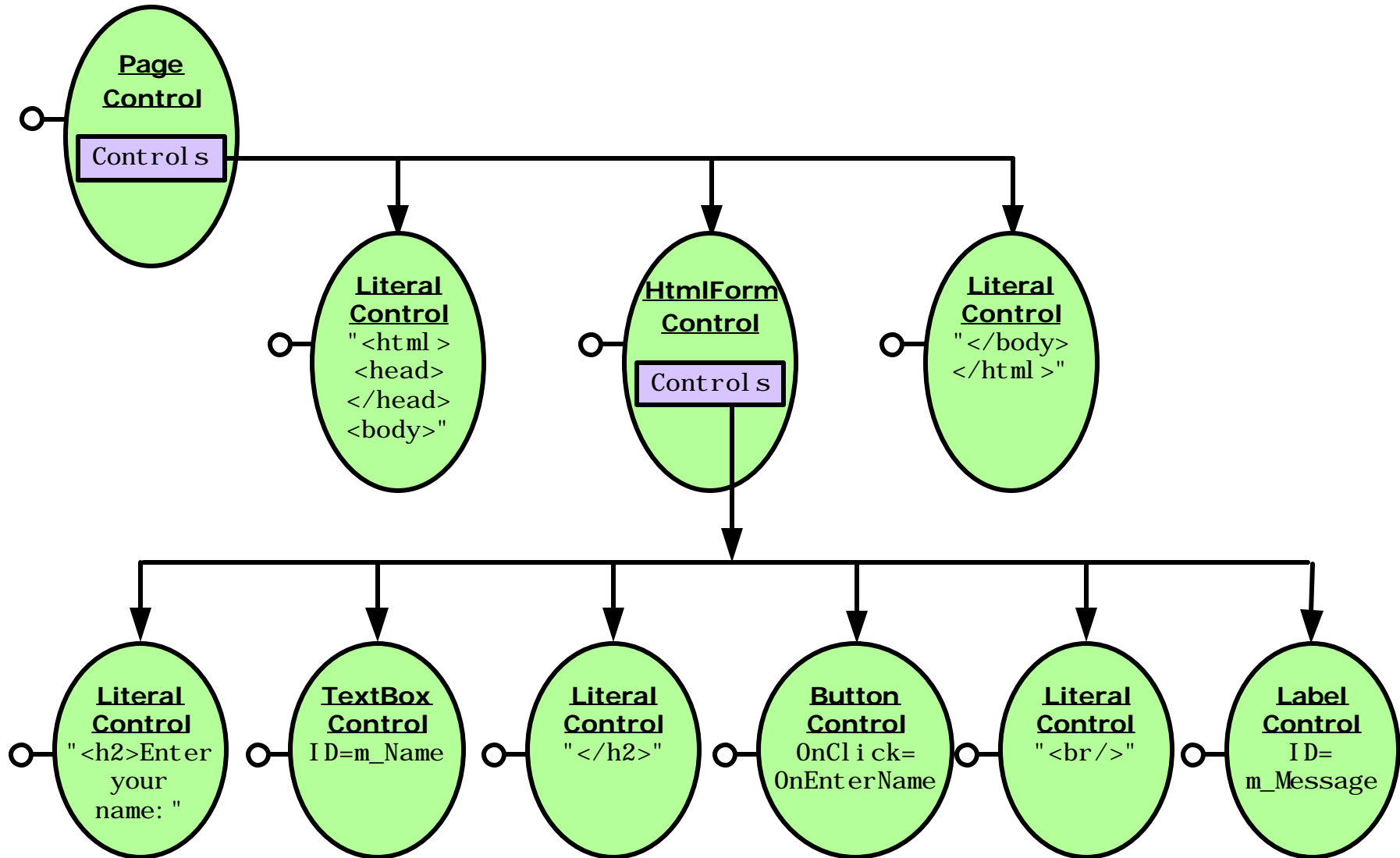




Figure 7.3: Important members of the System.Web.UI.Control class

```
public class Control : IComponent, IParserAccessor, ...
{
    // Called to render a control as HTML
    protected virtual void Render(HtmlTextWriter w);
    // Issued when a control is first created
    public event EventHandler Init;
    // Request to create child controls
    protected virtual void CreateChildControls();
    // Access to containing Page
    public virtual Page Page {get; set;}
    // Access to immediate parent control
    public virtual Control Parent {get;}
    //...
}
```



## Building Custom Controls

- You build new custom server controls by creating a new class derived from Control
  - Create a new class derived from **System.Web.UI.Control** (typically in a distinct namespace)
  - Add any control-specific properties
  - Add any control-specific events
  - Override its **Render()** method to generate HTML using the **HtmlTextWriter** passed in
  - Compile the code into an assembly and deploy



Figure 7.4: A simple custom control

```
namespace DM.AspDotNet
{
    using System;
    using System.Web.UI;
    using System.Web.UI.WebControls;
    using System.ComponentModel;

    public class SimpleControl : Control
    {
        private string m_Prop = "";

        public string Prop
        {
            get { return m_Prop; }
            set { m_Prop = value; }
        }

        protected override void Render(HtmlTextWriter output)
        {
            output.Write("<h1>Property value: ");
            output.Write(m_Prop + "</h1>");
        }
    }
}
```



## Using Custom Controls

- Custom controls may be used from any ASP.NET page with the Register directive
  - `<%@ Register TagPrefix="DM" Namespace="DM.AspDotNet" Assembly="SimpleControl" %>`
  - **TagPrefix** is the shorthand name you would like to use to scope any controls in the **Namespace**
  - **Namespace** specifies the namespace of the control you want to reference
  - **Assembly** is the name of the assembly containing the control



Figure 7.5: Client .aspx page for the SimpleControl custom server control

```
<%@ Page Language="C#" %>
<%@ Register TagPrefix="DM" Namespace="DM.AspDotNet"
           Assembly="SimpleControl" %>

<html>
<body>

    <form runat=server>
        <DM:SimpleControl id="MyCtrl" Prop="Hi!" runat=server />
    </form>

</body>
</html>
```



## Composite Controls

- Custom controls can contain other embedded controls
  - Involves adding a new control to the **Controls** collection
  - Child controls should be created in the **CreateChildControls()** method
  - Use the **LiteralControl** class to intermingle HTML tags with controls
  - Composite controls should implement the **INamingContainer** interface to scope its child controls in a distinct namespace



## Custom Events

- It is possible to define custom events for controls
  - Common for composite controls to propagate events of children to parent
  - May be logical to define completely new events for a control
  - Implemented by defining a public **EventHandler** member and invoking the event when appropriate



## User Controls

- User controls provide a simpler way of defining composite controls
  - Instead of writing a composite control entirely in code, you can take any .aspx page and turn it into a 'user control'
  - User controls are defined in .ascx pages and use the **@Control** directive instead of **@Page**
  - All controls on a user control page are compiled into a separate composite control



## Using User Controls

- Access to a user control is much like any other custom control
  - Clients reference user controls using the **@Register** directive specifying the .ascx file in the optional **Src** attribute
  - User controls can also be created programmatically by calling **LoadControl** with the .ascx filename
  - If loaded programmatically, the type of the user control will be **filename\_ascx**. For example, foo.ascx would generate a type of **foo\_ascx**



## Designer integration

- Custom controls can control their integration with the DevStudio designer by:
  - Providing a custom bitmap for the toolbox (as a 15x15 bmp file compiled as an embedded resource with `<classname>.bmp`)
  - Specifying the exact HTML tag that should be generated when a user drops the control on a form (with the **ToolboxData** attribute)
  - Defining which properties should be displayed for modification in the designer
  - Providing a custom class derived from **ControlDesigner** to customize design-time rendering
  - Defining custom property editors for complex properties



## Properties and appearance

- How properties are displayed in the designer can be influenced by applying attributes, including
  - **Browsable** - whether the property is displayed for editing
  - **DefaultValue** - what value the property should use as default
  - **Category** - the category this property should appear under
  - **Editor** - a specific editor to use to edit this property
  - **MergableProperty** - whether this property can be combined with other properties on other objects
  - **NotifyParentProperty** - whether a parent property should be notified when this property is modified



## Designers

- Classes the control display in design-time can be associated with controls
  - Derived from **System.Web.UI.Design.ControlDesigner**
  - Override **GetDesignTimeHtml ( )** to decide on rendering
  - Associate with control class using **Designer** attribute
  - Control class is implicitly instantiated in designer, and is accessible through the **Component** property of the designer class



## Summary

- A page is constituted by collection of controls
- All controls derive from Control and typically override the Render() method
- Custom controls are referenced using the Register directive from an .aspx page
- Composite controls are controls that contain other controls as children
- User controls are composite controls authored in a .ascx page
- Controls can customize their appearance and behavior in a designer