

ASP+ Basics

Brian Francis, NCR/WROX Press

Introduction

ASP+ introduces a new way of coding that will be familiar to VB or DHTML programmers but a little alien to developers who have only coded in script languages. This presentation will outline this event driven and more structured code model by looking at the ASP+ object model and what Web Controls are and how they can be used.

Be forewarned, this is really cool stuff – revolutionary some may say – so don't be overwhelmed by the enthusiasm of everyone you have met that has played with the Beta.



Brian Francis is the technical evangelist for NCR's Web Kiosk Solutions. From his office in Duluth, Georgia, Brian is responsible for enlightening NCR and their customers in the technologies and tools used for Web Kiosk Applications. He is the author/co-author of numerous Wrox books including Professional ASP 3.0, Beginning ASP 3.0, IE5 Programmer's Reference, Professional IE4 Programming and ASP 3.0 Programmer's Reference.

Agenda

In this presentation, we will be looking at the basics that you need to have to start working with ASP+. This will include how to create your first ASP+ page, and then build on that page using the new ASP+ Web controls. Throughout this track in the conference, you will draw on these skills to build other web applications using ASP+. The concepts that we will examine in detail are:

1. ASP+ Page – A look at the components that make up the ASP+ page, and how it differs from an ASP page.
2. Web Forms – The new display concept in ASP+ is called Web Forms, and we will look at how they can be used to provide interaction with the browser clients.
3. WEB Controls – Within the new Web Forms display model, the developer can take advantage of more advanced, yet easier to use controls known as Web Controls.
4. Validation Controls – The bane of ASP developers – control validation – can now be declaratively handled using Validation Controls.
5. Code-Behind Programming – Moving the ASP+ programming model to a more supportable and maintainable, code-behind forms separate the presentation from the page logic.
6. User Controls – With ASP+, we can now take include files to the next level. User Controls allow us to encapsulate pieces of a page into a reusable object, but not require us to write any extra code.
7. More Cool Stuff – This is just the tip of the iceberg. There is so much cool stuff in ASP+ that we had to dedicate a whole conference track to cover it. So stay tuned...

ASP+ Page

Coding the Old Way

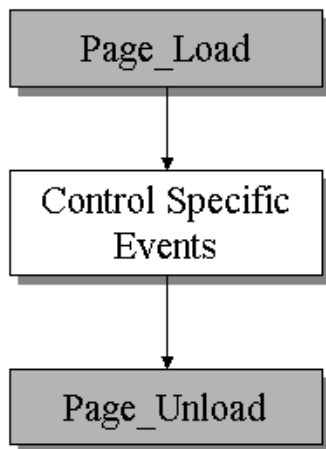
ASP+ introduces a new way of coding, and for those of you who have programmed in event driven languages such as Visual Basic or DHTML scripters, then this model will seem familiar, although it might seem a little alien to programmers who have only coded in script languages in ASP. However, it's extremely simple, providing many advantages, and leads to much more structured code. No longer is the code intermixed with the HTML, and the event driven nature means the code is easily separated into related blocks.

Start at the top, work your way to the bottom

To produce a list of items from a data store in ASP you have to loop through the list of items, manually creating the HTML. When the item is selected, the form is then posted back to the server.

Coding in ASP+

In ASP+ we have a structured set of events, so our code can be placed in these, rather than interspersed with the HTML. The event order is:



Page Events

The `Page_Load` event is always the first event, and is fired every time the page is loaded. After than any control specific events are implemented they are fired, and finally the `Page_Unload` is always fired last.

Web Control Events

When we select an item from a drop-down list and click a submit button, we invoke the postback mechanism. If the button was defined as:

```
<asp:button id="PlaceOrder" Text="Place Order"
  onclick="PlaceOrder_Click"
  runat="server" />
```

The `onclick` property identifies the name of the procedure to be run when the button is clicked. Remember that this is server side event processing, so there is no special client side code. When this button is clicked the form is submitted back to itself, and the defined event handler `PlaceOrder_Click()` is run.

So, when ASP+ is processing a page:

- Server based controls can be accessed from server code.
- The `Page_Load` event is run every time the page is loaded.
- The control event is only run when fired by a server control.

There is, however, one big flaw with our code above. Because the `Page_Load` runs every time the page loads, the code in it will be run even under a postback scenario. This means that work that does not need to be performed every time will be automatically, unless there is a way to catch it.

Page.IsPostBack property

The `Page.IsPostBack` property is designed to counter this problem, as it is a Boolean value that is set to `True` whenever the page has been posted to (for example, when running the event procedure for a control). We can use this property in the `Page_Load` event so that our data access code is only run the first time the page is loaded.

```
If Not Page.IsPostBack Then
```

Web Forms

There are four sets of Web Controls

- Intrinsic Controls, which map to simple HTML elements
- List Controls, to provide data flow across a page
- Rich Controls, to provide rich UI and functionality
- Validation Controls, to provide a variety of data validation

HTML Controls

You might be questioning the actual need for both HTML controls that work server side, as well as ASP+ controls. That's a question that the ASP+ team have asked themselves several times, but they opted for the flexible approach, allowing the user to decide.

As one member of the ASP+ team puts it "it's a lifestyle choice". Use whichever set of controls you feel happier with. The HTML controls keep you closer to the actual content, but they don't offer the advanced functionality (such as consistent naming and so on). On the other hand, the Web controls provide a more consistent programming model, but divorce you a little from the actual content output.

Web Controls

You use Web Controls in the same way as you use HTML controls. The only difference is that they must have the `runat="server"` attribute set. You don't have to do anything special to access this code library, as it's available by default, but you do have to ensure you use the correct tag prefix (or code namespace as it is sometimes called) when using the control. For example, a normal HTML list control is added to the page like this:

```
<select id="ShipMethod">
```

An ASP+ list control has the following form:

```
<asp:ListBox id="ShipMethod">
```

The code namespace is the `asp:` at the beginning of the control name. The reason we have tag prefix, is to ensure that controls with the same names can be uniquely identified. This might not seem a big problem, but when you realize you can author your own controls, you can see what a problem this might be.

Intrinsic Controls

The Intrinsic Controls are designed to provide replacements for the standard set of HTML controls. So, why do we need replacements, especially since the existing HTML controls can be run server-side? Simply put, it's consistency.

Rich Controls

The controls described so far map fairly well onto their HTML equivalents, but just like the standard controls in Visual Basic, there's room for more. In VB we've seen a large market for advanced controls, with rich functionality, and although this has happened in the ASP market, it's not been to such an extent.

Since one of the tenets of ASP+ is its ease of use, you'll not be surprised to learn that an advanced set of controls come with it. There are only two in the beta release, but we expect to see more from Microsoft in the final release, as well as a host from third parties.

Validation Controls

ASP+ is designed to make validation much easier, by carrying out all the arduous parts of the task automatically. It has six **validation controls** that take care of this.

Intrinsic Controls

One of the problems with HTML has been its lack of consistent naming for similar controls. Consider the case of an inputting data, as there are many forms this can take:

```
<input type="radio">  
<input type="checkbox">  
<input type="button">
```

Although these are all input controls, they all behave in a different manner, and should really be different controls, such as:

```
<asp:RadioButton>  
<asp:CheckBox>  
<asp:Button>
```

Text input in HTML is the opposite of this, with two controls performing the same task:

```
<input type="text">  
<textarea>
```

Both of these provide areas for text input – the first only allows a single line, while the second allows multiple lines. A more sensible solution is a single control, where you can specify the number of lines:

```
<asp:TextBox rows="1">  
<asp:TextBox rows="5">
```

Control Transfer

There are four types of controls that allow passing of control back to the server.

- `Button`, which acts as a standard submit button. Use this when you want to perform normal postback to the server.
- `LinkButton`, allows custom processing before postback is actually received on the server. This allows you to have a button and perform server side scripting instead of the normal postback routine. It is the only intrinsic control that relies on client side scripting.
- `ImageButton`, a standard image button - use this when you want to display an image that performs postback to the server.
- `Hyperlink`, which performs the actions of an `A` tag.

Text Entry

The `TextBox` is the only text entry control, since it not only handles both single and multi line entry, but also password entry too. There are three different modes for this control. The first uses the default of single line, whereas the second uses a `MultiLine` mode and specifies the size of the text box. The third specifies the mode to be `Password`, so whatever is typed in is not echoed back to the screen. This sort of form is typical for collecting user information from people visiting your site.

Selections

Selections are catered for by the following controls:

- `CheckBox`, which allows multiple options to be selected.
- `RadioButton`, which allows only a single button to be selected.
- `ListBox`, which provides a list of items, allowing single or multiple selection.
- `DropDownList`, which provides a drop down list allowing only a single selection.

Images

The `Image` control is a replacement for the standard HTML image control, and really doesn't require much in the way of explanation.

Containers

Container controls are controls designed to be parent controls, with other controls within them. You've already seen use of one of them – the `Label` control, which actually renders to a `SPAN` element.

The `Panel` renders to a `DIV` element, and is great for encompassing other controls, and is particularly useful when you need a group of controls to change their visibility.

Rich Controls

asp:AdRotator

The displaying of advertisements on Web pages is one of the few ways of making money on the Web. ASP originally shipped with an Ad Rotator, and this functionality has been encapsulated into a server side control. It is extremely simple to use, and consists of two components:

- An XML file that lists all of the ads to be displayed
- A server control to actually display the ads

asp:Calendar

The other control initially supplied is the `Calendar` control, which provides a rich calendar. Like the `AdRotator` it's extremely simple to use, although it does offer a great deal of customisation.

Two Events

There are two events that are useful to respond to – when the date changes, and when the month changes.

SelectionMode property

By default the calendar only allows selection of single dates. However, the `SelectionMode` property allows us to choose between the following:

- `None`, where no selection of dates is allowed.
- `Day`, where only the day can be selected.
- `DayWeek`, where a day or a whole week can be selected.
- `DayWeekMonth`, where a day, and whole week, or a whole month can be selected.

ASP+ Validation Controls

ASP+ is designed to make validation much easier, by carrying out all the arduous parts of the task automatically. It has six **validation controls** that take care of this:

- The **RequiredFieldValidator** control, which checks that a value has been entered into a control.
- The **CompareValidator** control, which checks that the control contains a specific value, or matches it with the contents of another control.
- The **RangeValidator** control, which checks that the value entered into a control falls within a particular range of values.
- The **RegularExpressionValidator** control, which checks that the value entered matches a specific regular expression.
- The **CustomValidator** control, which passes the value that a user enters into a control to a specified client-side or server-side function for custom validation.
- The **ValidationSummary** control, which collects all the validation errors and places them in a list within the page.

Each of the controls can be linked to one (or more in the case of the **CompareValidator**) HTML or ASP+ form control elements, such as text boxes, list boxes, password boxes, file upload boxes and the **ASP:RadioButtonList** control. You can also link more than one validation control to each control element on a form.

Validation Controls

The standard code to insert the validation controls is:

```
<asp:type_of_validator id="validator_id" runat="server"
  controlToValidate="control_id"
  errorMessage="error_message_for_summary"
  ... other control-specific property settings go here ...
  display="static|dynamic|none">

  ... text to display inline when a validation error occurs ...

</asp:type_of_validator>
```

Each control type has one or more specific properties that define how it should behave. For example the **RegularExpressionValidator** control has a property `validationExpression`, which specifies the regular expression against which the value of the form control is matched.

RequiredFieldValidator

The **RequiredFieldValidator** control is used where a value must be supplied.

```
<input type="text" id="txtName" value="Enter your name" runat="server" />
...
<asp:RequiredFieldValidator id="validator1" runat="server"
  controlToValidate="txtName"
  errorMessage="You must enter your name"
  initialValue="Enter your name"
  display="static">
  *-Error-*
</asp:RequiredFieldValidator>
```

CompareValidator

The **CompareValidator** control, as the name suggests, can be used to compare the value in one form control with the value in a different form control, or with a specific value.

```
<input type="password" id="pwdPassword" runat="server" />
<input type="password" id="pwdConfirm" runat="server" />
...
<asp:CompareValidator id="validator2" runat="server"
  controlToValidate="pwdConfirm"
  errorMessage="The password and confirmation values do not match"
  controlToCompare="pwdPassword"
  type="String"
  operator="Equal"
  display="static">
  *****
</asp:CompareValidator>
```

RangeValidator

The **RangeValidator** control is useful where you want to ensure that a value falls within a specific range.

```
<input type="text" id="txtAge" runat="server" />
...
<asp:RangeValidator id="validAgeRange" runat="server"
  controlToValidate="txtAge"
  type="Integer"
  minimumValue="18"
  maximumValue="90"
  errorMessage="Applications can only be accepted for ages between 18 and
90"
  display="dynamic">
  *
</asp:RangeValidator>
```

RegularExpressionValidator

The **RegularExpressionValidator** control is probably the most versatile of the validation controls. It performs a match between the attached form control's value and a regular expression that you specify.

```
<input type="password" id="pwdPassword" runat="server" />
...
<asp:RegularExpressionValidator id="validator3" runat="server"
  controlToValidate="pwdPassword"
  validationExpression=".*\d.*[A-Z].*|.*[A-Z].*\d.*"
  errorMessage="Your password must contain a number and an uppercase letter"
  display="none">
</asp:RegularExpressionValidator>
```

CustomValidator

There will always be some cases where comparisons, range-checking or regular expressions can't provide the complex validation that you need. In these cases, the **CustomValidator** control can be used. It allows you to create a custom client-side or server-side function (in any supported language) that performs the validation and returns either `True` or `False`.

ValidationSummary

The **ValidationSummary** control collects the values from the `errorMessage` property of each control where the validation test failed, and can present these messages to the user within the page.

```
<asp:ValidationSummary id="validSummary1" runat="server"
  headerText="The following errors were found:"
  showSummary="True"
  displayMode="List" />
```

Code-Behind Programming

One of the problems facing Web programmers and Web page designers is that the increasing complexity required in pages makes it difficult to separate different parts of the development process. Server-side code and HTML are often intermixed in the page, and it's hard to separate them in a sensible and intuitive way so that each member of the development team can work on their own parts without disturbing the work of others.

ASP+ provides two new ways to accomplish this task without having to resort to using the `#include` directive, as is common in earlier versions of ASP. The first of these techniques is to separate the code content of a page completely from the HTML and other text, layout or graphical information by placing it in a separate file.

This technique is called '**code behind**' (or '*code behind forms*'), and takes advantage of the fact that the ASP+ runtime framework compiles each part of the page into a COM+ object and from these creates an execution 'tree'. The file containing the code is just another component of the page, and it exposes the routines it contains like any other COM+ object.

What is actually happening is that we are creating a **base class** that the compiled `.aspx` page will inherit from. And when we declare variables of the correct control type with the correct name (i.e. that matches the name of the class in the 'code behind' template) these variables will be correctly initialized to refer to the control that the page instantiates.

Create a class file (VB or C#)

Finally, your class must inherit from the ASP+ `Page` class so that it can be integrated into the ASP+ page in which it's used. This is done with the `Inherits` statement in Visual Basic:

```
Imports System
Imports System.Web
System.Web.UI
System.Web.UI.WebControls
System.Web.UI.HtmlControls

Public Class class_name
Inherits System.Web.UI.Page
  ...
End Class
```

In C#, you add the system objects by simply defining the inheritance of the `Page` class when you create your new class:

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

```
public class class_name : Page {  
    ...  
}
```

Inherit from the ASP+ Page class

To connect the class file containing the code implementation to your ASP+ page, you add an `Inherits` entry to the `<%@Page...%>` directive, and specify the location of the 'code behind' file:

```
<%@Page Inherits="class_name" Src="path_to_class_file" %>
```

For example, to inherit from a Visual Basic class named `validateClass` that is implemented in a file named `validateclass.vb` in the same directory as the page, we would use:

```
<%@Page Inherits="ValidateClass" Src="validateclass.vb" %>
```

The `Src` attribute is optional, and if omitted ASP+ will expect to find a compiled class file in the `/bin` directory of the application.

User Controls

The second new technique for creating re-usable code, and separating code from content, is through the use of **user controls**.

This brings several advantages – especially when compared to the previous technique with ASP of using the `#include` directive to insert or include the contents of a file that create sections of the page:

- User Controls are self-contained. They provide a separate variable namespace, which that means no methods or variables conflict with existing ones in the hosting page that uses the same name.
- User Controls can be parameterized, which means users can set arguments on them using attributes on the element that inserts the pagelet into the main hosting page.
- User Controls can be used more than once within a hosting page without having to worry about variable and method conflicts (as each one lives inside its own namespace).
- User Controls can be written in a different language from the main hosting page.

Summary

In summary, there are five things that you should take away from this talk:

- ASP+ is COOL! – If you don't think that already, then you haven't been paying attention.
- ASP+ just works! – There are a lot of things going on in the .NET frameworks, but the great thing is that it JUST WORKS!
- Bye bye VBScript – We can now develop using Visual Basic or C#, freeing us from the un-typed mess of VBScript.
- Web Forms are great – When has a programmer ever complained about powerful tools that make them a better programmer, and cause them to write less code?
- Create your own with User Controls – You can extend ASP+ using this control model by creating User Controls.

So taking a line from the Chicago Machine of Mayor Daley – "Download Early, Use Often" – go and get .NET and ASP+ now and start playing with it today.