Introduction to ASP.NET
Ania Wedracki

ASP v ASP.NET
Microsoft’s previous server side scripting technology was Active Server Pages (or ASP).

ASP.NET is an entirely new technology and is not compatible with ASP.

ASP.NET is the major part of the Microsoft .NET Framework.
What is ASP.NET?

ASP.NET is a server side scripting technology that allows scripts embedded in web pages to be executed by a web server.

- ASP.NET makes use of IIS as a web server.
- ASP.NET makes use of the .aspx extension for files.
- ASP.NET files can contain basic HTML markup and scripts like Visual Basic and C#.

Advantages of ASP.NET

- **Separation of Code from HTML**
  In ASP.NET you have the ability to completely separate layout and business logic.

- **Support for Compiled Languages**
  Using a compiled language means that ASP.NET pages do not suffer the performance penalties associated with interpreted code.

- **State Management**
  ASP.NET provides a number of solutions for session and application state management. State information can be kept in memory or in the database and can be shared across web farms. State information can be recovered, even if the server fails or the connection breaks down.
The Microsoft .NET Framework

The .NET framework is an environment for building, deploying and running web applications and web services.

It consists of three main parts...

The Microsoft .NET Framework

Programming languages:

- **C#** (comparable to Java),
- **Visual Basic**, and;
- **J#**.

copyright 2010 © ania wedracki
The Microsoft .NET Framework

Server and client technologies:

- ASP.NET (for the web),
- Windows Forms (for the desktop), and;
- Compact Framework (for mobiles).

The Microsoft .NET Framework

Development environments:

- Visual Studio .NET, and;
- Visual Web Developer.
The Microsoft .NET Framework

Today, we will be using the Microsoft .NET Framework 4.0 with the following:

- C#,
- ASP.NET, and;
- Visual Web Developer 2010 Express.

This is a free application available at http://www.microsoft.com/express/downloads/

Separating Layout and Business Logic

Although you can write C# within an .aspx page, best practice states that you separate your layout from your business logic.

Consequently, when creating a new page in Visual Web Developer, you will notice that between two to three files are created.
Separating Layout and Business Logic

Generally, you can ignore the file ending with `.designer.cs` as this is automatically generated by newer *Microsoft IDEs* to link web controls to business logic.

These files your life as a developer easier.

copyright 2010 © ania wedracki

---

Separating Layout and Business Logic

The `.aspx` file should contain your HTML and server controls only.

The `.aspx.cs` file (for C#, or `.aspx.vb` when using Visual Basic) can contain only your business logic.

copyright 2010 © ania wedracki
HTML Server Controls

Because of this file separation, HTML elements must be declared as server controls before you can manipulate them using C#.

This is done using the attribute `runat="server"`.

```xml
<Default.aspx
  <div id="cMyElement" runat="server">
    This is my original text.
  </div>
</Default.aspx.cs

cMyElement.InnerText = "This is my new text."
```

Web Server Controls

In this way, any HTML element can become a server control.

However, ASP.NET also provides access to a large number of custom server controls performing various functions.

```xml
<asp:Label ID="cMyLabel" AssociatedControlID="cMyTextBox" runat="server">
  My Label</asp:Label>
<asp:TextBox ID="cMyTextBox" runat="server"></asp:TextBox>

<asp:Button ID="cMyButton" Text="My Button" Onclick="ProcessMyTextBox" runat="server" />
```
Web Server Controls

Most web server controls must appear **within** a form marked as a server control. HTML controls can appear anywhere.

This form is always submitted to the page itself. If you specify an action attribute, it is ignored.

An .aspx page can contain only one server form.

```
<form runat="server">
  <asp:Label ID="cMyLabel" AssociatedControlID="cMyTextBox"
runat="server">
    My Label
  </asp:Label>
</form>
```

Copyright 2010 © Ania Wedracki

---

Submitting a Server Form

Submission of a server form is referred to as a **postback**.

A server form is most often submitted by clicking a button.

The click handler within a button specifies the named subroutine to execute from the business logic.

- This is the **OnClick** attribute within ASP.NET buttons, and;
- The **OnServerClick** attribute within HTML buttons.

Copyright 2010 © Ania Wedracki
Submitting a Server Form

```csharp
Default.aspx

<form runat="server">
    <asp:Label ID="cMyLabel" AssociatedControlID="cMyTextBox" runat="server">My Label</asp:Label>
    <asp:TextBox ID="cMyTextBox" runat="server"></asp:TextBox>
</form>

protected void ProcessMyTextBox(object sender, EventArgs args) {
    Response.Write("TextBox value = " + cMyTextBox.Text);
}

Default.aspx.cs

protected void ProcessMyTextBox(object sender, EventArgs args) {
    Response.Write("TextBox value = " + cMyTextBox.Value);
}
```

copyright 2010 © ania wedracki
Page.IsPostBack

You can determine if your page is currently processing a postback using the IsPostBack property.

```csharp
protected void Page_Load(object sender, EventArgs e) {
    if (!Page.IsPostBack) {
        cTextBox.ForeColor = System.Drawing.Color.White;
    }
}
```

State Management

In most web languages, when a form is submitted, all of the values entered by a user are lost unless manually restored by the business logic.

In ASP.NET, this is automatically handled by the state engine.

This behaviour is frequently referred to as the ViewState; however, this is **WRONG**!

copyright 2010 © ania wedracki
The .NET ViewState

Simply put, the .NET ViewState maintains the state of server controls before form submission.

It affects things like colour, font size, position and visibility.

**It does not affect the restoration of form values!**
You can completely disable the ViewState and your application will still remember form values over postback.

---

IntroductionToASPNET

ViewState.aspx
The .NET ViewState

You can store any serializable object within the ViewState of a page or control.

```csharp
protected DateTime PageLoadTime {
    get {
        if (this.ViewState["PageLoadTime"] != null) {
            return (DateTime)ViewState["PageLoadTime"];    
        }
        return DateTime.MinValue;
    }
    set { this.ViewState["PageLoadTime"] = value; }
}
protected void Page_Load(object sender, EventArgs e) {
    if (!Page.IsPostBack) { this.PageLoadTime = DateTime.Now; }
}
```

copyright 2010 © ania wedracki

---

The .NET ViewState

The ViewState is a powerful tool, but it should be used carefully.

The ViewState is actually stored as a hidden form value within every page.

```html
<input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE" value="/wEPdWURMTA5NzX4HHjcxMNRk
```

So the larger your ViewState, the larger each page load and form submission is! Remember to disable the ViewState on controls that do not need it.

```html
<asp:Label ID="cMyLabel" AssociatedControlID="cTextBox"
    EnableViewState="False" runat="server">I maintain my state:</asp:Label><br />
```

copyright 2010 © ania wedracki
Page Lifecycle

Actions such as the OnClick event or the restoration of the ViewState, are run by event handlers.

Different events occur at different stages of a page lifecycle.

Where you place your business logic will determine when it is executed and what you can do.

copyright 2010 © ania wedracki
Event Handlers

Page related event handlers are linked and executed automatically, if they are defined in your business logic.

```csharp
protected void Page_PreInit(object sender, EventArgs e) {
}
protected void Page_Init(object sender, EventArgs e) {
}
protected void Page_Load(object sender, EventArgs e) {
}
protected void Page_PreRender(object sender, EventArgs e) {
}
protected void Page_Render(object sender, EventArgs e) {
}
```

copyright 2010 © ania wedracki

Event Handlers

The **PreInit** or **Init** events do not have access to the properties remembered by the ViewState or any restored postback data.

Similarly, properties set within the PreInit, Init, **PreRender** or **Render** events will not be remembered by the ViewState.

Consequently, your business logic will usually sit within the **Load** event.

copyright 2010 © ania wedracki
Event Handlers

Action related event handlers (such as the **OnClick** event) must be linked within your page.

They are always executed at the **RaisePostBackEvent** phase of the page lifecycle.

Consequently, your Load event will never have access to any values set within an OnClick event. You can access these in PreRender and Render if required.

---

Event Handlers

Event handlers for most action events must take a specific format.

The name of the method can differ, but the access modifier, return type and parameters must remain the same.

```csharp
protected void AnyNameYouLike(object sender, EventArgs args) {
}
```
Validation

When processing forms, it is always a good idea to validate the data provided by the user.

ASP.NET attempts to simplify this task by providing a number standard validators.

When the value entered by a user is incorrect according to the validator, a warning message is displayed using Javascript or server code.

Validation

The various validators provided by ASP.NET are:

- asp:RequiredFieldValidator
- asp:CompareValidator
- asp:RangeValidator
- asp:RegularExpressionValidator
- asp:CustomValidator
Validation

Let’s look at an example...

**IntroductionToASPNET**

Validation.aspx

---

**Validation**

You can group actions and validators into **validation groups**.

When a button is clicked, the page will only show warnings on those validators linked to the same validation group as the button.

```xml
<asp:TextBox ID="cUsername" runat="server"></asp:TextBox>
<asp:RequiredFieldValidator ValidationGroup="vUsername"
    ControlToValidate="cUsername" ErrorMessage="Please enter your username." Display="Dynamic" runat="server"></asp:RequiredFieldValidator>
<asp:Button ID="cMyButton" Text="Submit Me!" OnClick="ProcessUsername"
    ValidationGroup="vUsername" runat="server" />
```
Data Binding

You can use data binding to fill server controls with list items from a database, an XML file or a collection.

Server controls that support data binding include:

- asp:RadioButtonList
- asp:DropDownList
- asp:Repeater

```csharp
if (!Page.IsPostBack) {
    string[] myList = new string[] { "One", "Two", "Three", "Four", "Five" };
    cDropDownList.DataSource = myList;
    cDropDownList.DataBind();
}
```
Data Binding

This results in the following HTML:

```html
<select name="cDropDownList" id="cDropDownList">
  <option value="One">One</option>
  <option value="Two">Two</option>
  <option value="Three">Three</option>
  <option value="Four">Four</option>
  <option value="Five">Five</option>
</select>
```

Data Binding

Once filled, the server control can remember the list items using the ViewState. Consequently, the data needs to be retrieved for binding only when the page is first loaded by the user.

This is especially useful when the cost of retrieving the data from the data source, exceeds the cost of sending the data through the ViewState each time a user submits the form.
Master Pages

Master pages were introduced in ASP.NET 2.0 and are used as templates.

When using a master page, you can avoid having to include a header and footer within all of your pages.

The surrounding template of a page is simply specified with the master page.

copyright 2010 © ania wedracki

Master Pages

Master pages work by allowing you to specify a content place holder.

You can have one or more content place holders within a master page.

The content for these is then specified by the page or nested master page inheriting the master page.

Content place holders can be left empty.

copyright 2010 © ania wedracki
Master Pages

The master page:

```csharp
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/1999/xhtml">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>My Master Page</title>
    <asp:ContentPlaceHolder ID="cHeadContainer" runat="server" />
</head>
<body>
    <asp:ContentPlaceHolder ID="cBodyContainer" runat="server" />
</body>
</html>
```

copyright 2010 © ania wedracki

---

Master Pages

The page:

```csharp
<asp:Content ID="cHead" ContentPlaceHolderID="cHeadContainer" runat="server"></asp:Content>
<asp:Content ID="cBody" ContentPlaceHolderID="cBodyContainer" runat="server"></asp:Content>

<h1>My Master Page</h1>

<p>Hello, and welcome to my page using a template!</p>

</asp:Content>
```

copyright 2010 © ania wedracki
Debugging

Visual Studio and Visual Web Developer allow you to debug your web applications while they are running.

You can insert breakpoints, step through code, watch values and even edit code during runtime.

To use this mode, you need to use the web server built into the IDE.

You can start debugging the page you are currently viewing by pressing **F5**.

copyright 2010 © ania wedracki

---

Debugging

You can also debug your code by writing data to **trace**.

```csharp
Trace.Write("I am writing something to trace.");
Trace.Warn("Hey, I'm warning you!");
```

Although you can display the trace associated with a page each time the page is loaded, this can get unwieldy.

Instead, you can see all trace data written by an application by accessing:

http://localhost/trace.axd

Trace is cleared each time the application is recompiled.

copyright 2010 © ania wedracki
Debugging

Trace Information

<table>
<thead>
<tr>
<th>Category</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>aspx.page</td>
<td>Begin PreInit</td>
</tr>
<tr>
<td>aspx.page</td>
<td>End PreInit</td>
</tr>
<tr>
<td>aspx.page</td>
<td>Begin Init</td>
</tr>
<tr>
<td>aspx.page</td>
<td>End Init</td>
</tr>
<tr>
<td>aspx.page</td>
<td>Begin InitComplete</td>
</tr>
<tr>
<td>aspx.page</td>
<td>End InitComplete</td>
</tr>
<tr>
<td>aspx.page</td>
<td>Begin PreLoad</td>
</tr>
<tr>
<td>aspx.page</td>
<td>End PreLoad</td>
</tr>
<tr>
<td>aspx.page</td>
<td>Begin Load</td>
</tr>
<tr>
<td>aspx.page</td>
<td>End Load</td>
</tr>
<tr>
<td>aspx.page</td>
<td>Begin LoadComplete</td>
</tr>
<tr>
<td>aspx.page</td>
<td>End LoadComplete</td>
</tr>
<tr>
<td>aspx.page</td>
<td>Begin PreRender</td>
</tr>
<tr>
<td>aspx.page</td>
<td>End PreRender</td>
</tr>
<tr>
<td>aspx.page</td>
<td>Begin PreRenderComplete</td>
</tr>
<tr>
<td>aspx.page</td>
<td>End PreRenderComplete</td>
</tr>
</tbody>
</table>

I am writing something to trace.

Hey, I'm warning you!

Debugging

Trace also outputs many other interesting things about the request automatically:

- What server controls are on the page and their ViewState cost,
- What is stored in the session state,
- What is stored in the application state,
- What is stored in cookies,
- What was passed through the form,
- What was passed through the query string, and;
- The server variables.
Configuration

Configuration for ASP.NET web applications is handled by the `web.config` file.

This file is automatically generated and populated when you create a new web application in *Visual Web Developer* or *Visual Studio*.

It specifies application configuration values using XML.

copyright 2010 © ania wedracki

---

Configuration

In addition to standard configuration values, you can specify your own...

```xml
<?xml version="1.0"?>
<configuration>
    <appSettings>
        <add key="MyKey" value="MyKeyValue" />
    </appSettings>

    <connectionStrings>
        <add name="ApplicationServices" connectionString="data source=.\SQL" />
    </connectionStrings>

    <system.web>
        <compilation debug="true" targetFramework="4.0" />
        <trace enabled="true" pageOutput="false" traceMode="SortByTime"/>
    </system.web>
</configuration>
```

copyright 2010 © ania wedracki
Configuration

... which you can then use in your application.

Response.Write(System.Configuration.ConfigurationManager.AppSettings["MyKey"]);
Deployment

Compiling a web application before deployment is standard practice in enterprise systems. Some of the benefits include:

• The application starts faster,
• The application performs better,
• Source code is not available on the web server, and;
• Version management within web farms is easier.

Deployment

Allowing the web server to compile your code also brings some benefits:

• Deploying a change to production requires less overhead,
• Changes can be deployed quickly in an emergency, and;
• Developers do not need to use a Microsoft IDE to make changes.
Deployment

When deploying to a live system, remember to disable the `debug` mode within your configuration file.

Leaving this on will result in significant performance problems.

```xml
<system.web>
  <compilation debug="false" targetFramework="4.0" />
</system.web>
```

copyright 2010 © ania wedracki

---

Deployment

For security, also remember to hide any `errors` generated by your application.

This can be specified in your configuration file.

```xml
<system.web>
  <customErrors mode="RemoteOnly" defaultRedirect="/App_Errors/505.aspx">
    <error statusCode="404" redirect="/App_Errors/404.aspx" />
  </customErrors>
</system.web>
```

copyright 2010 © ania wedracki
Practical Example

We will now put these principles into practice by looking at some practical examples:

• Guestbook
• Web Service