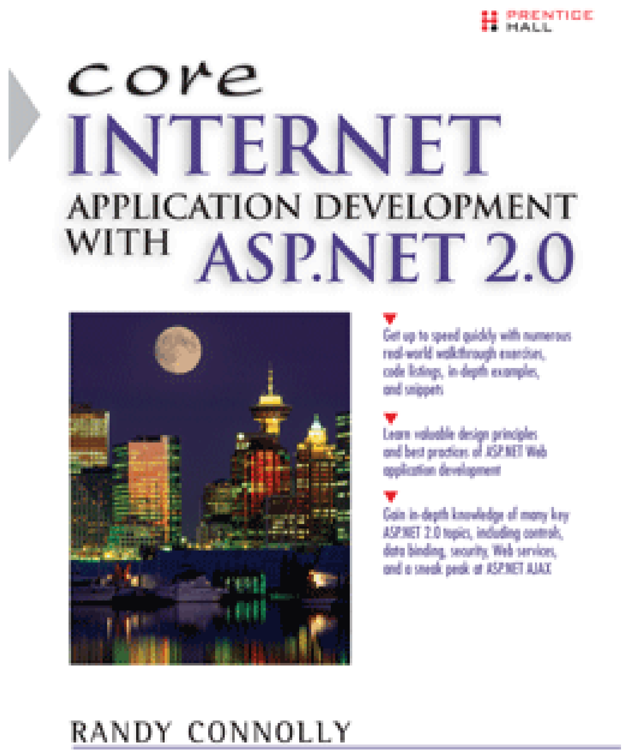
Chapter 2
How ASP.NET Works

And slowly, softly, from above the darkness is unfurled
A wondrous curtain loosened on the windows of the world.
Then suddenly, like magic, …
Ten thousand lights flash out …

Alfred Chandler, "Lights along the Mile,"
The Oxford Book of Australian Verse,
ed. Walter Murdoch, 1918.



PRENTICE HALL

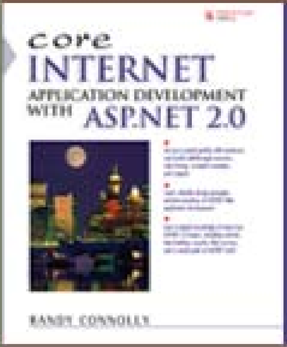
core
INTERNET
APPLICATION DEVELOPMENT
WITH ASP.NET 2.0

- Get up to speed quickly with numerous real-world walkthrough exercises, code listings, in-depth examples, and snippets
- Learn valuable design principles and best practices of ASP.NET Web application development
- Gain in-depth knowledge of many key ASP.NET 2.0 topics, including controls, data binding, security, Web services, and a sneak peek at ASP.NET AJAX

RANDY CONNOLLY

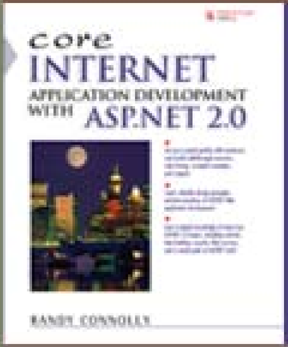
Overview

- ASP.NET Event Model
- ASP.NET Code Compilation
- The `Page` Class
- ASP.NET Application Lifecycle



CORE INTERNET APPLICATION DEVELOPMENT WITH ASP.NET 2.0

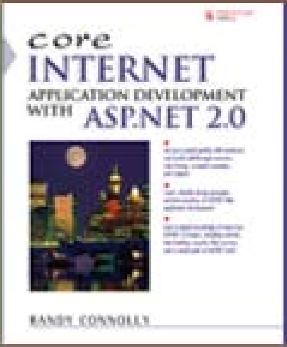
Prentice Hall, 2007
www.randyconnolly.com/core



Event Model

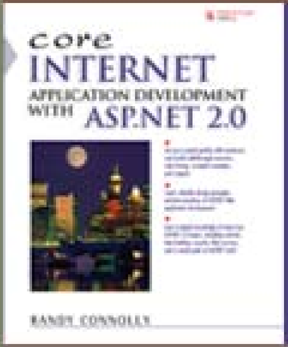
- One of the key features of ASP.NET is that it uses an event-based programming model.
- In the simple Hello World example, we added a small bit of programming to a method named `Page_Load`.
  - This method is an **event handler**.
  - An event handler is a method that determines what actions are performed when an event occurs, such as when the user clicks a button or selects an item from a list.
  - When an event is raised, the handler for that specific event is executed.

Event Handlers



- In the .NET Framework, all event handlers have a specific method signature, that is, a specific return type and parameters.
  - Event handlers are always `void` methods.
  - Event handlers always accept two parameters:
    - an `object` parameter
    - an `EventArgs` parameter
      - (or a subclass of `EventArgs`, such as `CommandEventArgs` or `ImageClickEventArgs`).

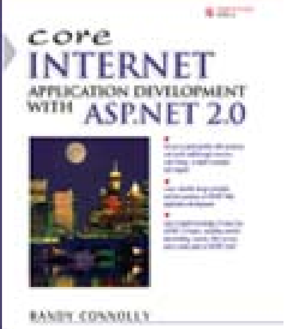
```
protected void Page_Load(object sender, EventArgs e)
{
   …
}
```

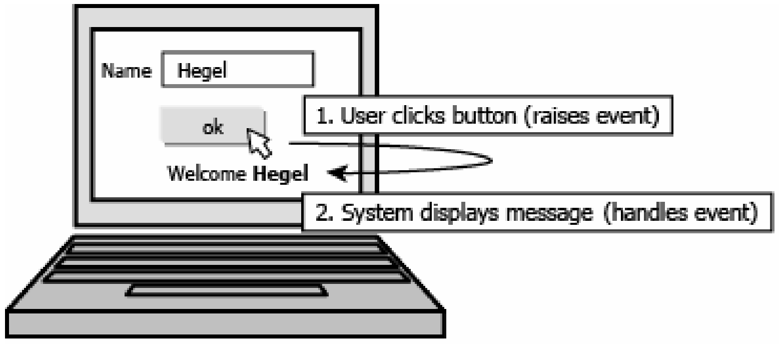


ASP.NET Event System

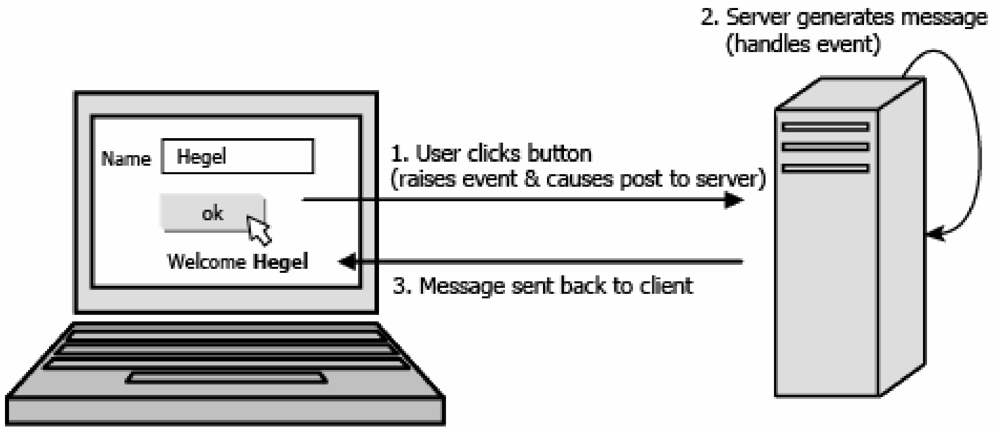
- The event system in ASP.NET operates in a different manner than in a Windows application or from the event system in browser-based Javascript.
  - In a Windows application, for instance, events are raised and handled on the same processor.
  - In contrast, ASP.NET events are raised on the client (the browser) but transmitted to and handled on the server.
    - Since its event handling requires a round-trip to the server, ASP.NET offers a smaller set of events in comparison to a totally client-based event system.

ASP.NET Event System



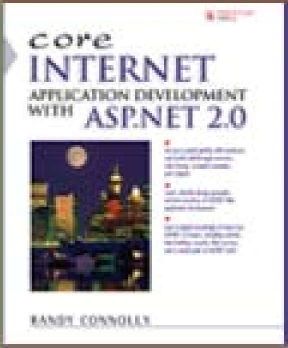

Name
Hegel
ok
Welcome Hegel
1. User clicks button (raises event)
2. System displays message (handles event)

Client-based event system


2. Server generates message (handles event)
Name
Hegel
ok
Welcome Hegel
1. User clicks button (raises event & causes post to server)
3. Message sent back to client

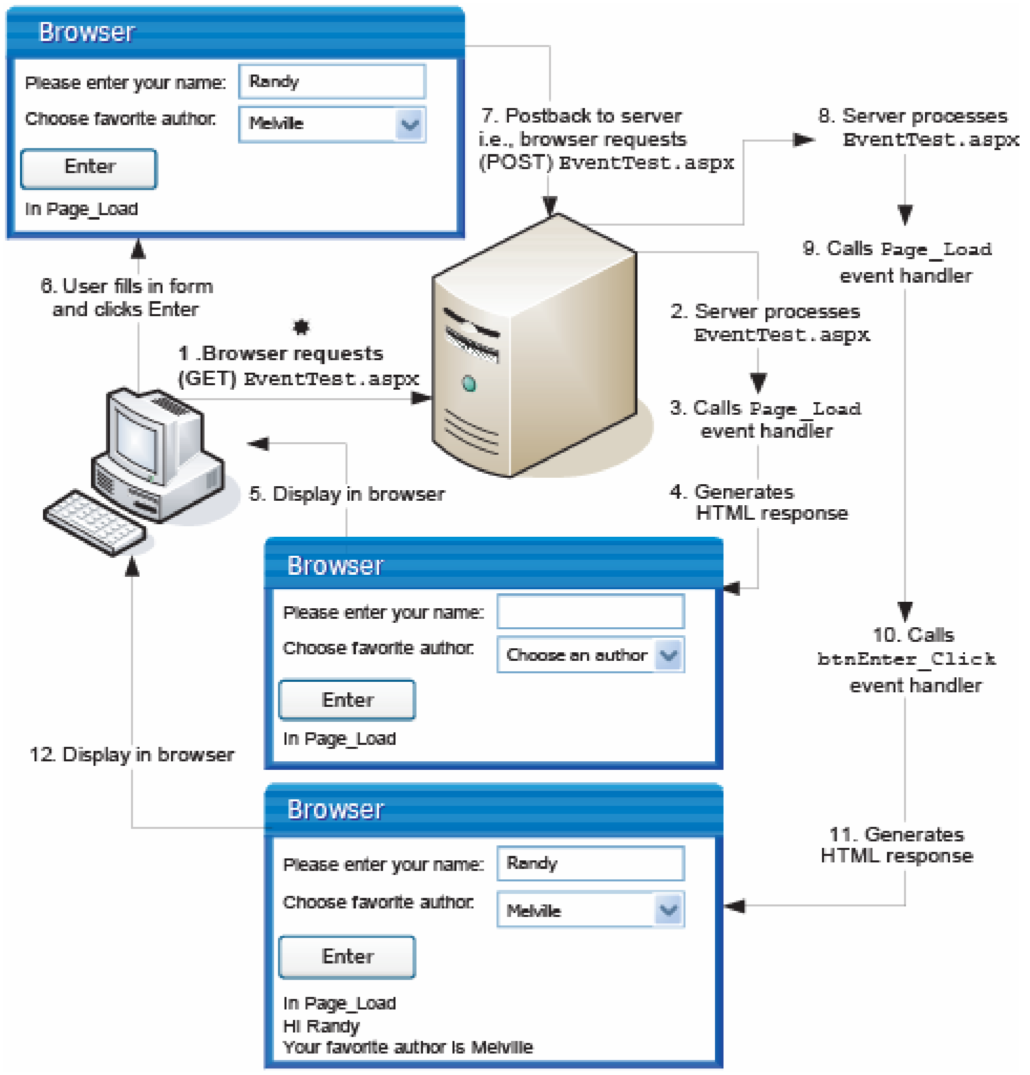
ASP.NET event system

Postback



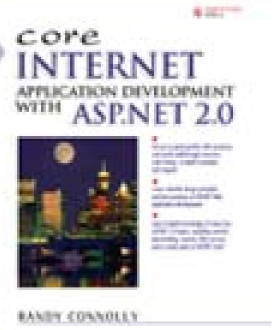
- **Postback** is the process by which the browser posts information back to itself.
  - That is, posts information back to the server by requesting the same page.
- Postback in ASP.NET only occurs within web forms (i.e., within a form element with runat=server), and only server controls post back information to the server.
- Each cycle in which information is displayed then posted back to the server, and then redisplayed again, is sometimes also called a **round trip**.

Postback

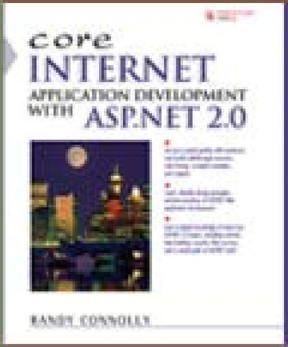

Browser
Please enter your name: Randy
Choose favorite author: Melville
Enter
In Page_Load
7. Postback to server i.e., browser requests (POST) EventTest.aspx
8. Server processes EventTest.aspx
9. Calls Page_Load event handler
6. User fills in form and clicks Enter
1. Browser requests (GET) EventTest.aspx
2. Server processes EventTest.aspx
3. Calls Page_Load event handler
4. Generates HTML response
5. Display in browser
Browser
Please enter your name:
Choose favorite author: Choose an author
Enter
In Page_Load
10. Calls btnEnter_Click event handler
12. Display in browser
Browser
Please enter your name: Randy
Choose favorite author: Melville
Enter
In Page_Load
Hi Randy
Your favorite author is Melville
11. Generates HTML response

CORE INTERNET APPLICATION DEVELOPMENT WITH ASP.NET 2.0

Prentice Hall, 2007
www.randyconnolly.com/core



Event Types

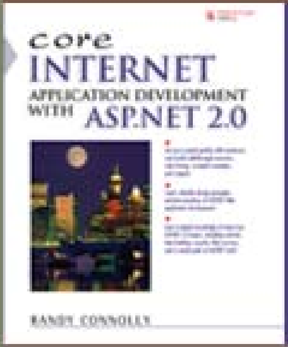


- Two types
  - Page events
    - Always triggered and always in a certain specific order (see Page Lifecycle)
  - Control events
    - Associated with particular controls and only triggered in certain circumstances.

View State

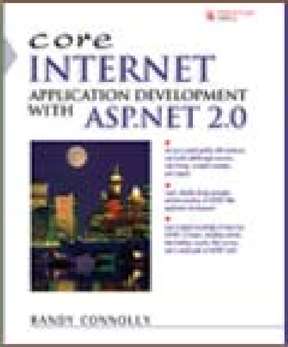
- View state is one of the most important features of ASP.NET.
- It is a specially encoded string that is used to retain page and form information between requests and is sent to the browser within a hidden HTML `<input>` element.
- All page elements not posted back via the standard HTTP POST mechanism are stored within this string.

```
<input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
   value="/wEPDwUJODExMDE5NzY5D2QWAgIDD2QWAgIBDw8WAh4EVGV4dAUKMDgvMDE
     vMjAwNmRkZDZPhFHJER4chf3nmlgfL+uq4W58" />
```



View State

- View state is a mechanism for preserving display **state** within web forms.
- Recall that HTTP is by nature **stateless**.
  - This means that after the server responds to a request, it no longer preserves any data used for that request.
- Nonetheless, web applications very frequently need to retain state on a page between requests.

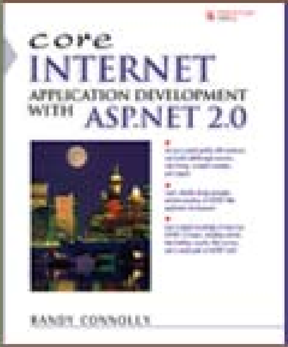


CORE INTERNET APPLICATION DEVELOPMENT WITH ASP.NET 2.0

Prentice Hall, 2007
www.randyconnolly.com/core

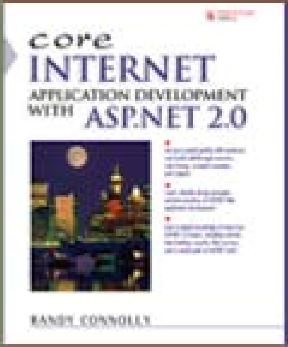
View State

- View state is generated once all the page code has executed but before the response is rendered.
- The value of each web server control on the page is serialized into text as a number of Base64-encoded triplets, one of which contains a name-value pair.
- This view state string is then output to the browser as a hidden `<input>` element named `"__VIEWSTATE"`.



View State

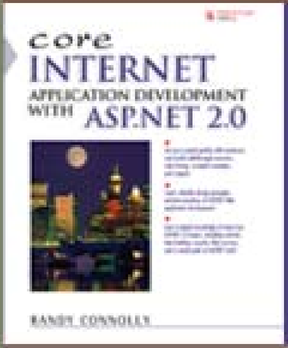
- When the form is posted back, ASP.NET receives the view state (since it was contained in a form element), deserializes this information and restores the state of all the controls prior to the post.
- ASP.NET updates the state of the controls based on the data that has just been posted back, and then calls the usual page and control event handlers.



CORE INTERNET APPLICATION DEVELOPMENT WITH ASP.NET 2.0

Prentice Hall, 2007
www.randyconnolly.com/core

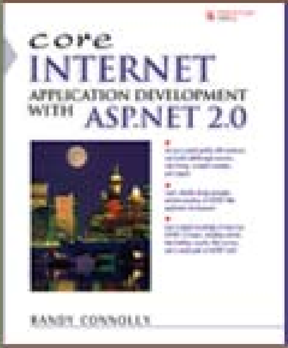
View State



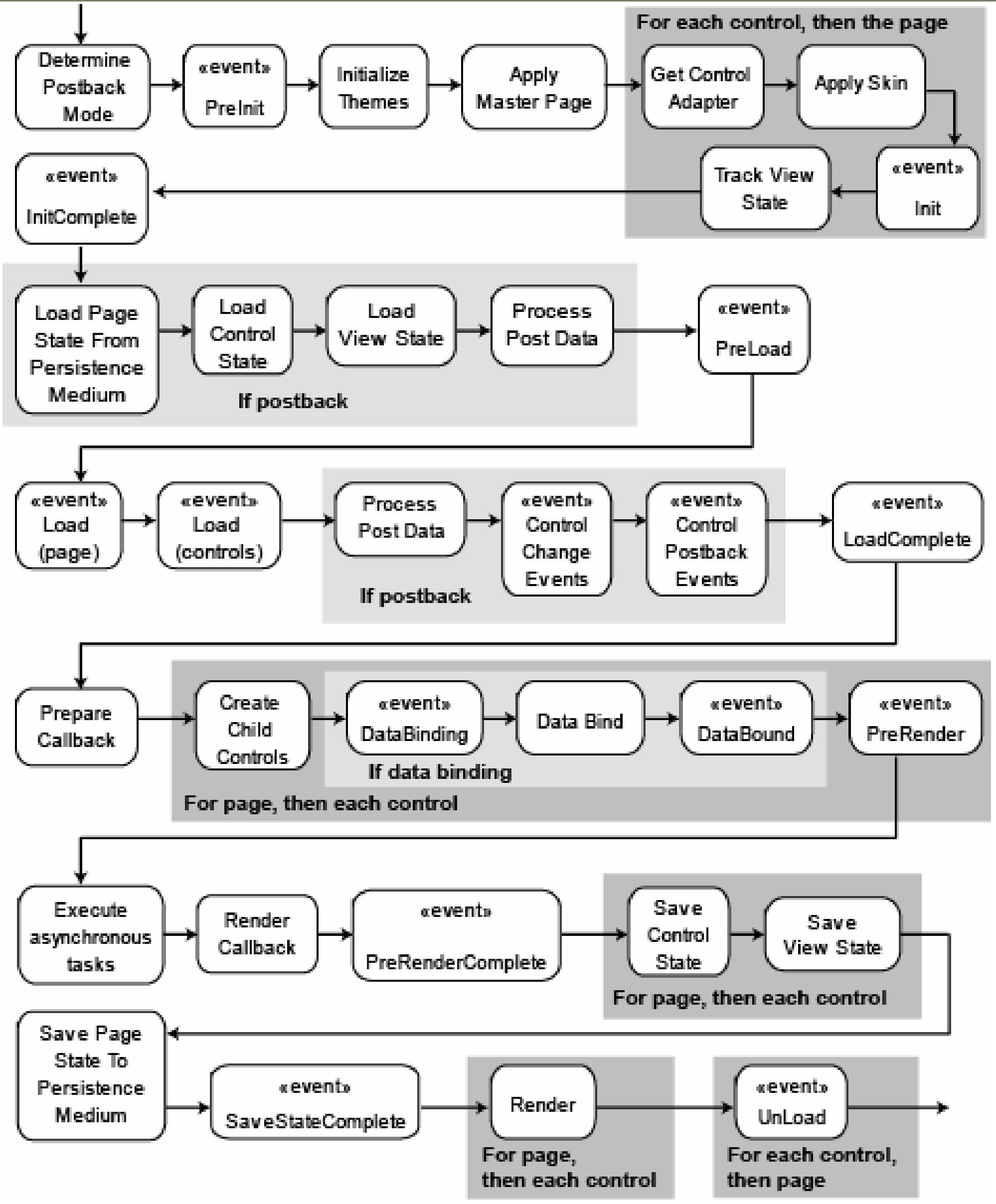
- Since the details of encoding and decoding values from the view state are handled by the ASP.NET runtime, you can generally ignore the view state and simply enjoy its benefits.
- However, sometimes a developer may wish to turn off the view state for a page.
  - For instance, if a very large data set is being displayed, the view state will also be quite large, which may significantly lengthen the time it takes the browser to download and render the page.
  - If a page is not going to post back to itself, you can improve page performance by disabling the view state for the page within the Page directive

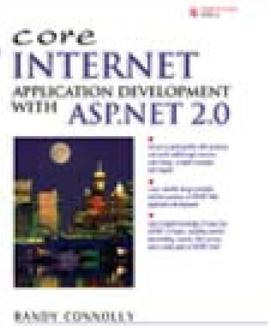
Page Life Cycle

- Page and control events occur in a certain order which we can call the **page life cycle**.
- Five general stages:
  - Page initialization
  - Loading
  - Postback control event handling
  - Rendering
  - Unloading



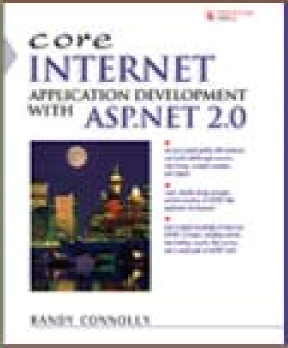
Complete Page Life Cycle





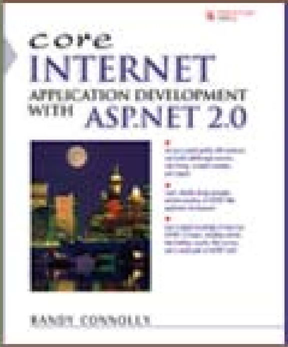
Event Handlers

- Within each of these stages, the ASP.NET page raises events that you can handle in your code.
- For most situations, you only need to worry about the `Page_Load` event and certain control events.



CORE INTERNET APPLICATION DEVELOPMENT WITH ASP.NET 2.0

Prentice Hall, 2007
www.randyconnolly.com/core



Event Handlers

- Because page events always happen, you simply need to define a page event handler by using the appropriate naming convention:
  - `Page_XXXX` where `XXXX` is the event name
- Control events need to be explicitly wired.
  - i.e., you must explicitly bind the handler method to the event.
  - This can be done declaratively in the markup (the usual case),

```
<asp:Button id="btnSubmit" runat="server"
   OnClick="btnSubmit_Click" />
```

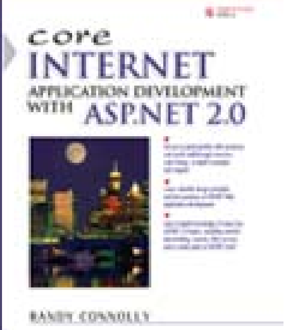
  - or programmatically in the code.

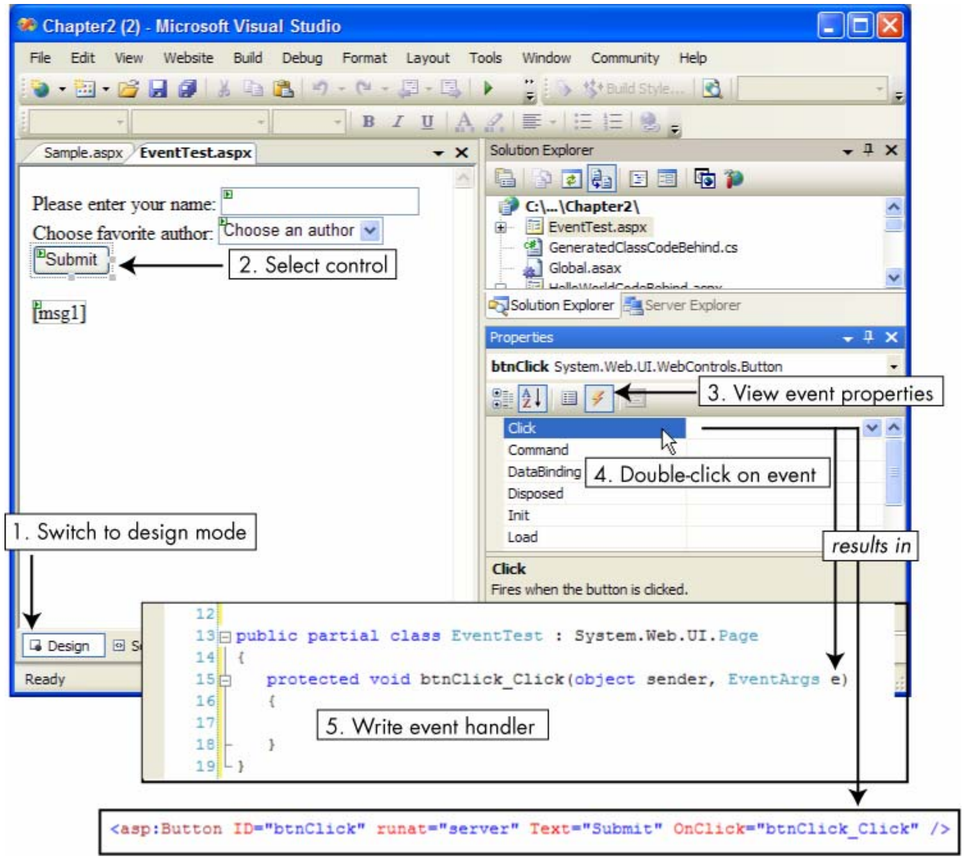
```
btnSubmit.Click += new EventHandler( this.btnSubmit_Click );
```

Adding Event Handlers in VS

CORE INTERNET APPLICATION DEVELOPMENT WITH ASP.NET 2.0

Prentice Hall, 2007
www.randyconnolly.com/core

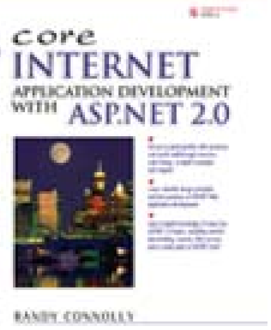




```
public partial class EventTest : System.Web.UI.Page
{
   protected void btnClick_Click(object sender, EventArgs e)
   {
   }
}
```

```
<asp:Button ID="btnClick" runat="server" Text="Submit" OnClick="btnClick_Click" />
```

Event Example

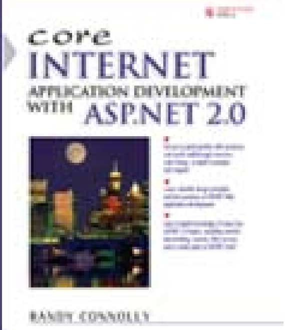


```
Please enter your name:
<asp:TextBox ID="name" runat="server" />
<br />
Choose favorite author:
<asp:DropDownList ID="myList" runat="server">
   <asp:ListItem>Choose an author</asp:ListItem>
   <asp:ListItem>Atwood</asp:ListItem>
   <asp:ListItem>Austin</asp:ListItem>
   <asp:ListItem>Hawthorne</asp:ListItem>
   <asp:ListItem>Melville</asp:ListItem>
</asp:DropDownList>
<br />
<asp:Button ID="btnEnter" Text="Enter" runat="server"
   OnClick="btnEnter_Click" />
<p><asp:Label ID="msg1" runat="server" /></p>
```

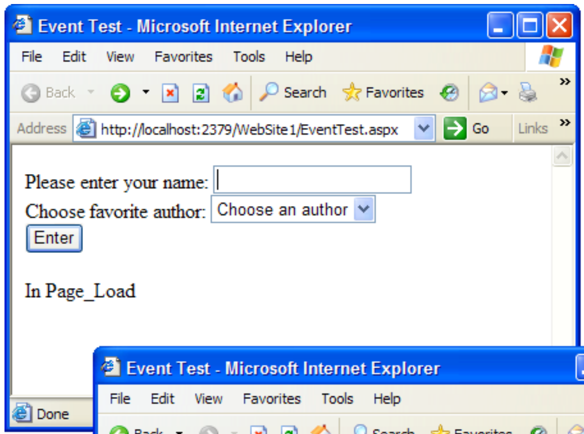
Event Example

CORE INTERNET
APPLICATION DEVELOPMENT
WITH
ASP.NET 2.0

Prentice Hall, 2007
www.randyconnolly.com/core

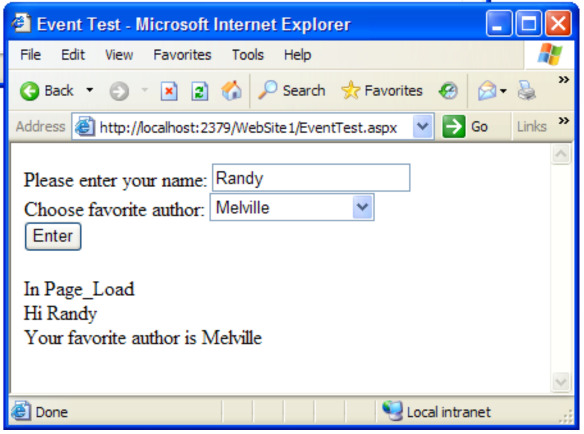


```
…
public partial class EventTest: Page
{
   protected void Page_Load(object sender, EventArgs e)
   {
       msg1.Text = "In Page_Load<br/>";
   }


   protected void btnEnter_Click(object sender, EventArgs e)
   {
       if (myList.SelectedIndex > 0)
       {
           msg1.Text += "Hi " + name.Text + "<br/>";
           msg1.Text += "Your favorite author is ";
           msg1.Text += myList.SelectedItem;
       }
   }

}
```


Event Test - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:2379/WebSite1/EventTest.aspx Go Links
Done

Please enter your name:
Choose favorite author: Choose an author
Enter

In Page_Load

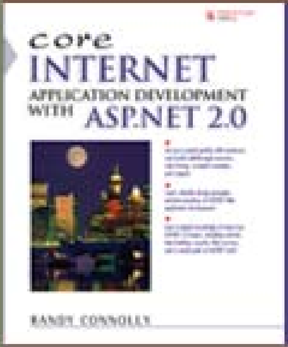

Event Test - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:2379/WebSite1/EventTest.aspx Go Links
Done
Local intranet

Please enter your name: Randy
Choose favorite author: Melville
Enter

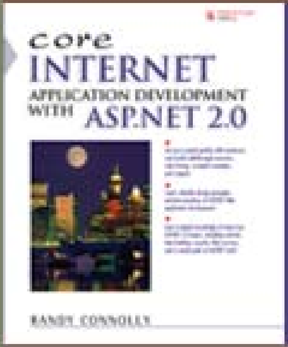
In Page_Load
Hi Randy
Your favorite author is Melville

Detecting Postback

- There are times when you may want your page to behave differently the very first time it is requested.
  - One typical example is that you want to read and display values from a database in a list only the first time the page is requested.
  - In subsequent postbacks, the data is preserved by the view state so there is no need to re-read the database.



Detecting Postback

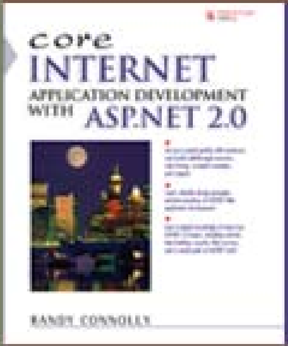


- Your page can test if it is being requested for the first time via the `IsPostBack` property
  - This property is equal to `false` if the page is being requested for the first time.

```
protected void Page_Load(object sender, EventArgs e)
{
   …
   if (! IsPostBack)
   {
      // Do something here for very first request
   }
   …
}
```

Postback and Non-Postback Controls

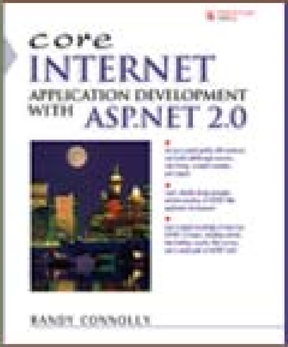
- Button-type controls with Click events always generate an immediate postback to the server.
- But not all control events generate an immediate postback.
- In fact, most control events by default do not cause a postback.
  - Some controls—for instance, a Label control—never can cause a postback.
  - **Change events** also do not generate a postback, by default.
    - An example of a change event is selecting an item from a drop-down list or entering text into a text box.



CORE INTERNET APPLICATION DEVELOPMENT WITH ASP.NET 2.0

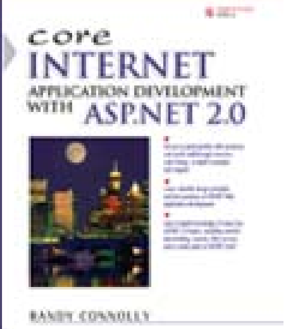
Prentice Hall, 2007
www.randyconnolly.com/core

Change Events



- You may be able to enable postback for change-type events by setting the control's `AutoPostBack` property to `true`.
  - e.g., you could change the previous example so that the `DropDownList` control automatically causes a postback.
  - By doing so, you could eliminate the button completely and instead do your message processing in the event handler for the `SelectedIndexChanged` event.

Using AutoPostBack

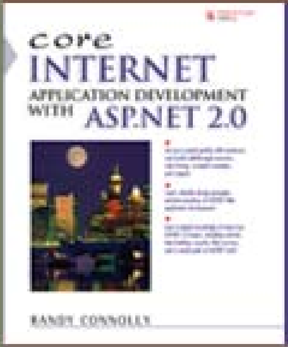


```
Choose favorite author:
<asp:DropDownList ID="myList" runat="server"
   AutoPostBack="true"
   OnSelectedIndexChanged="myList_SelectedIndexChanged" >
   <asp:ListItem>Choose an author</asp:ListItem>
   <asp:ListItem>Atwood</asp:ListItem>
   <asp:ListItem>Austin</asp:ListItem>
   <asp:ListItem>Hawthorne</asp:ListItem>
   <asp:ListItem>Melville</asp:ListItem>
</asp:DropDownList>
<p><asp:Label ID="msg1" runat="server" /></p>
```

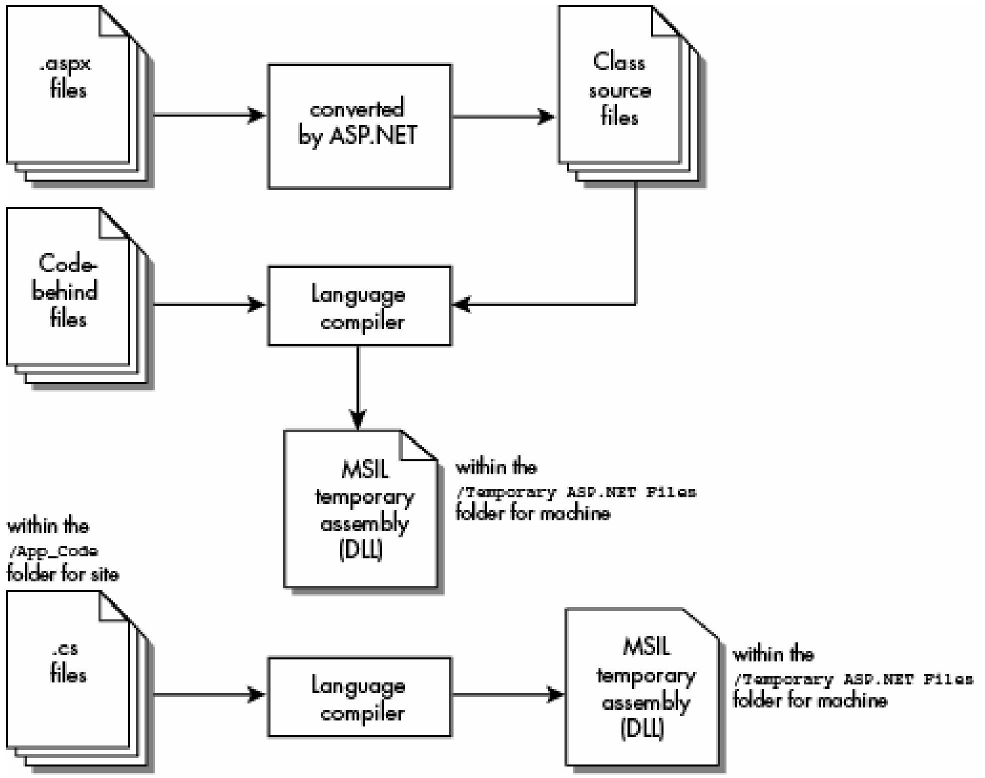
```
protected void myList_SelectedIndexChanged(object sender, EventArgs e)
{
  // Ignore first item in list
  if (myList.SelectedIndex > 0)
  {
    msg1.Text += "Hi " + name.Text + "<br/>";
    msg1.Text += "Your favorite author is ";
    msg1.Text += myList.SelectedItem;
  }
}
```

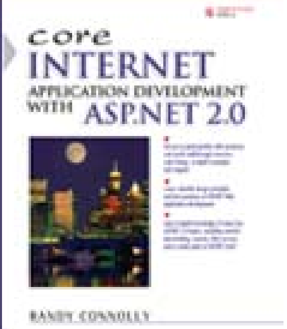
ASP.NET Behind the Scenes

- What happens when the browser requests an ASP.NET web page?
- Quick Answer
  - the visual elements of the page are parsed into a class,
  - this class, along with its code is dynamically compiled (into MSIL),
  - This MSIL is JIT compiled and then executed on the server,
  - Execution produces the HTML and Javascript that is then sent to the browser.

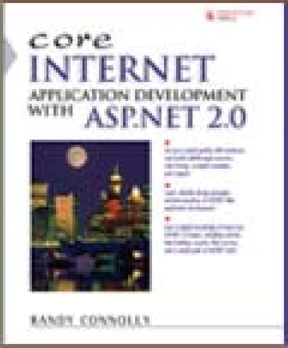


ASP.NET 2.0 Compilation Process


.aspx files
converted by ASP.NET
Class source files
Code-behind files
Language compiler
MSIL temporary assembly (DLL)
within the /Temporary ASP.NET Files folder for machine
within the /App_Code folder for site
.cs files
Language compiler
MSIL temporary assembly (DLL)
within the /Temporary ASP.NET Files folder for machine



Where is this stuff?

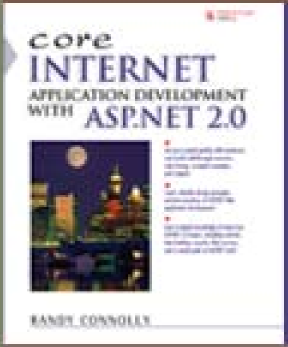


- The path for the generated class files created from the web forms along with the temporary assemblies is
  - `\[.NET System Directory]\Temporary ASP.NET Files\[virtual directory]\[x]\[y]`
    where x and y are randomly-generated names.
  - For instance, the path for the assembly on my development server was

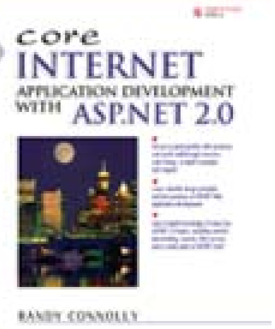
```
C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\Temporary ASP.NET Files\chapter2\7229f9fd\8d0746a9.
```

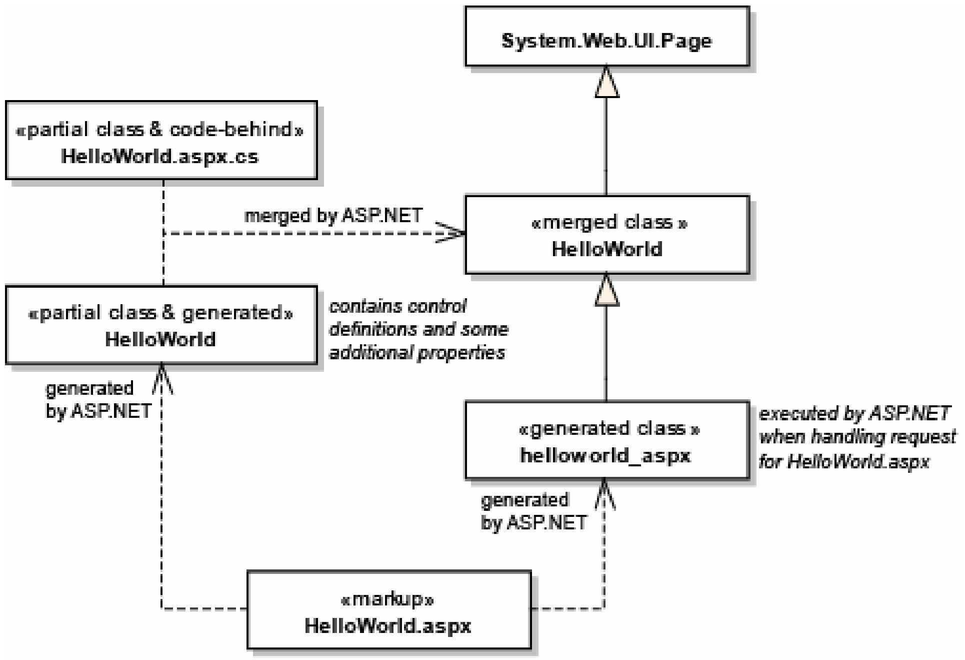
Page class

- All Web forms ultimately inherit from the `Page` class, which is defined in the `System.Web.UI` namespace.



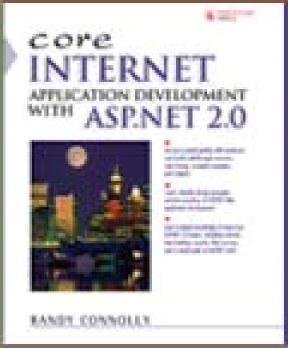
ASP.NET 2.0 Class Inheritance




System.Web.UI.Page
«partial class & code-behind» HelloWorld.aspx.cs
merged by ASP.NET
«merged class» HelloWorld
«partial class & generated» HelloWorld
contains control definitions and some additional properties
generated by ASP.NET
«generated class» helloworld_aspx
executed by ASP.NET when handling request for HelloWorld.aspx
generated by ASP.NET
«markup» HelloWorld.aspx

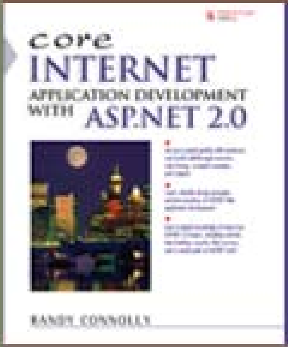
Page class

- The `Page` class inherits from the `TemplateControl` class, which in turn inherits from the `Control` class.
- As a result, the `Page` class provides a great deal of functionality exposed as properties and methods that you can make use of in your web forms.
- Some of these properties are analogous to the intrinsic global objects of ASP classic, such as `Request`, `Response`, `Session`, and `Server`.

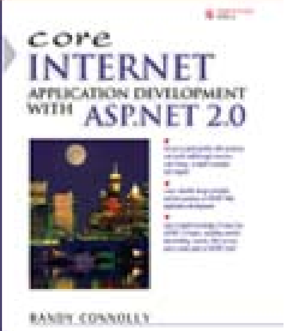


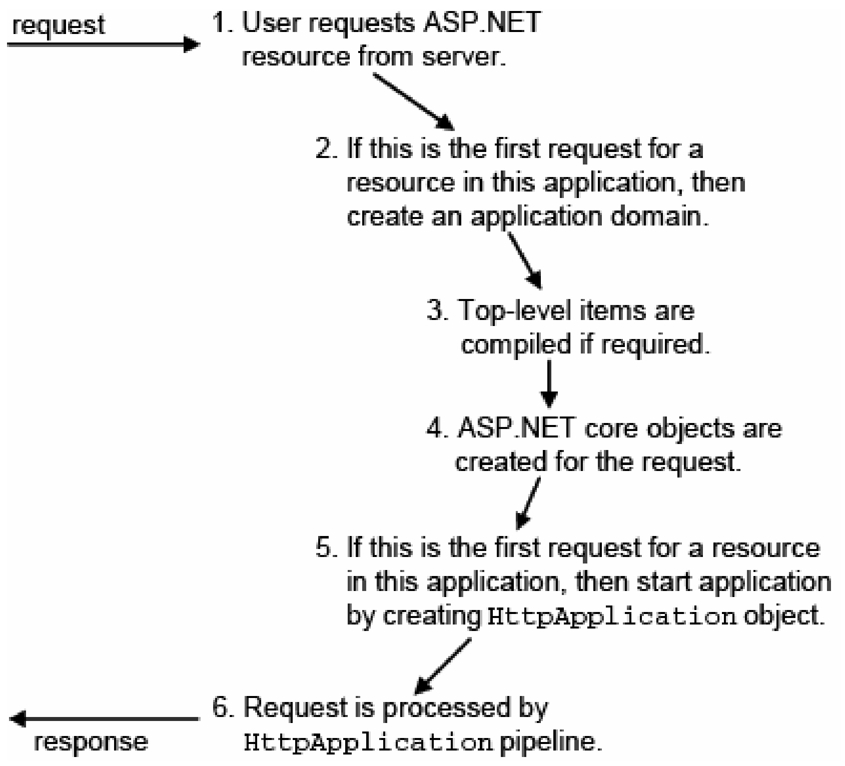
Application Lifecycle

- The page life cycle is just one of several processing steps which occur as part of the larger **ASP.NET life cycle**.



Application Lifecycle




request
1. User requests ASP.NET resource from server.
2. If this is the first request for a resource in this application, then create an application domain.
3. Top-level items are compiled if required.
4. ASP.NET core objects are created for the request.
5. If this is the first request for a resource in this application, then start application by creating HttpApplication object.
6. Request is processed by HttpApplication pipeline.
response