ASP.NET

Lecture 13

CS 638 Web Programming

Definitions

- ASP dot Net
- ASP = Active Server Pages
  - Superset of HTML + Programs that get rendered (the Active part) into plain HTML
- dot Net
  - A well done but enormous object oriented distillation of all the services of Windows

Other Examples of “Active” Pages

- Take “Active” to mean that a server processes non-HTML information to generate a final HTML page
- Many other examples of “Active” pages:
  - PHP, Perl, Active Java, javascript (client) Active Python, etc.
- Without a concept of “Active” pages, every conceivable page would be “hard coded”

ASP.NET “Page”

- The typical ASP.NET page is two files
  - foo.aspx   Page layout
  - foo.aspx.cs   “Code Behind” implementation
- The “.cs” refers to C# - other languages are possible such as Visual Basic and Python
- Separation of layout from implementation is a VERY good thing

Trivial ASPX File

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Stub.aspx.cs" Inherits="Stub" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Untitled Page</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:Label ID="lblTime" runat="server" />
    </div>
    </form>
</body>
</html>
```

Trivial ASPX File

- Entire “Page” directive for the server only
- CodeFile says where implementation is
- Inherits defines Class name – the whole page becomes an instance of this class
- Runat tells server to make the object visible in the class – value always “server”
- All ASP.NET objects begin with “<asp:”
- Look for “lblTime” in next slide

Trivial Code Behind File

```
using System;
using System.Data;
using System.Configuration;
using System.Collections;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;

public partial class Stub : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        lblTime.Text = DateTime.Now.ToLongTimeString();
    }
}
```

Trivial Code Behind File

- Class name same as defined in ASPX file
- “partial” class – notice you don’t see lblTime’s C# definition, it is elsewhere
- “Page_Load” is an event handler that all implementations have by default
- lblTime is an instance of the Label class

Output from Trivial Page

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head><title>
        Untitled Page
</title></head>
<body>
   <form name="form1" method="post" action="Stub.aspx" id="form1">
<div>
<input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE" value="/stuff" />
</div>
   <div>
      <span id="lblTime">2:52:13 PM</span>
   </div>
   </form>
</body>
</html>
```

What Just Happened?

- Browser user asks for Stub.aspx
- Server checks to see if Stub.aspx.cs needs to be compiled
- Stub.aspx.cs is executed in various stages part of the server “lifecycle” of the page
- Second to last stage is HTML “rendering” into the layout specified by ASPX file
  - The final HTML is composed and returned

Concepts Before “Lifecycle”

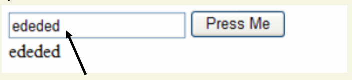
- Discuss three concepts before lifecycle can be discussed
  - State
  - Postback events
  - Lifecycle events

State in a Stateless World

- Some controls keep their history or “state”
- Example: text box

```
……
<asp:TextBox runat="server" ID="tbTextBox" />
<asp:Button runat="server" ID="btnButton" Text="Press Me" /><br />
<asp:Label runat="server" ID="lblText" />
……

protected void Page_Load(object sender, EventArgs e)
{
   lblText.Text = tbTextBox.Text;
}
```



How is State Maintained?

- ASP.NET has *several* alternative methods
- Most common is use of VIEWSTATE
  - Content of VIEWSTATE is encrypted state information (potentially long string)
  - Gets around stateless environment by tagging along in the page's HTML from server to browser and back again
  - "Invisible" HTML input object

<input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE" value="serialized stuff" />

Serialization

- How does the gibberish in VIEWSTATE represent real data?
  - Serialization is the process of converting a complex object to a series of bytes for storage http://www.codeproject.com/csharp/objserial.asp
  - Deserialization is the reverse: unpacking a series of bytes into an original form
- Old fashioned: write binary data
- Modern general purpose way: XML (perhaps encrypted)

.NET Serialization

- Any .NET object can be serialized / deserialized using standard functions
- Data is created as name / value pairs of text and binary along with specification of the object (data types etc.)
- Result is unencrypted
- ASP.NET handles serialization *and* encrypting for you of VIEWSTATE and "Session"

XML vs. .NET Serialization

XML	.NET Serialization
Entirely text	Text / binary mixed
Meant for humans and computers	Meant for computers
Naturally handles large volumes of data	Meant for smaller data objects
Easily searched, modified, extended	

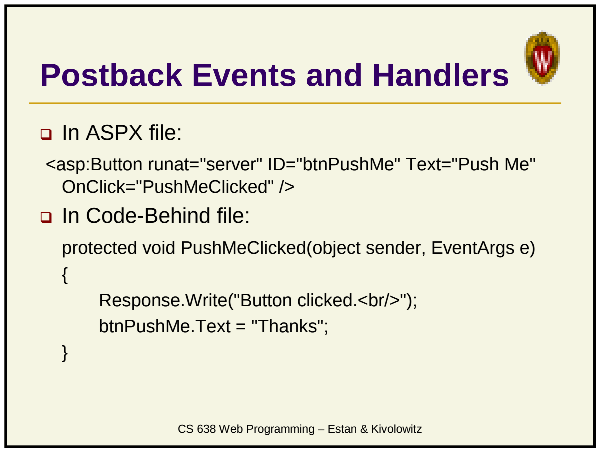
You will learn more about XML later in the semester

Other Means of Preserving State

- Cookies
  - State information is "serialized" and stored on client's computer – privacy implications
- URL "Query Strings"
  - State information is "serialized" and tagged onto the URL of the page
- Database
  - Allows for long term state preservation on server
- Session
  - State information "serialized" and stored in memory of server

Postback Events

- A postback happens when something causes the browser to send a request back to the server (button push, for example)
- When the request arrives at the server, it is examined to see what caused the postback and an "event" is "raised"
- Fancy way of saying if some code is defined to "handle" the event, it is called



Postback Events and Handlers

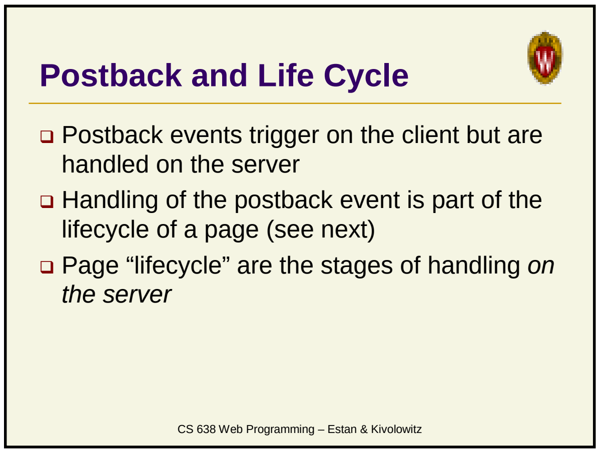
- In ASPX file:

```
<asp:Button runat="server" ID="btnPushMe" Text="Push Me"
   OnClick="PushMeClicked" />
```

- In Code-Behind file:

```
protected void PushMeClicked(object sender, EventArgs e)
{
      Response.Write("Button clicked.<br/>");
      btnPushMe.Text = "Thanks";
}
```

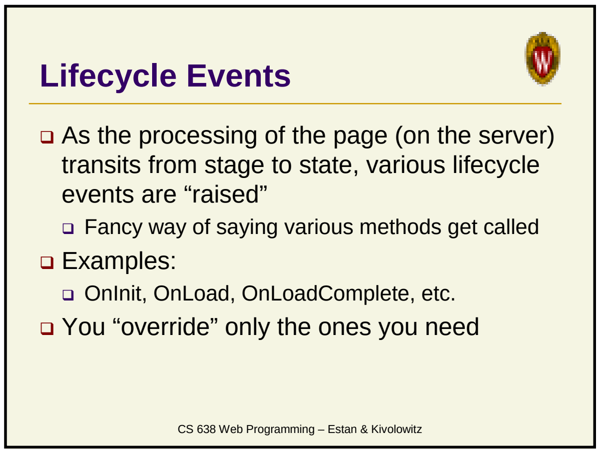
CS 638 Web Programming – Estan & Kivolowitz



Postback and Life Cycle

- Postback events trigger on the client but are handled on the server
- Handling of the postback event is part of the lifecycle of a page (see next)
- Page “lifecycle” are the stages of handling *on the server*

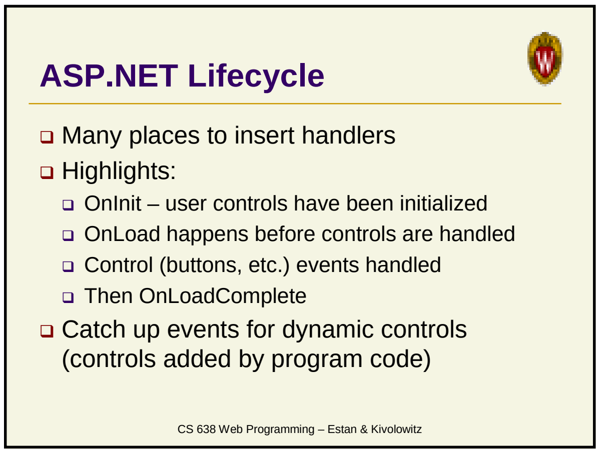
CS 638 Web Programming – Estan & Kivolowitz



Lifecycle Events

- As the processing of the page (on the server) transits from stage to state, various lifecycle events are “raised”
  - Fancy way of saying various methods get called
- Examples:
  - OnInit, OnLoad, OnLoadComplete, etc.
- You “override” only the ones you need

CS 638 Web Programming – Estan & Kivolowitz



ASP.NET Lifecycle

- Many places to insert handlers
- Highlights:
  - OnInit – user controls have been initialized
  - OnLoad happens before controls are handled
  - Control (buttons, etc.) events handled
  - Then OnLoadComplete
- Catch up events for dynamic controls (controls added by program code)

CS 638 Web Programming – Estan & Kivolowitz