The World Wide Web: Web Applications and Web Forms, continued

Web Application Programming

- Simple HTML is ok for "static pages"
  - When no user input is solicited
- But "real" web apps accept input from users
  - May need to retrieve data from a database
  - May perform computations
  - The HTML they return to browsers will change depending on input and results
  - Client side is easy
    - Just use ordinary HTML
    - Or scripts that run on the client side
      - e.g., JavaScript or VBscript
  - What about server side?

Forms and the Server Side

- The heart of most real web apps that accept input are HTML Forms: <form> … </form> tags
  - Some fields (lots of them):
    - <input type="text" … />
      - Browser renders this tag as a textbox input field
    - <input type="submit" … />
      - Browser renders this as a push button
      - When clicked, form is submitted to Web server
        - » i.e., browser submits form with any input from form's controls
    - How submitted depends on whether a "Method" attribute is present
      - If not or if form contains a method="get" tag:
        - » Browser sends an HTTP GET command to server with user input appended (e.g., user enters 2 and 3):
          GET /calc.html?op1=2&op2=3 HTTP/1.1
      - If there's a method="post" tag
        - » Form is submitted with an HTTP POST command with user input in the body of the HTTP request

Postback

- When user input from an HTML form is submitted back to the server, a "postback" has occurred
  - Look at http://localhost/calc.html
    - "View" | "Source" to see html
    - Click "+" & look in browser address bar to see GET postback data
- The Server should respond to the postback
- An important reality: HTML is "stateless"
  - A page stores no information about its contents from one invocation to another
  - So server side code must be running to extract the user input and generate a new web page that displays the desired result
    - … and restore the original data if it is to be visible

Server Response in calc.html Form

- Calc Form allows user to enter two numbers to be added
- Pressing "=" button submits numbers to server
- Original numbers and sum should be returned to browser
- Server should generate something like the following HTML in response to user entering 2 and 3 and clicking the "=" button:

```
<html>
  <body>
    <form>
      <input type="text" name="op1" value="2" />
      +
      <input type="text" name="op2" value="3" />
      <input type="submit" value=" = " />
      5
    </form>
  </body>
</html>
```

- Note: generating repeat input values gives illusion user is seeing one Web page when really we're seeing two in succession

Generating the Server Response

- One way:
  - Use the Common Gateway Interface (CGI)
    - A low-level programmatic interface between web servers and applications that run on Web servers
    - A server-side program script that reads inputs, computes, and writes http responses back to browser
    - Usually written in Perl, but can be done in other languages (C)
    - Hard to use, slow, and has security issues
    - Not used any more except on UNIX-based Web servers
- Another way:
  - Windows ISAPI Extensions
    - Internet Server Application Programming Interface
    - Set of Windows DLLs to which IIS forwards HTTP requests
      - DLL generates HTML responses
    - Faster than CGI
      - But also hard to use

Programming Web Apps Using ASP

- Mix html and server-side scripts in a single .asp file
  - Scripts usually written in VBScript or JavaScript
    - Interpreted, so can be slow
- When an ASP page is requested, the page is parsed by server and any scripts it contains are executed
  - ASP Request object accesses input
  - ASP Response object writes HTML to the http response
  - The following example uses VBScript code between <% and %> tags to check incoming requests for inputs named op1 & op2
    - VB script converts them to integers, adds them, converts result to a string, and writes string to http response using Response.Write( )
  - Note the HTML returned ("View" | "Source")

Calc.asp Example Program

```
<%@ Language = "VBScript" %>

<html>
  <body>
    <form>
      <input type="text" name="op1" value="<%= Request ("op1") %>"/>
      +
      <input type="text" name="op2" value="<%= Request ("op2") %>"/>
      <input type = "submit" value=" = " />
      <%
        If Request ("op1") < > "" And Request ("op2") < > "" Then
          a = CInt (Request("op1"))
          b = CInt (Request("op2"))
          Response.Write (CStr (a + b))
        End If
      %>
    </form>
  </body>
</html>
```

Problems with ASP

- ASP is a good solution for doing server-side processing of HTML form input and dynamically generating HTML
  - Higher level of abstraction than CGI or ISAPI
  - Also integrates seamlessly with ADO data bases
- But it has some problems
  - Interpreted scripts means slow execution
  - ASP has no true encapsulation model
    - Can't build reusable controls that encapsulate complex rendering and behavior logic
    - No event model as for Windows Forms
- ASP.NET – the new Microsoft solution

ASP.NET and Web Forms

- Web Forms are built from a combination of HTML, scripts, and server controls
  - Some examples: Button, TextBox, Label, DropdownList
  - Defined in System.Web.UI.WebControls
- Object oriented
  - Whenever a Web page with server control objects is requested:
    - ASP.NET asks each object to render itself into HTML
    - HTML returned by controls is included in the HTTP response
    - Scripts (w/ event handlers) are executed as in ASP
- Example calc.aspx
  - Put into \Inetpub\wwwroot IIS virtual directory
    - IIS must be running on the computer
  - Run locally by typing http://localhost/Calc.aspx

Calc.aspx

```
<html>
  <body>
    <form runat="server">
      <asp:TextBox ID="op1" RunAt="server" />
      +
      <asp:TextBox ID="op2" RunAt="server" />
      <asp:Button Text=" = " OnClick="OnAdd" RunAt="server" />
      <asp:Label ID="Sum" RunAt="server" />
    </form>
  </body>
</html>

<script language="C#" RunAt="server">
 void OnAdd (Object sender, EventArgs e)
 {
   int a = int.Parse(op1.Text);
   int b = int.Parse(op2.Text);
   Sum.Text = (a + b).ToString ();
 }
</script>
```

ASP.NET Controls

- Server Control tag HTML format:

  <asp:Control properties event="handler" RunAt="server" />

  - Button Control example:

    <asp:Button Text=" = " OnClick="OnAdd" RunAt="server" />

    - Text= " = " Text property (= sign is displayed on button)
    - OnClick="OnAdd" Wires OnAdd event handler to button's Click event
  - RunAt="server" attribute
    - Signals ASP.NET to execute the tag rather than treat as static HTML
    - Must be used with every tag that ASP.NET is to process
- Server Control script
  - Specify language and content of event handlers
  - Here script's Click handler reads Text properties of TextBoxes ("op1" & "op2") and converts them to integers
  - Result is converted to a string and put into "Sum" Label's Text property

HTML Returned by calc.aspx

```
<html>
 <body>
  <form name="_ctl0" method="post" action="calc.aspx" id="_ctl0">
   <input type="hidden" name="__VIEWSTATE"
   value="dDwxOTE0NDY4ODE2O3Q8O2w8aTwxPjs+O2w8dDw7bD
   xpPDc+Oz47bDx0PHA8cDxsPFRleHQ7PjtsPDU7Pj47Pjs7Pjs+ Pjs+ Pj
   s+dBISuMwmkqCcHr+yOOms28nF+A0=" />

   <input name="op1" type="text" value="2" id="op1" />
   +
   <input name="op2" type="text" value="3" id="op2" />
   <input type="submit" name="_ctl1" value="  =  " />
   <span id="Sum">5</span>
  </form>
 </body>
</html>
```

What Has ASP.NET Done?

- (Look at "View" | "Source" after entering data and clicking "=")
- ASP.NET:
  - Turned TextBox controls into <input type="text"> tags
  - Turned Button control into <input type="submit"> tag
  - Turned Label control into (formatting) tag
- The controls project a user interface by rendering themselves into HTML

What Happened?

- User clicking "=" button posts form back to server using HTTP POST
- ASP.NET notifies the Button object and Button responds by firing a Click event on the server
- ASP.NET then calls OnAdd() handler
  - (We write this)
  - In calc.aspx, it computes the sum, converts it to a string, and puts it in the Sum label's Text property
- Then renders the result into an HTML page
  - Since Sum.Text is now a non-null string, output of Sum label control includes that string inserted between tags
- Page is returned to the browser

__VIEWSTATE Tag

- Mechanism ASP.NET uses to round-trip data from client to server back to client
- Recall HTML is stateless
  - Nothing is remembered when a new page replaces the old one
  - So how do we determine if the state changed?
  - View State
    - A place where controls can store their state in a way that it remains valid from one request to the next
    - Especially useful for controls that fire change events
  - View State is transmitted to the client in a hidden control and transmitted back to the server with the form's postback data
    - __VIEWSTATE tag contains all data encoded so that ASP.NET can detect changes to the page and fire change events

Using VS Designer to Create Web Forms

(Must have IIS installed to be able to use VS for Web Forms)

- Start Visual Studio
- "File" | "New" | "Project"
  - Visual C# Projects
  - "ASP.NET Web Application" template
  - Location: http://localhost/FormName
- Two source files are generated:
  - ***.aspx and ***.aspx.cs
    - Separates the HTML from the C# code script
    - Called code-behind programming
- Drag and drop server controls from tool box
  - Set properties and add event handlers as for Windows Forms applications
  - Edit the skeleton code generated by Designer
- An example: http://localhost/SimpleTest/WebForm1.aspx

Using Microsoft Web Matrix

- "Light" version of Visual Studio ASP.NET for Web development
  - Mini Web Server that doesn't require IIS
- Download from: http://asp.net/webmatrix
- Contains most of the features of Visual Studio
  - But no " intellisense" typing and no online help
  - And Web apps and services only run on localhost
    - But can be ported to an IIS server on another machine
  - No code-behind
    - A single *.aspx file is generated with both HTML and code
    - But it can be viewed in:
      - Design View (where tool box can be used to add server controls)
      - Code View (C# or VB code only, like *.aspx.cs in VS)
      - HTLM View (like *.aspx in VS)
      - All: the entire *.aspx file

Using Web Matrix

- Best way to learn is to go through the first part of the Web Matrix on-line guided tour:
  - http://asp.net/webmatrix/guidedtour
- Also see the tutorial:
  - http://www.alanddave.com/books/webmatrix/download/webmatrix.pdf

Creating and Running a New Web Matrix App

- Start Web Matrix
- "File" | "New File"
- Enter name (.aspx) and choose language (C#)
  - *.aspx skeleton file is generated
- Drag and drop server controls from tool box
  - (Not as much flexibility in placing the controls as in VS)
    - You'll need to use the <Enter> and <space> keys a lot
  - Set control properties and add event handlers as in Visual Studio
  - Edit the skeleton code generated by Web Matrix
    - Select the "Code" tab
  - Edit the HTML generated by Web Matrix as needed
    - Select the "HTML" tab
- Try to run the application
  - Click on blue right-pointing triangle on tool bar
  - Or "View" | "Start", Or press the F5 key
- An example: SimpleTest.aspx

Running your Web Matrix Web Form App on an IIS Server

- Copy the .aspx file into the IIS server's Inetpub\wwwroot directory
- If your browser is running on the same machine as the server:
  - Run the app by typing the following into the browser's URL address:
    http://localhost/pgm-name.aspx
- If your browser is running on another machine:
  - Run the app by typing the following into the browser's URL address:
    http://machine-name/pgm-name.aspx

Using a Virtual Directory

- If you don't want to clutter up the Inetpub\wwwroot directory on the target computer, put your program into a virtual directory
- Creating a Virtual Directory on target machine
  - "Start" | "Control Panel" | "Administrative Tools"
  - Start "Internet Information Services"
    - In left pane expand the Local Computer\Web Sites folder
    - Select Default Web Site
    - From "Action" menu item
      - "New" | "Virtual Directory"
      - Starts the Virtual Directory Creation Wizard
        - » Type in an alias name for the directory
        - » Enter or browse to the path of the desired directory
        - » Click "Next" and "Finish"
- To run the app, type URL address into browser:
  http://machine-name/alias-name/pgm-folder/pgm-name.aspx