ASP – 2

The Page Class

Page class

- Your class always inherits from System.Web.UI.Page and you should be aware of some of its **properties**:
- **Response** Object
  - Sends http responses to the client
- **Request** Object
  - Gets the HttpRequest object that provides access to data contained in the current request

Request object

- Request object retrieves data sent to the server from the client as part of a POST request.
- If the specified variable is not in one of the following five collections, the Request object returns EMPTY.

Request object

- All variables can be accessed directly by calling Request(*variable*) without the collection name. In this case, the Web server searches the collections in the following order:

1. **QueryString**
2. **Form**
3. **Cookies**
4. **ClientCertificate**
5. **ServerVariables**

Request object

- If a variable with the same name exists in more than one collection, the Request object returns the first instance that the object encounters.
- It is strongly recommended that when referring to members of a collection that the full name be used. For example, rather than Request.("AUTH_USER") use Request.**ServerVariables**("AUTH_USER").
- This allows the server to locate the item more quickly.

Page class - IsPostBack

- IsPostBack – A bool that indicates whether the page is being loaded for 1st time (false) or in response to a postback (true).
- Use IsPostBack in the Page_Load function to perform one time initialization of things, e.g., connect to a database and fill a combo box with some values.

Page class - IsPostBack

```
void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)   // page is requested for the first time
    {
        // do any one time things; e.g., fill controls with data
    }
}
```

- The Page_Load event is one of 4 defined by the Page class. Each occurs each time a page is sent to the server:

Page class events

- **Init** – occurs before server-side controls have been restored
- **Load** – controls have been initialized and their values are accessible
- **PreRender** – server side events have been fired but no html has been rendered
- **UnLoad** – occurs after page rendering has been completed

Cross page posting

- In .NET 1.1 a page could only post to itself.
- Now you can designate a target page, rather than the current page, by setting the PostBackUrl property of a Button, ImageButton or LinkButton to the address of the target page.
- MyButton.PostBackUrl = “nextpage.aspx”;

Cross page posting

- PostBackUrl property can also be set using the property editor.
- In nextpage.aspx Page_Load event handler:

String s = PreviousPage.Title;

Note how, in the page that receives control, you can use the PreviousPage property to find out things about the page that posted to you.

Web Form Controls

- You can use three types of controls in your web forms:
- Client side html controls – traditional html controls <table> <submit> <input> <input type=text name=htf>
- Html server controls – These are just like the client controls except that they execute on the server. <input type=text name=htf runat=server>

Web Form Controls

- Native ASP.NET web controls – extend features of html server controls and add non-html controls like Calendar and DataGrid.
- They are classes and expose properties and methods that give you a great deal of control.

Web Form Controls

- Work much like the html server controls – each is a class with a corresponding interface that renders as html understandable by browsers.
- Declared using asp prefix:

```
<asp:TestBox id="FirstName" type="text"
  width="100px" runat="server>
</asp:TextBox>
```

Web Form Controls

- Most of the controls have lots of properties that can be set with the property editor.
- I leave it up to you to figure out how to work with controls and get them functioning in web pages.

Session Object

- You can use the Session object to store information needed for a particular user session.
- Variables stored in the Session object are not discarded when the user jumps between pages in the application; instead, these variables persist for the entire user session.

Session Object

- The Web server automatically creates a Session object when a Web page from the application is requested by a user who does not already have a session.
- The server destroys the Session object when the session expires or is abandoned.

Session Object Methods

- Session.Abandon
- Session.Contents.Remove
- Session.Contents.RemoveAll

Session Object Properties

- Session.CodePage
- Session.Contents.Collection
- Session.LCID
- Session.SessionID
- Session.Timeout

Session Object

- Anything that you want to persist from request to request must be placed in Session, else it's gone.

e.g., Page_Load sets class instance variable to 10;

Button click adds 20 to it

Next time page is loaded, variable = 10

Session Object

- Uses:

```
Session["Username"] = tbUsername.Text;
Session["cust'] = new Cust("Henry");

string username = (string) Session["username"];
Cust c = (Cust) Session["cust"];
```

Session Object

- Session variables can be any valid .NET type (ArrayList, List, Array, int, etc.)
- ASP.NET 2.0 and higher support 4 Session modes:

  InProc - enables in-process management (aspnet_state.exe) nt service

Session Object

- **Off** – no session management
- **StateServer** – stored by a surrogate process running on a selected server
- **SqlServer** - maintained in a SQL Server table
- Settings for Session contained in web.config file

Session Object

```
<sessionState
  mode="InProc"
stateConnectionString="tcpip=127.0.0.1:42424"
sqlConnectionString="data source=127.0.0.1;
   Trusted_Connection=yes"
cookieless="false"
timeout="30"
/>
```

Session Object

- Exposes 2 events:
- OnStart – raised when a new Session starts
- OnEnd – raised when a Session is abandoned or expires
- Specified in the Global.asax file