Agenda

- **ASP.NET Overview**
- Programming Basics
- Server Controls

ASP.NET Overview

```
<%@ Page language="c#" %>
<html>
<head></head>
<script runat="server">
public void B_Click (object sender, System.EventArgs e) {
  Label1.Text = "Hello, the time is " + DateTime.Now;
}
</script>
<body>
<form method="post" runat="server">
<asp:Button onclick="B_Click" Text="Push Me“ runat="server“ />
<p>
<asp:Label id=Label1 runat="server" />
</form>
</body>
</html>
```

Programming Model
Postbacks

- A postback occurs when a page generates an HTML form whose values are sent back (via HTTP Post or Get) to the same page
- A common technique for handling form data
- In ASP and other server-side technologies the state of the page is lost upon postback...
- Unless you explicitly write code to maintain state
- This is tedious, bulky and error-prone

Programming Model
Postbacks Maintain State

- By default, ASP.NET maintains the state of all server-side controls during a postback
- Can use `method="post"` or `method="get"`
- Server-side control objects are automatically populated during postback
- No state stored on server
- Works with all browsers

Programming Model
Code-behind pages

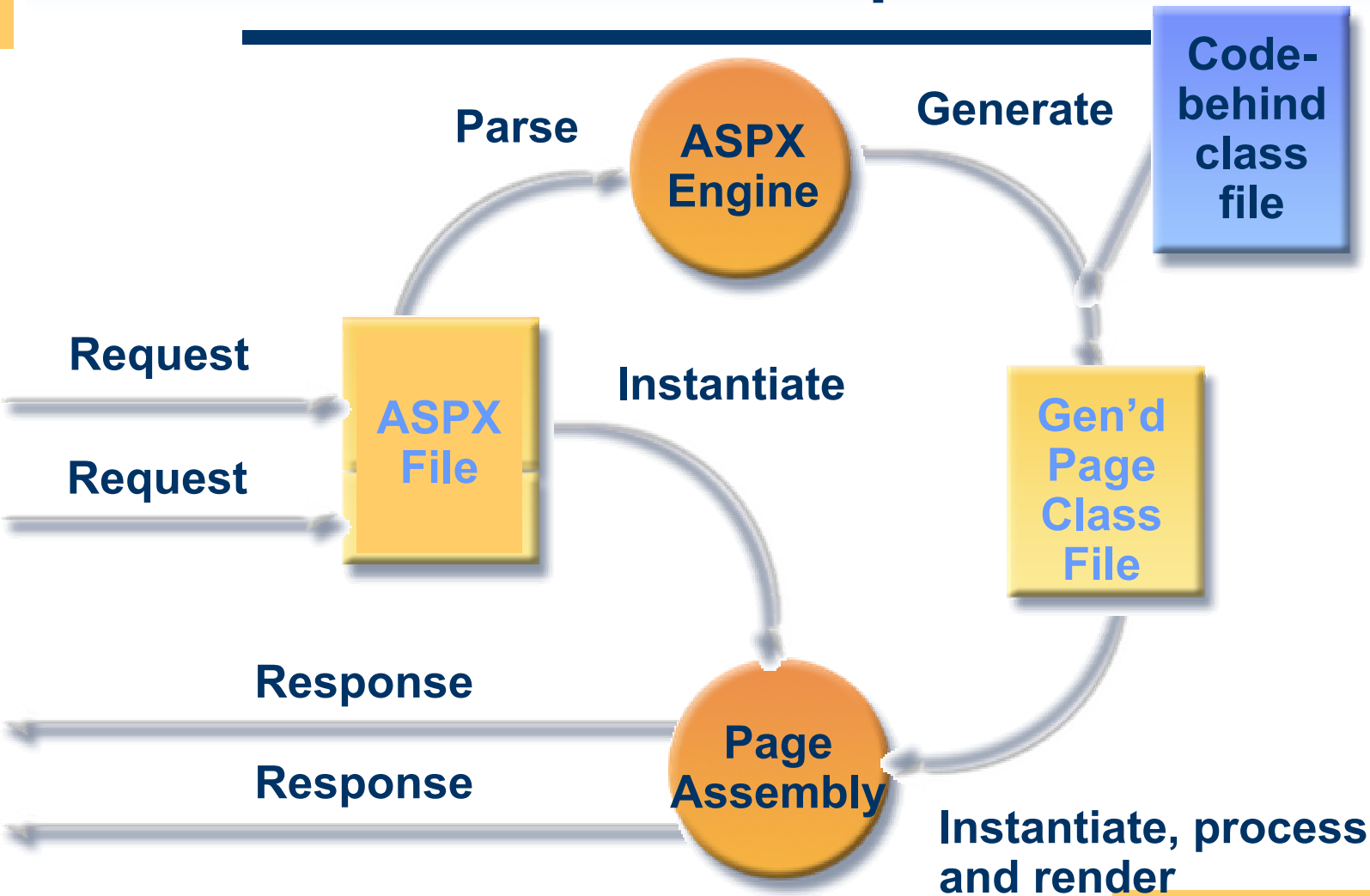
- Two styles of creating ASP.NET pages
  - Controls and code in .aspx file
  - Controls in .aspx file, code in code-behind page
    - Supported in Visual Studio.NET
- Code-behind pages allow you to separate the user interface design from the code
  - Allows programmers and designers to work independently

```
<%@ Codebehind="WebForm1.cs"
    Inherits=WebApplication1.WebForm1" %>
```

Programming Model
Automatic Compilation

- Just edit the code and hit the page
- ASP.NET will automatically compile the page into an assembly
- Compiled code is cached in the CLR Assembly Cache
- Subsequent page hits use compiled assembly
- If the text of the page changes then the code is recompiled
  - Works just like ASP: edit, save and run

Programming Model
Automatic Compilation



Agenda

- ASP.NET Overview
- **Programming Basics**
- Server Controls

Programming Basics
Page Syntax

- The most basic page is just static text
  - Any HTML page can be renamed `.aspx`
- Pages may contain:
  - Directives: `<%@ Page Language="C#" %>`
  - Server controls: `<asp:Button runat="server">`
  - Code blocks: `<script runat="server">…</script>`
  - Data binding expressions: `<%# %>`
  - Server side comments: `<%-- --%>`
  - Render code: `<%= %>` and `<% %>`
    - Use is discouraged; use `<script runat=server>` with code in event handlers instead

Programming Basics
The Page Directive

- Lets you specify page-specific attributes, e.g.
  - `AspCompat`: Compatibility with ASP
  - `Buffer`: Controls page output buffering
  - `CodePage`: Code page for this .aspx page
  - `ContentType`: MIME type of the response
  - `ErrorPage`: URL if unhandled error occurs
  - `Inherits`: Base class of Page object
  - `Language`: Programming language
  - `Trace`: Enables tracing for this page
  - `Transaction`: COM+ transaction setting
- Only one page directive per .aspx file

Programming Basics
Server Control Syntax

- Controls are declared as HTML tags with `runat="server"` attribute

```
<input type=text id=text2 runat="server" />
<asp:calendar id=myCal runat="server" />
```

- Tag identifies which type of control to create
  - Control is implemented as an ASP.NET class
- The `id` attribute provides programmatic identifier
  - It names the instance available during postback
  - Just like Dynamic HTML

Programming Basics
Server Control Properties

- Tag attributes map to control properties

```
<asp:button id="c1" Text="Foo" runat="server">
<asp:ListBox id="c2" Rows="5" runat="server">
```

- Tags and attributes are case-insensitive
- Control properties can be set programmatically

```
c1.Text = "Foo";
c2.Rows = 5;
```

Programming Basics
Maintaining State

- By default. controls maintain their state across multiple postback requests
  - Implemented using a hidden HTML field: `__VIEWSTATE`
  - Works for controls with input data (e.g. `TextBox`, `CheckBox`), non-input controls (e.g. `Label`, `DataGrid`), and hybrids (e.g. `DropDownList`, `ListBox`)
- Can be disabled per control or entire page
  - Set `EnableViewState="false"`
  - Lets you minimize size of `__VIEWSTATE`

Programming Basics
Maintaining State

- Demo: Maintaining State

Programming Basics
Server Code Blocks

- Server code lives in a script block marked `runat="server"`

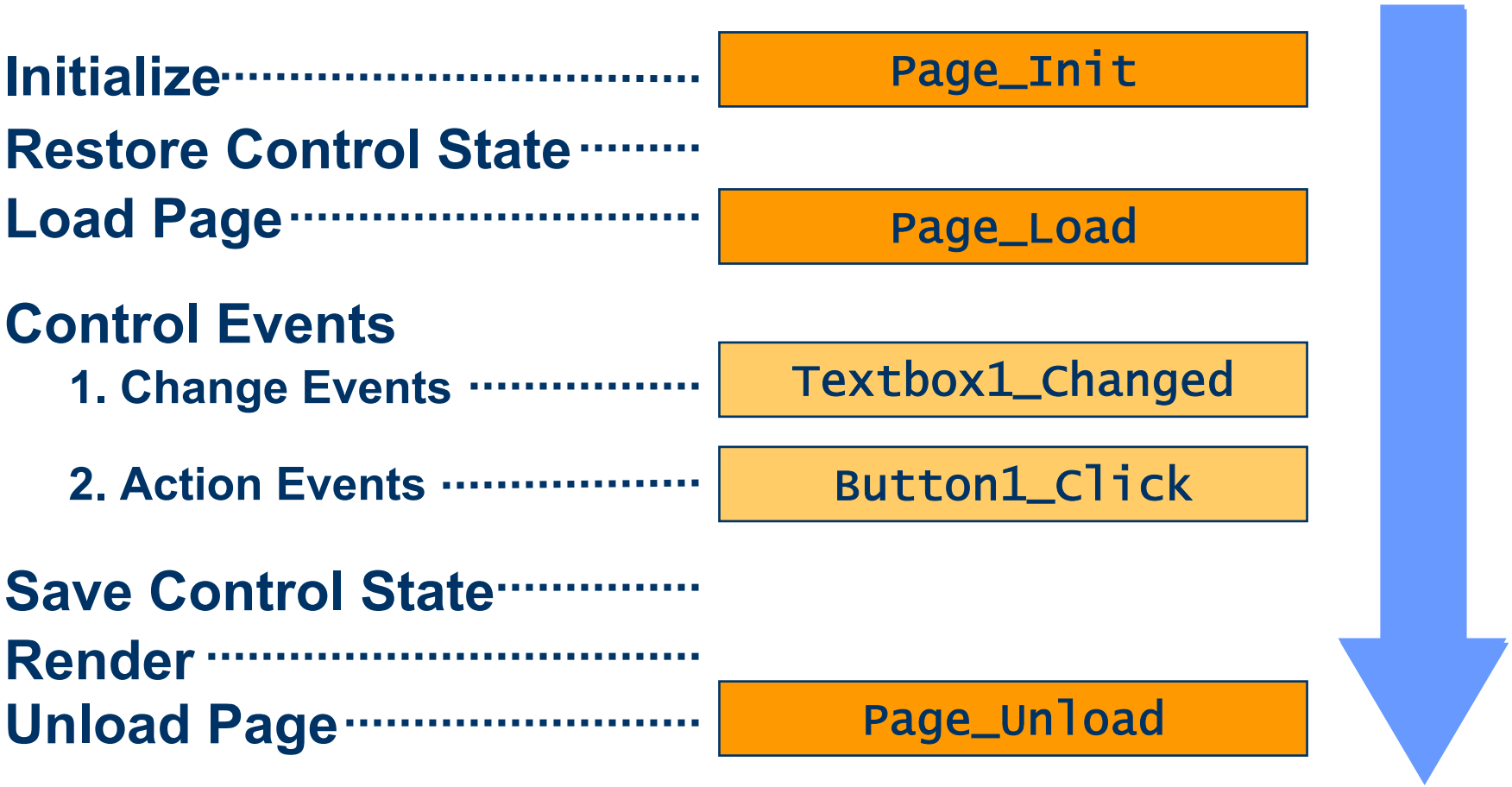
```
<script language="C#" runat=server>
<script language="VB" runat=server>
<script language="JScript" runat=server>
```

- Script blocks can contain
  - Variables, methods, event handlers, properties
  - They become members of the custom `Page` object

Programming Basics
Page Events

- Pages are structured using events
  - Enables clean code organization
  - Less complex than ASP pages
- Code can respond to page events
  - e.g. `Page_Load`, `Page_Unload`
- Code can respond to control events
  - `Button1_Click`
  - `Textbox1_Changed`

Programming Basics
Page Event Lifecycle



Programming Basics
Page Loading

- Page_Load fires at beginning of request after controls are initialized
  - Input control values already populated

```
protected void Page_Load(Object s, EventArgs e) {
  message.Text = textbox1.Text;
}
```

Programming Basics
Page Loading

- Page_Load fires on every request
  - Use Page.IsPostBack to execute conditional logic
  - If a Page/Control is maintaining state then need only initialize it when IsPostBack is false

```
protected void Page_Load(Object s, EventArgs e) {
  if (! Page.IsPostBack) {
    // Executes only on initial page load
    Message.Text = "initial value";
  }
  // Rest of procedure executes on every request
}
```

Programming Basics
Server Control Events

- Change Events
  - By default, these execute only on next action event
  - E.g. `OnTextChanged`, `OnCheckedChanged`
  - Change events fire in random order
- Action Events
  - Cause an immediate postback to server
  - E.g. `OnClick`
- Works with any browser

Programming Basics
Wiring Up Control Events

- Control event handlers are identified on the tag

```
<asp:button onclick="btn1_Click“ runat=server>
<asp:textbox onchanged="text1_changed“ runat=server>
```

- Event handler code

```
protected void btn1_Click(Object s, EventArgs e) {
     ((Button)s).Text = “Button1 clicked”;
}
```

Programming Basics
Event Arguments

- Events pass two arguments:
  - The sender, declared as type `object`
    - Usually the object representing the control that generated the event
    - Allows you to use the same event handler for multiple controls
  - Arguments, declared as type `EventArgs`
    - Provides additional data specific to the event
    - `EventArgs` itself contains no data; a class derived from `EventArgs` will be passed.
      - E.g. `ServerValidateEventArgs`, `DataGridItemEventArgs`, `DataGridCommandEventArgs`

Programming Basics
Page Unloading

- Page_Unload fires after the page is rendered
  - Don't try to add to output
- Useful for logging and clean up

```
protected void Page_Unload(Object s, EventArgs e) {
  MyApp.LogPageComplete();
}
```

Programming Basics
`Import` Directive

- Adds code namespace reference to page
  - Avoids having to fully qualify .NET types and class names
  - Equivalent to the C# `using` directive

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Net" %>
<%@ Import Namespace="System.IO" %>
```

Programming Basics
Page Class

- The `Page` object is always available when handling server-side events
- Provides a large set of useful properties and methods, including:
  - `Application`, `Cache`, `Controls`, `EnableViewState`, `EnableViewStateMac`, `ErrorPage`, `IsPostBack`, `IsValid`, `Request`, `Response`, `Server`, `Session`, `Trace`, `User`, `Validators`
  - `DataBind()`, `LoadControl()`, `MapPath()`, `Validate()`

Agenda

- ASP.NET Overview
- Programming Basics
- **Server Controls**

Server Controls

- ASP.NET ships with ~50 built-in controls
- Organized into logical families
  - HTML controls
    - Controls / properties map 1:1 with HTML
  - Web controls
    - Richer functionality
    - More consistent object model

Server Controls
HTML Controls

- Work well with existing HTML designers
- Properties map 1:1 with HTML

```
table.bgcolor ="red“;
```

- Can specify client-side event handlers
- Good when quickly converting existing pages
- Derived from `System.Web.UI.HtmlControls.HtmlControl`
- Supported controls have custom class, others derive from `HtmlGenericControl`

Server Controls
HTML Controls

- Supported controls
  - <a>
  - <img>
  - <form>
  - <table>
  - <tr>
  - <td>
  - <th>
  - <select>
  - <textarea>
  - <button>
  - <input type=text>
  - <input type=file>
  - <input type=submit>
  - <input type=button>
  - <input type=reset>
  - <input type=hidden>

Server Controls
HTML Controls

- Can use controls two ways:
  - Handle everything in action events (e.g. button click)
    - Event code will read the values of other controls (e.g. text, check boxes, radio buttons, select lists)
  - Handle change events as well as action events

Server Controls
Web Controls

- Consistent object model

```
Label1.BackColor = Color.Red;
Table.BackColor = Color.Blue;
```

- Richer functionality
  - E.g. `AutoPostBack`, additional methods
- Contains controls for support
  - E.g. validation controls
- Strongly-typed; no generic control
  - Enables better compiler type checking

Server Controls
Web Controls

- Web controls appear in HTML markup as namespaced tags
- Web controls have an `asp:` prefix

```
<asp:button onclick="button1_click“ runat=server>
<asp:textbox onchanged="text1_changed“ runat=server>
```

- Defined in the `System.Web.UI.WebControls` namespace
- This namespace is automatically mapped to the `asp:` prefix

Server Controls
Web Controls

- Web Controls provide extensive properties to control display and format, e.g.
  - `Font`
  - `BackColor`, `ForeColor`
  - `BorderColor`, `BorderStyle`, `BorderWidth`
  - `Style`, `CssClass`
  - `Height`, `Width`
  - `Visible`, `Enabled`

Server Controls
Web Controls

- Four types of Web Controls
  - Basic controls
  - List controls
  - Rich controls
  - Validation controls

Server Controls
Basic Controls

- Correspond to HTML controls
- Supported controls
  - `<asp:button>`
  - `<asp:imagebutton>`
  - `<asp:linkbutton>`
  - `<asp:hyperlink>`
  - `<asp:textbox>`
  - `<asp:checkbox>`
  - `<asp:radiobutton>`
  - `<asp:image>`
  - `<asp:label>`
  - `<asp:panel>`
  - `<asp:table>`

Server Controls
Basic Controls

- Some controls such as `RadioButton` and `CheckBox` don't automatically do a postback when their controls are changed
- Specify `AutoPostBack=true` to make change events cause a postback

Server Controls
List Controls

- Controls that handle repetition
- Supported controls
  - `<asp:dropdownlist>`
  - `<asp:listbox>`
  - `<asp:radiobuttonlist>`
  - `<asp:checkboxlist>`
  - `<asp:repeater>`
  - `<asp:datalist>`
  - `<asp:datagrid>`

Server Controls
List Controls

- `Repeater`, `DataList` and `DataGrid` controls
  - Powerful, customizable list controls
  - Expose templates for customization
  - Can contain other controls
  - Provide event bubbling through their `OnItemCommand` event

Server Controls
CheckBoxList & RadioButtonList

- Provides a collection of check box or radio button controls
- Can be populated via data binding

```
<asp:CheckBoxList id=Check1 runat="server">
  <asp:ListItem>Item 1</asp:ListItem>
  <asp:ListItem>Item 2</asp:ListItem>
  <asp:ListItem>Item 3</asp:ListItem>
  <asp:ListItem>Item 4</asp:ListItem>
  <asp:ListItem>Item 5</asp:ListItem>
</asp:CheckBoxList>
```

Server Controls
Rich Controls

- Custom controls with rich functionality
- Supported Controls
  - `<asp:calendar>`
  - `<asp:adrotator>`
- Demo: Rich Controls

Server Controls
Validation Controls

- Rich, declarative validation
- Validation declared separately from input control
- Extensible validation framework
- Supports validation on client and server
  - Avoids roundtrips when validating on client
- Server-side validation is always done
  - Prevents users from spoofing Web Forms

Server Controls
Validation Controls

- `<asp:RequiredFieldValidator>`
  - Ensures that a value is entered
- `<asp:RangeValidator>`
  - Checks if value is within minimum and maximum values
- `<asp:CompareValidator>`
  - Compares value against constant, another control or data type
- `<asp:RegularExpressionValidator>`
  - Tests if value matches a predefined pattern
- `<asp:CustomValidator>`
  - Lets you create custom client- or server-side validation function
- `<asp:ValidationSummary>`
  - Displays list of validation errors in one place

Server Controls
Validation Controls

- Validation controls are derived from `System.Web.UI.WebControls.BaseValidator`, which is derived from the `Label` control
- Validation controls contain text which is displayed only if validation fails
- `Text` property is displayed at control location
- `ErrorMessage` is displayed in summary

Server Controls
Validation Controls

- Validation controls are associated with their target control using the `ControlToValidate` property

```
<asp:TextBox id=TextBox1 runat=server />

<asp:RequiredFieldValidator id="Req1"
  ControlToValidate="TextBox1"
  Text="Required Field" runat=server />
```

- Can create multiple validation controls with the same target control

Server Controls
Validation Controls

- `Page.IsValid` indicates if all validation controls on the page succeed

```
void Submit_click(object s, EventArgs e) {
  if (Page.IsValid) {
    Message.Text = "Page is valid!";
  }
}
```

Server Controls
Validation Controls

- `Display` property controls layout
  - `Static`: fixed layout, display won't change if invalid
  - `Dynamic`: dynamic layout
  - `None`: no display; can still use `ValidationSummary` and `Page.IsValid`
- `Type` property specifies expected data type: `Currency`, `Date`, `Double`, `Integer`, `String`

Server Controls
Validation Controls

- Demo: Validation