6. Server-Side Coding

a. What is a Server?

b. Server Architectures

c. Why Server-Side Coding?

d. Server-Side Technologies

e. Overview of ASP

f. Introduction to ASP.NET and Web Forms

What is a Server?

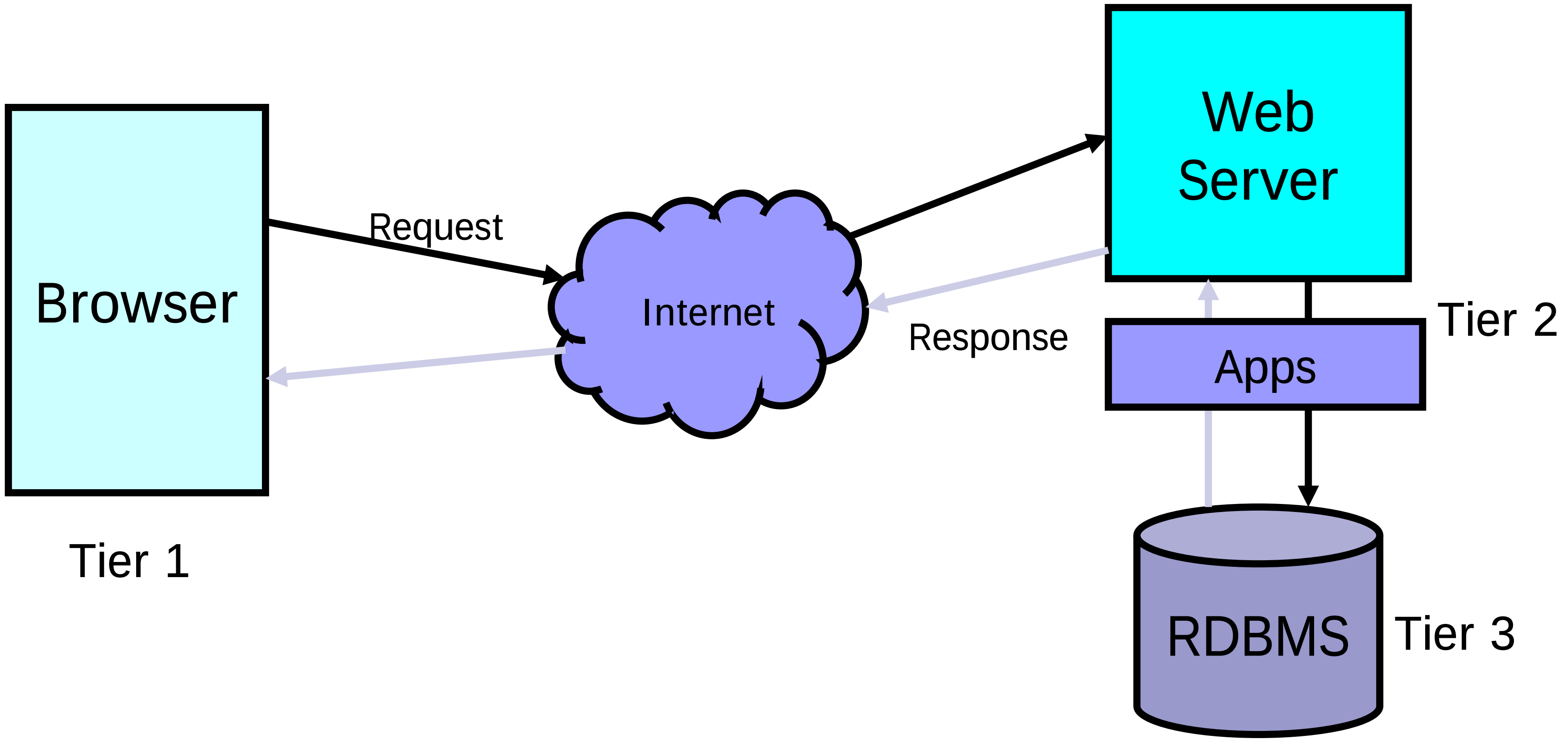
- Server
  - A computer (or the software that runs on it) that provides services to others
- Many types of server

File server	file: networked file space
FTP server	ftp: remote file space, often read-only
Web server	http: web pages and more
Mail server	mail: email system
News server	news: newsgroups messages

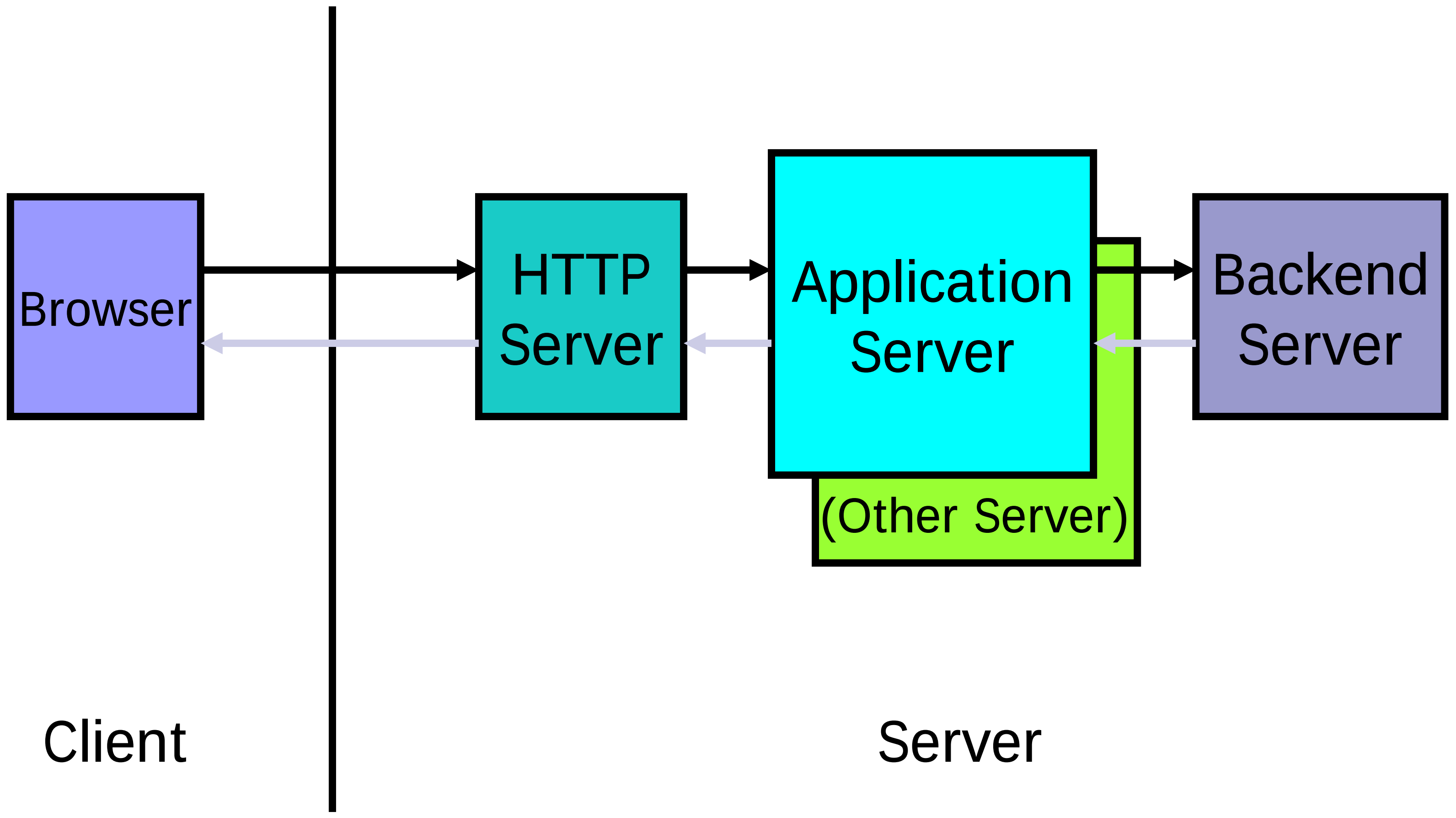
Application Architectures

- Multi-tier applications divide functionality into separate tiers (i.e., logical groupings of functionality)
- Tiers can be located on the same computer or on separate computers
- A three-tier architecture consists of
  - The client/top tier
  - Middle-tier
  - Information/data/bottom-tier
- The client-tier is the application’s user interface
- The middle-tier implements business logic and presentation logic to control interactions between application clients and application data
- The bottom-tier maintains data for the application
  - Typically stores data in relational DBMS

Dynamic, Interactive Web Server (3-tier Architecture)



General Multi-tier Architecture



Why Server-Side Coding?

- Accessibility
  - You can reach the Internet from any browser, any device, any time, anywhere
- Manageability
  - Does not require distribution of application code
  - Easy to change code
- Security
  - Source code is not exposed
  - Once user is authenticated, can only allow certain actions
- Scalability
  - Web-based 3-tier architecture can scale out
    - If bottlenecks in terms of performance occur, the server process can be moved to other servers at runtime

Server-Side Technologies

- Scripting Languages:
  - Server Side Includes (SSI)
  - Perl
  - PHP
  - ASP (VBScript)
  - Python
- Compiled Languages:
  - C
  - C++
  - C#
  - ASP .Net
  - Java Servlets
  - Java Server Pages (JSP)
    - Looks like a scripting language, but is actually compiled into a Java Servlet
- Either portable byte code (such as a Java .class file) or a true executable (native to the microprocessor) is produced
- Common to all scripting languages is some sort of real time interpreter that parses text and turns it into executable instructions for the server

Which Technologies Should You Choose?

- Some criteria affecting decisions
  - Web server availability
  - Knowledge of language
  - Scalability and efficiency
  - Personal preference

What is ASP?

- ASP stands for **A**ctive **S**erver **P**ages
  - ASP is a program that runs inside **IIS (I**nternet **I**nformation **S**ervices)
  - ASP is a server-side programming technology developed by Microsoft
- **What is an ASP File?**
  - An ASP file (with ".asp" extension) can contain text, HTML, XML, and scripts
    - An ASP file can also contain **server scripts**, surrounded by the delimiters **<%** and **%>**.
  - Server scripts are executed on the server, and can contain any expressions, statements, procedures, or operators valid for the scripting language you prefer to use.
- **How Does ASP Differ from HTML?**
  - When a browser requests an HTML file, the server returns the file
  - When a browser requests an ASP file, IIS passes the request to the ASP engine.
    - The ASP engine reads the ASP file, line by line, and executes the scripts in the file. Finally, the ASP file is returned to the browser as plain HTML

ASP Intrinsic Objects

- Six built-in objects in the ASP environment used to provide services:
  - Request
    - Used to access information passed by an HTTP request (get information from users)
  - Response
    - Used to control information sent to the client (send information to users)
  - Server
    - Provides access to methods and properties on the server
  - Application
    - Used to hold information that can be used by many pages (e.g., DB connections)
  - Session
    - Used to maintain information about a user session
    - `Application` variables store info for all users while `Session` variables store info about a single user
  - ObjectContext
- Commonly uses ADO to interact with databases

Example 1: Salam Shabab

```
<%@ language="javascript"%>
<html>
<head><title>Salam.asp</title></head>
<body>
<%
Response.Write(“Salam Shabab!")
%>
</body>
</html>
```

Example 2: Salam Shabab

```
<%@ language="javascript"%>
<html>
<head><title>Salam2.asp</title></head>
<body>
<form method="post">
<input type="submit" id="bt" name="bt" value="Push Me"
   />
<%
if (Request.Form("bt”) != "")
  Response.Write("<p>Salam, the time is " +
   Now()+”</p>”);
%>
</form>
</body>
</html>
```

Server-Side Form Processing

- We have seen examples of client-side processing in earlier sessions
  - Where all processing is local to the Web page, encapsulated within browser scripts or event handlers.
- Form controls can be used for submitting information from a Web page to a processing program located on the Web server.
  - In this case, the controls to capture information are surrounded by a **`<form>`** tag containing **`action`** and **`method`** attributes.
  - **`action`** gives the URL of the external page that handles the processing
  - **`method`** indicates how the form information is transmitted (normally through the **`post`** method).
  - The form includes
    - a "submit" button to activate the form submission
    - an optional "reset" button can be coded to automatically clear all form fields.

Example 3: Handling User Input

```
<form method="get“
  action="simpleform.asp">
First Name:
  <input type="text" name="fname">
 <br />
Last Name:
  <input type="text" name="lname">
<br /><br />
<input type="submit" value="Submit">
</form>
```

… Handling User Input

- User input can be retrieved in two ways: With `Request.QueryString` or `Request.Form`.
- The `Request.QueryString` command is used to collect values in a form with `method="get"`.
  - Information sent from a form with the GET method is visible to everyone (it will be displayed in the browser's address bar) and has limits on the amount of information to send.
- If a user typed “`Feras"` and “`Nabulsi"` in the form example above, the URL sent to the server would look like this:
  - http://www.w3schools.com/simpleform.asp?fname=Feras&lname=Nabulsi
- The `Request.Form` command is used to collect values in a form with `method="post"`.
  - Information sent from a form with the POST method is invisible to others and has no limits on the amount of information to send.
- If a user typed “Feras" and “Nabulsi" in the form example above, the URL sent to the server would look like this:
  - http://www.w3schools.com/simpleform.asp

… Handling User Input

- Assume `simpleform.asp` has the following simple script:

```
<body> Welcome
<%
  response.write(request.querystring("fna
  me"));
    response.write(" " +
  request.querystring("lname"));
 %> </body>
```

- For both cases above, the browser will display:

```
Welcome Feras Nabulsi
```

ASP Challenges

- Coding overhead (too much code)
  - Everything requires writing code!
- Code readability (too complex; code and UI intermingled)
- Maintaining page state requires more code
  - After submit button is clicked, if we click the back button, we expect to maintain scroll position, maintain which control had focus, and restore focus, or allow server code to focus a new control
- Reuse is difficult – lack of modularity
- Supporting many types of browsers is difficult – ASP.NET has better browser support
- Deployment issues (e.g. DLL locking) – ASP.NET easier to deploy
- Session state scalability and availability – ASP uses cookies to maintain state while ASP.NET uses cookieless means

Introduction to ASP.NET and Web Forms

1. ASP.NET Overview
2. Programming Basics
3. Server Controls
4. Data Binding
5. Conclusion

ASP.NET Overview

- ASP.NET provides services to allow the creation, deployment, and execution of Web Applications and Web Services
- Like ASP, ASP.NET is a server-side technology
- Web Applications are built using Web Forms
- Web Forms are designed to make building Web-based applications easy

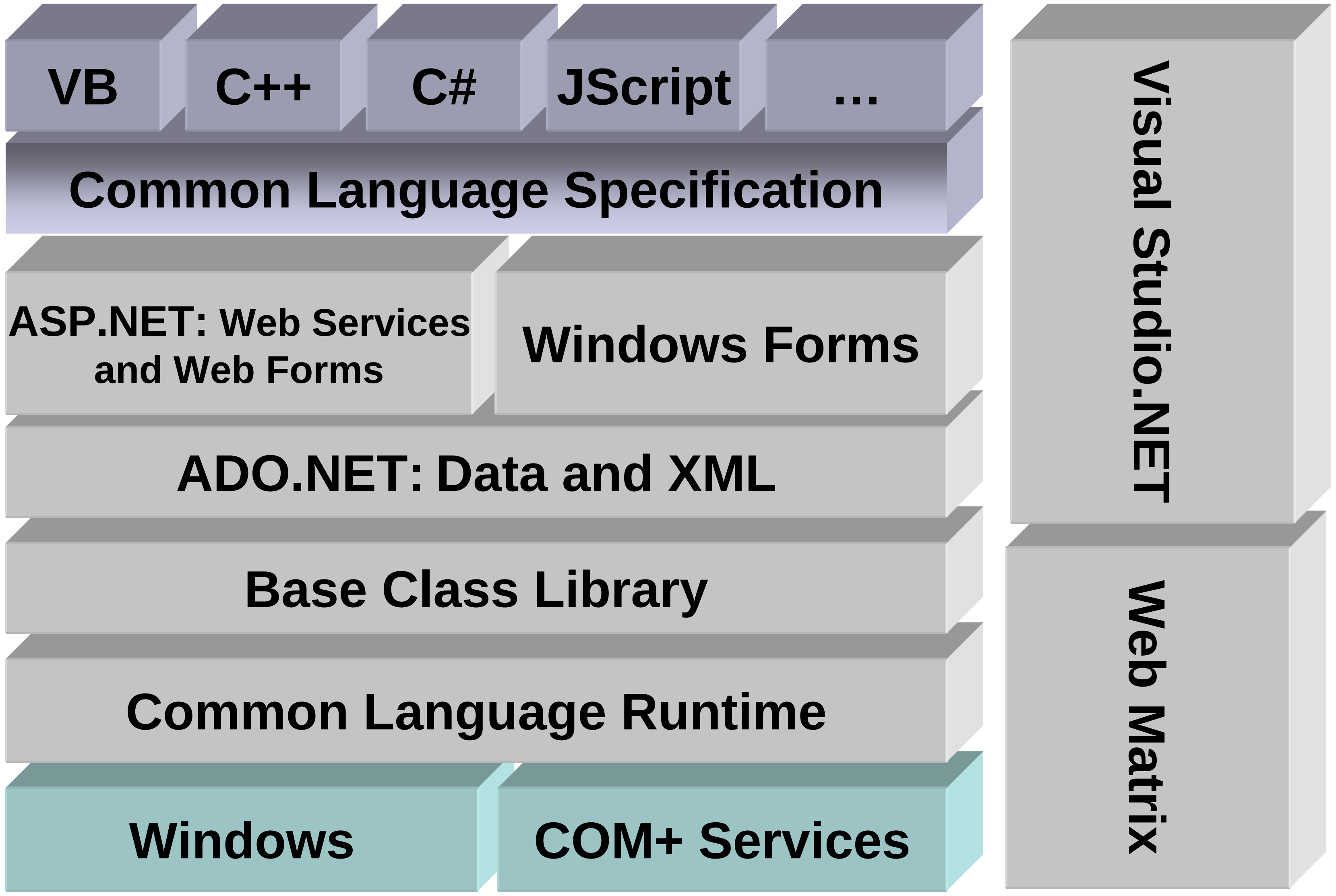
Key Features

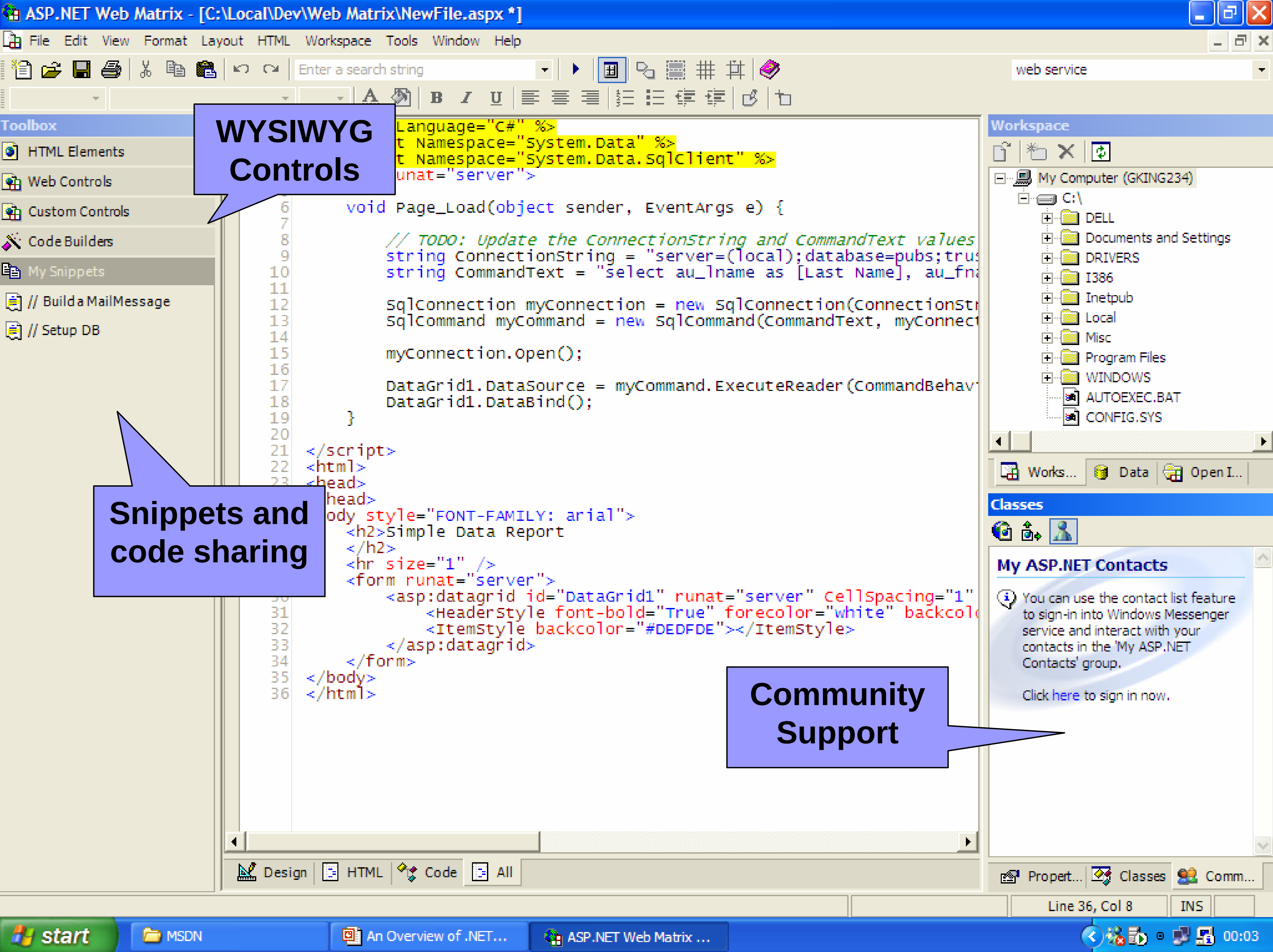
- Built on .NET framework
  - Supports C++, C#, Visual Basic, and JScript (Microsoft's version of JavaScript)
- Simple programming model
  - Complete object model
- Maintains page state
- XML configuration
- Separation of code and UI
- Security
- Simplified form validation

Development Environments

- Visual Studio.NET
  - Installed in college labs
  - Available on CD from PC Admin support
- ASP .NET Web Matrix
  - Lightweight, simple, community-oriented tool for building ASP.NET apps
  - Full WYSIWYG support
  - Small (~ 1.4 Mb)
  - Community features
    - IM integration, code sharing, chat features
  - Available free-of-charge at www.asp.net

Architecture





```
Language="C#" %>
t Namespace="System.Data" %>
t Namespace="System.Data.SqlClient" %>
unat="server">

    void Page_Load(object sender, EventArgs e) {

        // TODO: Update the ConnectionString and CommandText values
        string ConnectionString = "server=(local);database=pubs;trus
        string CommandText = "select au_lname as [Last Name], au_fna

        SqlConnection myConnection = new SqlConnection(ConnectionStr
        SqlCommand myCommand = new SqlCommand(CommandText, myConnect

        myConnection.Open();

        DataGrid1.DataSource = myCommand.ExecuteReader(CommandBehav
        DataGrid1.DataBind();
    }

</script>
<html>
<head>
head>
ody style="FONT-FAMILY: arial">
    <h2>Simple Data Report
    </h2>
    <hr size="1" />
    <form runat="server">
        <asp:datagrid id="DataGrid1" runat="server" CellSpacing="1"
            <HeaderStyle font-bold="True" forecolor="white" backcol
            <ItemStyle backcolor="#DEDFDE"></ItemStyle>
        </asp:datagrid>
    </form>
</body>
</html>
```

My ASP.NET Contacts

You can use the contact list feature to sign-in into Windows Messenger service and interact with your contacts in the 'My ASP.NET Contacts' group.

Click here to sign in now.

Programming Basics

- First ASP.NET Example
- Page Syntax
- Server Controls
- Code Blocks
- Data Bind Expressions
- Render Code

Salam.aspx

```
<%@ Page Language="c#" %>
<script runat="server">
    public void B_Click (object sender, System.EventArgs e) {
       Label1.Text = "Salam, the time is " + DateTime.Now;
    }
</script>
<html>
<head>
</head>
<body>
    <form method="post" runat="server">
        <asp:Button id="Button1" onclick="B_Click" runat="server"
   Text="Push Me" name="Button1"></asp:Button>
        <p>
            <asp:Label id="Label1" runat="server"></asp:Label>
        </p>
    </form>
</body>
</html>
```

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 bind expressions: `<%# %>`
  - Server side comments: `<%-- --%>`
  - Render code: `<%= %>` and `<% %>`
    - Use is discouraged; use `<script runat="server">` with code in event handlers instead

The Page Directive

- Directives are commands used by the compiler when the page is compiled
  - The `Page` directive is most frequently used directive
- Lets you specify page-specific attributes, e.g.
  - `AspCompat`: Compatibility with ASP
  - `CodePage`: Code page for this .aspx page
  - `Inherits`: Base class of Page object
  - `Language`: Programming language
  - `Trace`: Enables tracing for this page
- Only one `Page` directive per .aspx file

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
  - Controls are derived from `System.Web.UI.Control`
- The id attribute provides programmatic identifier
  - It names the instance available during postback

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;
```

Maintaining State

- By default, controls maintain their state across multiple postback requests
  - A postback occurs when a page generates an HTML form whose values are posted back to the same page
  - 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`

Server Code Blocks

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

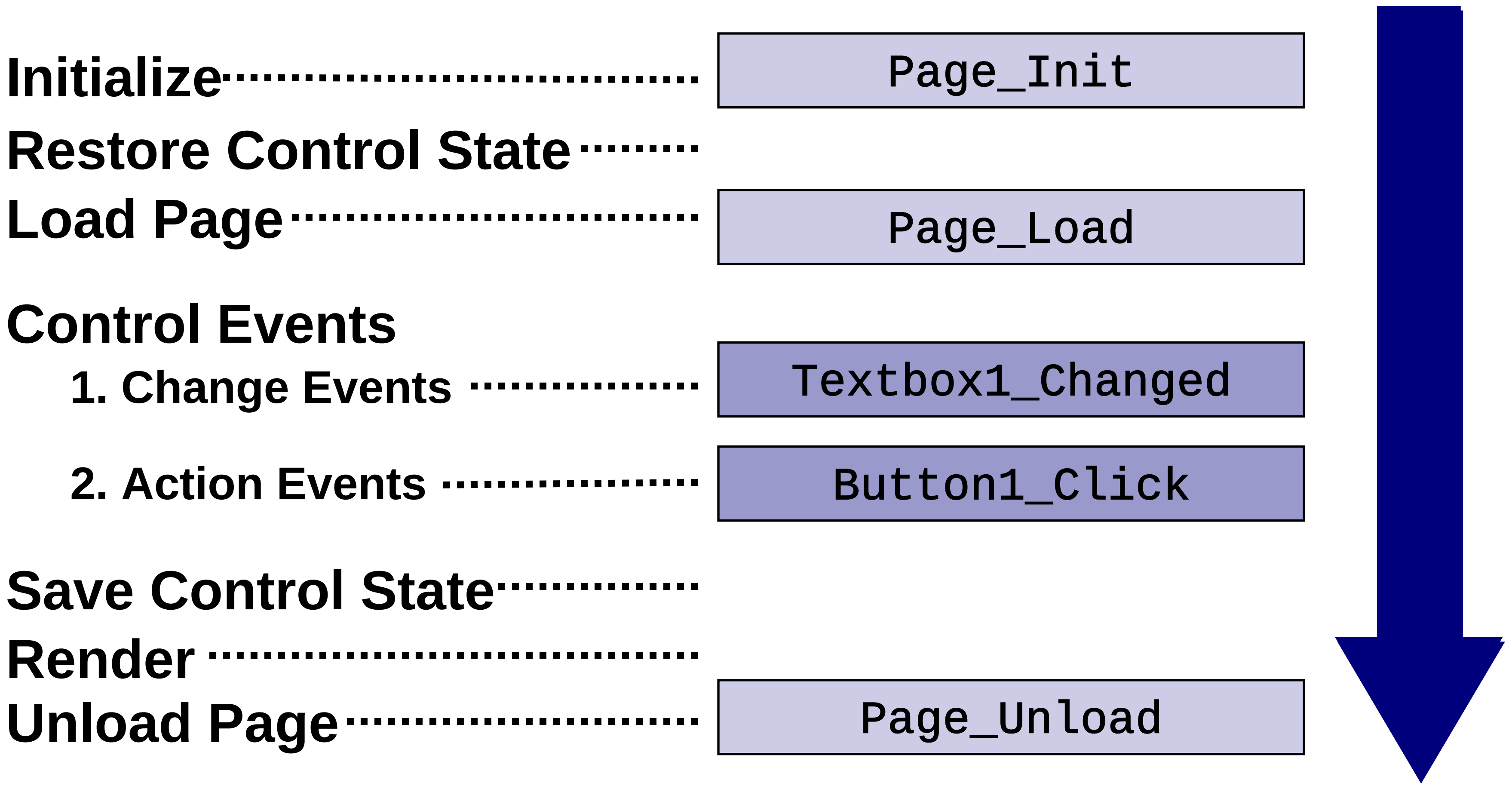
```
<script language="C#" runat="server">
<script language="VB" runat="server">
<script language="JScript" runat="server">
```

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

Page Events

- Pages are structured using events
  - Enables clean code organization
  - Avoids the “Monster IF” statement
  - 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`

Page Event Lifecycle



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;
}
```

Page Loading

- `Page_Load` fires on every request
  - Use `Page.IsPostBack` to execute conditional logic

```
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
}
```

Server Control Events

- Change Events
  - By default, these execute only on next action event
    - Use `autopostback="true"` atribute to make them respond directly without waiting for an action event
  - E.g. `OnTextChanged, OnCheckedChanged`
  - Change events fire in random order
- Action Events
  - Cause an immediate postback to server
  - E.g. `OnClick`

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) {
  Message.Text = "Button1 clicked";
}
```

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

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();
}
```

Import Directive

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

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

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()`

9

Introduction to ASP.NET and Web Forms

5. Server Controls

Server Controls

- ASP.NET ships with ~50 built-in controls
- Organized into two logical families
  - HTML controls
    - Controls / properties map 1:1 with HTML
  - Web controls
    - Richer functionality
    - More consistent object model
- Use Server controls when
  - You require a richer set of functionality to perform complicated page requirements
  - Developing pages for multiple browser types

HTML Controls

- Work well with existing HTML designers
  - i.e., when converting existing ASP 3.0 Web pages to ASP.NET Web pages
  - Easier to change to HTML server controls than to Web server controls
- 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`
  - HtmlAnchor controls the <a> element
  - HtmlButton controls the <button> element
  - HtmlForm controls the <for> element
  - etc

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>`

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

HTML Controls

- Demo 1: HTMLControls1.aspx
  - Basic page lifecycle with HTML Controls
- Demo 2: HTMLControls2.aspx
  - More HTML 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

Web Controls

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

Web Controls

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

Intrinsic 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>`

Intrinsic Controls

- `TextBox, ListControl, CheckBox` and their subclasses don’t automatically do a postback when their controls are changed
- Specify `AutoPostBack=“true”` to make change events cause a postback

List Controls

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

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
  - More about these controls and templates later

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>
```

Intrinsic & Simple List Controls

- Demo 1: WebControls1.aspx
  - Assorted intrinsic and list controls
- Demo 2: WebControls2.aspx
  - Same controls with `AutoPostBack`

Rich Controls

- Custom controls with rich functionality
- Supported Controls
  - `<asp:calendar>`
  - `<asp:adrotator>`
- More will be added
- 3rd party controls are coming
- Demo: RichControls1.aspx

Validation Controls

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

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

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

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

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!";
  }
}
```

Validation Controls

- `Display` property controls how the error message is shown
  - `Static`: fixed layout, display won't change if invalid; Space for the validation message is allocated in the page layout
  - `Dynamic`: Space for the validation message is dynamically added to the page if validation fails
  - `None`: Validation message is never displayed; can still use `ValidationSummary` and `Page.IsValid`
- Type property specifies expected data type: `Currency`, `Date`, `Double`, `Integer`, `String`

Validation Controls

- Can force down-level option
  - Only server-side validation

```
<% @ Page Language="c#"
          ClientTarget="DownLevel" %>
```

Validation Controls

- Demo: [ValidationControls1.aspx]
  - Demonstrates each type of validation control

9

Introduction to ASP.NET and Web Forms

6. Data Binding

How to Populate Server Controls?

- Specify the data in the control's tags
  - Not dynamic: can't get data from a database
- Write code that uses the control's object model
  - This is okay if you need to populate a simple value or list, but quickly gets too complicated for populating sophisticated displays
- Data binding (a better solution than the previous two methods)
  - Create an object that holds the data (`DataSet`, `Array`, `string`, `int`, etc.)
  - Associate that object with the control

What Is Data Binding?

- One of the most useful features of ASP.NET
- Allows programmers to easily display and manipulate data without dealing with the underlying data structure
- Provides a single simple yet powerful way to populate Web Form controls with data
  - Enables clean separation of code from UI (control separate from object holding data)
- Supports binding to any data source
  - Properties, expressions, method calls
  - Collections (`Array, Hashtable`, etc.)
  - `DataSet, DataTable, DataView, DataReader`
  - XML

… What Is Data Binding?

- Controls that support data binding expose a property named `DataSource` and a method called `DataBind`
- When a page is loaded, the user of the control initializes the `DataSource` property to some collection of data like an array, `DataReader` or `DataSet`
- When the `DataBind` method of the control is called, the expression is evaluated and bound
  - `DataBind` for a single control (and subcontrols)
  - `Page.DataBind` binds all controls on a page
- Works for scalars, e.g. `Label` control
- Works for lists, e.g. `DropDown` control, `ListBox` control, etc.
- Enables the use of templates

Scalar Expressions

- Data binding expression: `<%# expression %>`
- Expression is evaluated when `DataBind()` is called

```
<asp:Label id=label1
  Text=<%# "The result is " + (1 + 2) + ", the time is " +
DateTime.Now.ToLongTimeString() %> runat="server" />

public void Page_Load(object s, EventArgs e) {
  if (! Page.IsPostBack)
    Page.DataBind();
}
```

Scalar Expressions

- Demo: DataBinding1.aspx
  - Data binding to simple, scalar expressions

Simple Lists

- Data binding a list creates a user interface element for each item in the list
- Each item contains text (displayed to user) and an optional value (not displayed)
- The simple list controls:
  - <asp:ListBox>
    - Single or multiple select
  - <asp:DropDownList>
  - <asp:RadioButtonList>
  - <asp:CheckBoxList>
- The above controls can be populated with XML data or with the following objects:
  - ArrayList
  - Hashtable
  - SortedList

Simple Lists

- Steps to data bind a list control
  - Declare the list control
  - Optionally set `DataValueField and DataTextField`
  - Set its `DataSource`
  - Call `DataBind()` method
- Notes:
  - Use `ArrayList` to populate controls with `Text` and `Value` fields to have the same name
  - Use `HashTable` or `SortedList` object to add values that different for `Text`
  - E.g.,:

```
<asp:RadioButtonList id="countryList"
   runat="server">
    <asp:ListItem value="NG" text="Nigeria" />
</asp:RadioButtonList>
```

Simple Lists

- Demo: `DataBinding2.aspx`
    - Data binding to simple lists

Binding to Database Sources

- We have seen that it is possible to bind to many different types of collections in ASP.NET
- The most common type of binding is to bind to a result set retrieved from a database query
- ADO.NET provides two ways of retrieving result sets from a database:
  - Using the streaming `IDataReader` interface (or `DataReader` class)
  - Using the disconnected `DataSet` class
- Binding to database sources requires working with four main categories of objects:
  - Connections
    - Establishes a Web page connection to a specified data source.
  - Commands
    - Defines and executes a command, normally an SQL statement, to retrieve or update a set of records in the data source.
  - DataReader
    - Reads through a recordset retrieved by the Command object.
  - DataAdapter
    - Populates an in-memory data store with records from a data source, and manages updates to the source.

List Binding Examples

```
public void Page_Load(object s, EventArgs e) {
   if (! Page.IsPostBack) {

    // Databind a drop down to a DataReader

    DropDown1.DataSource = GetSampleDataDR();
    DropDown1.DataValueField = "CategoryID";
    DropDown1.DataTextField = "CategoryName";
    DropDown1.DataBind();

    // Databind a radio button list to a DataView

    RadiobuttonList1.DataSource = GetSampleDataDV();
    RadiobuttonList1.DataValueField = "CategoryID";
    RadiobuttonList1.DataTextField = "CategoryName";
    RadiobuttonList1.DataBind();
  }
}
```

Binding to a DataReader

```
IDataReader GetSampleDataDR() {

    SqlConnection cxn =
   new SqlConnection("Data Source=localhost;Initial
               Catalog=Northwind;Integrated Security=True");

  SqlCommand cmd = new SqlCommand("select CategoryID,
                        CategoryName from Categories", cxn);
    cxn.Open();

    SqlDataReader dr =
          cmd.ExecuteReader(CommandBehavior.CloseConnection);

    return dr;
}
```

Binding to DataView of DataSet

```
DataView GetSampleDataDV() {

  DataSet ds;

  SqlConnection cxn =
    new SqlConnection("Data Source=localhost;Initial
             Catalog=Northwind;Integrated Security=True");

SqlDataAdapter adp =
     new SqlDataAdapter("select CategoryID,
                   CategoryName from Categories", cxn);

  ds = new DataSet();

  adp.Fill(ds, "Categories");

  return ds.Tables["Categories"].DefaultView;
}
```

Binding to a Database

- Demo: DataBinding3.aspx
  - Data binding to a database

DataReader Vs DataSet for Data Binding

DataReader	DataSet
Supports a connected, read-only access style	Supports a disconnected access style
Programmer must explicitly open and close the `sqlConnection`	`SqlDataAdapter` automatically handles opening/closing connection
Does not cache query results	Makes a local copy of and manipulates query results, then reconcile with the actual DB later
Preferred for Web applications where short operations (displaying data) are usually performed	Suitable for long-running applications
Focuses on data access and presentation	Focuses on data management
Use when you simply need to read data and immediately bind it to a control	Use when binding one set of data to multiple controls using different `DataViews`

DataGrid

- A control that displays data from a data source (e.g., a database) in the form of a table
- Full-featured list output
- Default look is a grid
- Default is to show all columns, though you can specify a subset of columns to display
- Columns can be formatted with templates
- Optional paging
- Updateable

Binding to All Columns

- Binding all columns in the `datasource`
  - Declare an `<asp:DataGrid>`
  - Set its `DataSource`
  - Call `DataBind()`

```
void Page_Load(object s, EventArgs e) {
  myDataGrid.DataSource = GetSampleData();
  myDataGrid.DataBind();
}

<asp:datagrid id="myDataGrid" runat="server" />
```

Binding to Specific Columns

- By default, `DataGrid` will display all columns
- To control columns to display:
  - Set `AutoGenerateColumns`="`false`"
  - Specify Columns property
  - Add column definition
    - `BoundColumn`
    - `TemplateColumn`
    - `etc`

Binding to Specific Columns

- Binding to specific columns in the `datasource`
  - Declare an <asp:DataGrid>
  - Declare its Columns collection
  - Set its DataSource
  - Call its DataBind() method

```
<asp:datagrid id="myDataGrid"
       autogeneratecolumns="false" runat="server">
 <Columns>
  <asp:BoundColumn HeaderText="Id" DataField="title_id" />
  <asp:BoundColumn HeaderText="Title" DataField="title"/>
 </Columns>
</asp:datagrid>
```

`DataGrid` Paging

- When there is too much data to display in one screen, a `DataGrid` can provide automatic paging
    - Set `AllowPaging`="`true`"
    - Set `PageSize=5`
    - Handle `OnPageIndexChanged` event
        - Set page index
        - Fetch data
        - Re-bind data

DataGrid Demo

- Demo: DataBinding4.aspx
  - Binding to a database with `DataGrid`
- Demo: DataBinding5.aspx
  - Paging through data with `DataGrid`

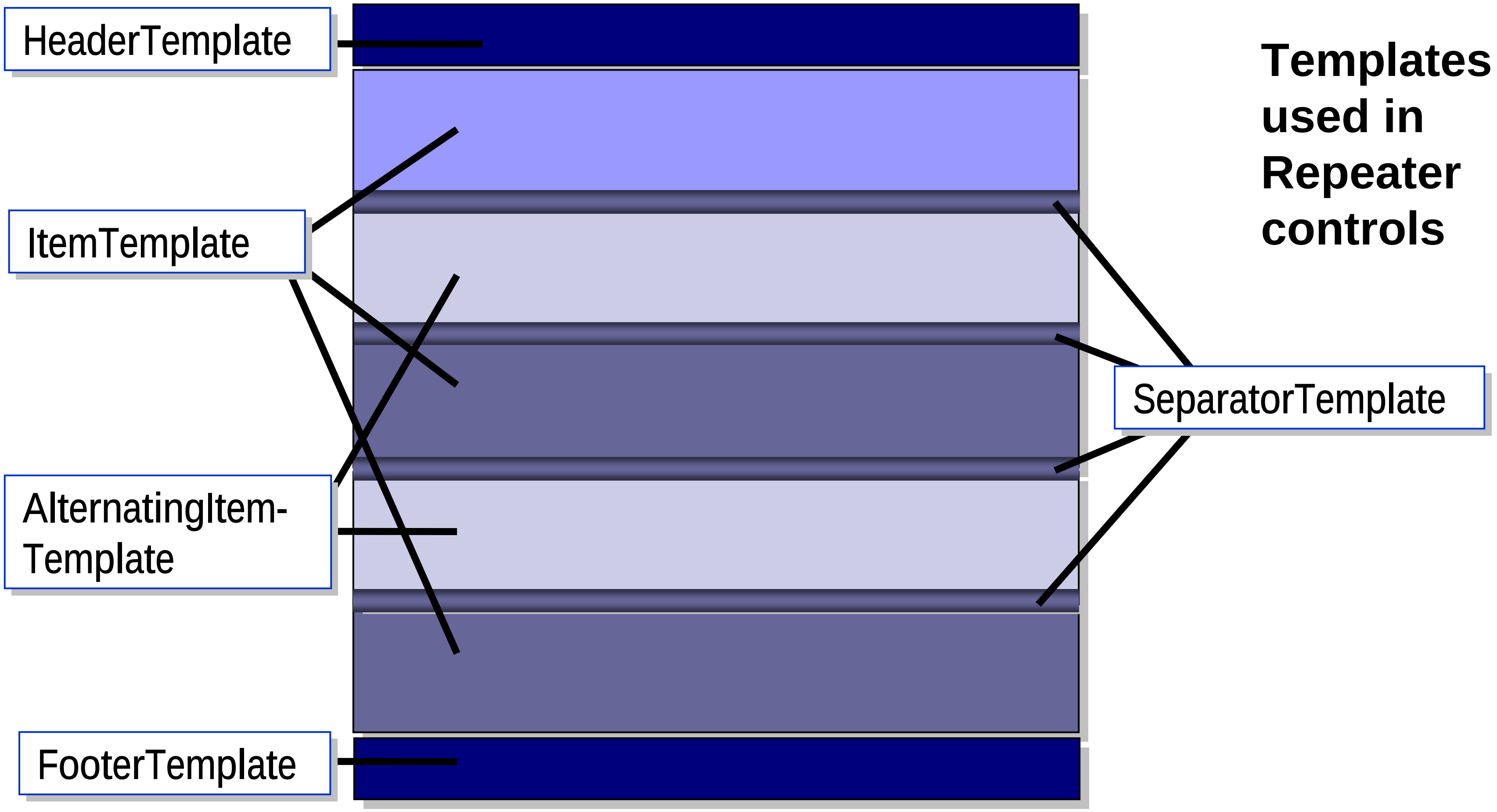
Templates

- Templates provide a powerful way to customize the display of a server control
  - Customize structure – not just style
  - Can use controls or other HTML within a template
  - 3rd party controls can expose new templates
- With data binding, templates specify a set of markup (HTML or server controls) for each bound piece of data
  - Not just specifying formatting and style for a column
- However, templates are not limited to data binding
  - No fixed set of templates
  - Controls may define their own and expose any number of them

Templates

- Standard templates for list-bound controls
  - `HeaderTemplate`: rendered once before all data bound rows
  - `ItemTemplate`: rendered once for each row in the data source
  - `AlternatingItemTemplate`: like `ItemTemplate`, but when present is used for every other row
  - `SeparatorTemplate`: rendered between each row
  - `FooterTemplate`: rendered once, after all data bound rows

Templates



Templates used in Repeater controls

Data Binding in Templates

- Templates need to access the bound data
- `Container` is an alias for the template's containing control
- `DataItem` is an alias for the current row of the datasource
- `DataBinder.Eval` is a utility function provided to retrieve and format data within a template

```
<%# DataBinder.Eval(Container.DataItem, "price", "$ {0}") %>
```

Repeater and DataList Controls

- Repeater and `DataList` controls are templated controls
    - They give greater flexibility over the rendering of list-like data
- The `repeater` control is just iterates over the bound data, rendering the `ItemTemplate` once for each item in the `DataSource` collection
    - A general-purpose iterator
    - Does not render anything beside the elements contained in the template
- `DataList` renders additional elements, like table rows and cells and spans containing style attributes
    - Enables end-users to exactly control the structure and layout of each item within the list (using the `ItemTemplate` property)

Repeater Control

- Provides simple output of a list of items
- No inherent visual form (DataGrid supports tabular visual form by default)
- Templates provide the visual form
- No paging (DataGrid supports paging)
- Can provide templates for separators
- Not updateable (DataGrid is updateable)

Repeater Control

```
<asp:Repeater id="repList" runat="server">
<template name="HeaderTemplate">
  <table>
  <tr><td>Title</td><td>Type</td></tr>
</template>

<template name="ItemTemplate">
 <tr>
   <td><%# DataBinder.Eval(Container.DataItem,"title_id") %></td>
   <td><%# DataBinder.Eval(Container.DataItem,"type") %></td>
 </tr>
</template>

<template name="FooterTemplate">
  </table>
</template>
</asp:Repeater>
```

`DataList` Control

- Provides list output with editing
- Default look is a table
- Customized via templates
- Directional rendering (horizontal or vertical)
  - Using `RepeaterColumns` and `RepeaterDirection` (similar to `background-repeat` in CSS)
- Updateable
- No paging

DataList Control

```
void Page_Load(object s, EventArgs e) {
  myDataGrid.DataSource = GetSampleData();
  myDataGrid.DataBind();
}

<asp:datalist id="myDataList" runat="server">
  <template name="itemtemplate">
  <b>Title id:</b>
  <%# DataBinder.Eval(Container.DataItem, "title_id") %>
  <br> <b>Title:</b>
  <%# DataBinder.Eval(Container.DataItem, "title") %>
  </template>
</asp:datalist>
```

Templates Demo

- Demo: DataBinding6.aspx, DataBinding7.aspx
  - Using templates and data binding to a database with `DataGrid`, `Repeater` and `DataList` controls

Running Our DB-Based Data Binding Examples

- Many of those examples use the sample databases shipped with SQL Server 2000
  - The PUBS and NORTHWIND databases, specifically
- Using SQL Server 2000
  - There should be no problems, hopefully
  - Otherwise you may need to
    - Locate the example databases in the install directory and add them to your application
      - See next slide
    - Change the connection string
      - See next slide

Adding a DB to a Website Project

- Open your Website project in Visual Studio 2005 (for example)
- Select the Solutions Explorer, if necessary, to expose the Website's files (in the right panel of the VS 2005 window)
- Right-click on the App_Data folder of your Website (in the Solution Explorer) and select "Add Existing Item …".
  - Browse the file system to locate database and add it
    - You may find it somewhere like **C:\Program Files\Microsoft SQL Server\MSSQL\Data**
  - If you downloaded the example databases separately and run the .msi file, they are likely to be placed in **C:\SQL Server 2000 Sample Databases**

Changing the Connection String

- A common problem in connecting to DBs from your application is due to using the wrong connection string
- A typical error due to wrong connection string looks like:
- Note that when you select the DB in your Website, the Properties window below the Solution Explorer's window displays a Connection String property of the DB
  - From my experience, connecting to the DB using this connection string always fails!
- I fix the problem as follows:
  - Open an .aspx file in my Website
  - Make sure the .aspx file contains a control requiring a datasource, like DataGrid. Otherwise add such a control to the file (you may delete it after following the next step)
  - View the .aspx page in Design view
  - Select the DataList control, right-click on it and select "Show Smart Tag"
  - Select "New Data Source" in the "Select Data Source" dropdown from the resulting pop-up
  - Select "Database" from the resulting "Data Source Configuration Wizard" and click OK
  - Select the database added to your application from the dropdown of the resulting popup window
  - Expand the "connection string" tab on this window to reveal the correct connection string you need to connect to the database.

Running Our DB-Based Data Binding Examples

- Using SQL Server 2005
  - I have had more problems with this than with SQL Server 2000
  - You may need to download the example databases
    - http://www.microsoft.com/downloads/details.aspx?FamilyID=06616212-0356-46A0-8DA2-EEBC53A68034&displaylang=en
    - Install them and add them to the appropriate examples
  - You will need to change the connection string
    - As explained in the preceding slides

Resources

- http://msdn.microsoft.com/net/aspnet/default.asp
- http://www.asp.net/
- http://www.asptoday.com/
- http://www.aspfree.com/
- http://www.devx.com/dotnet/
- http://msdn.microsoft.com
- ASP.NET QuickStart Tutorial
  - http://samples.gotdotnet.com/quickstart/aspplus/
- W3 Schools ASP .NET Tutorial
  - http://www.w3schools.com/aspnet/default.asp
- Several Online Presentations

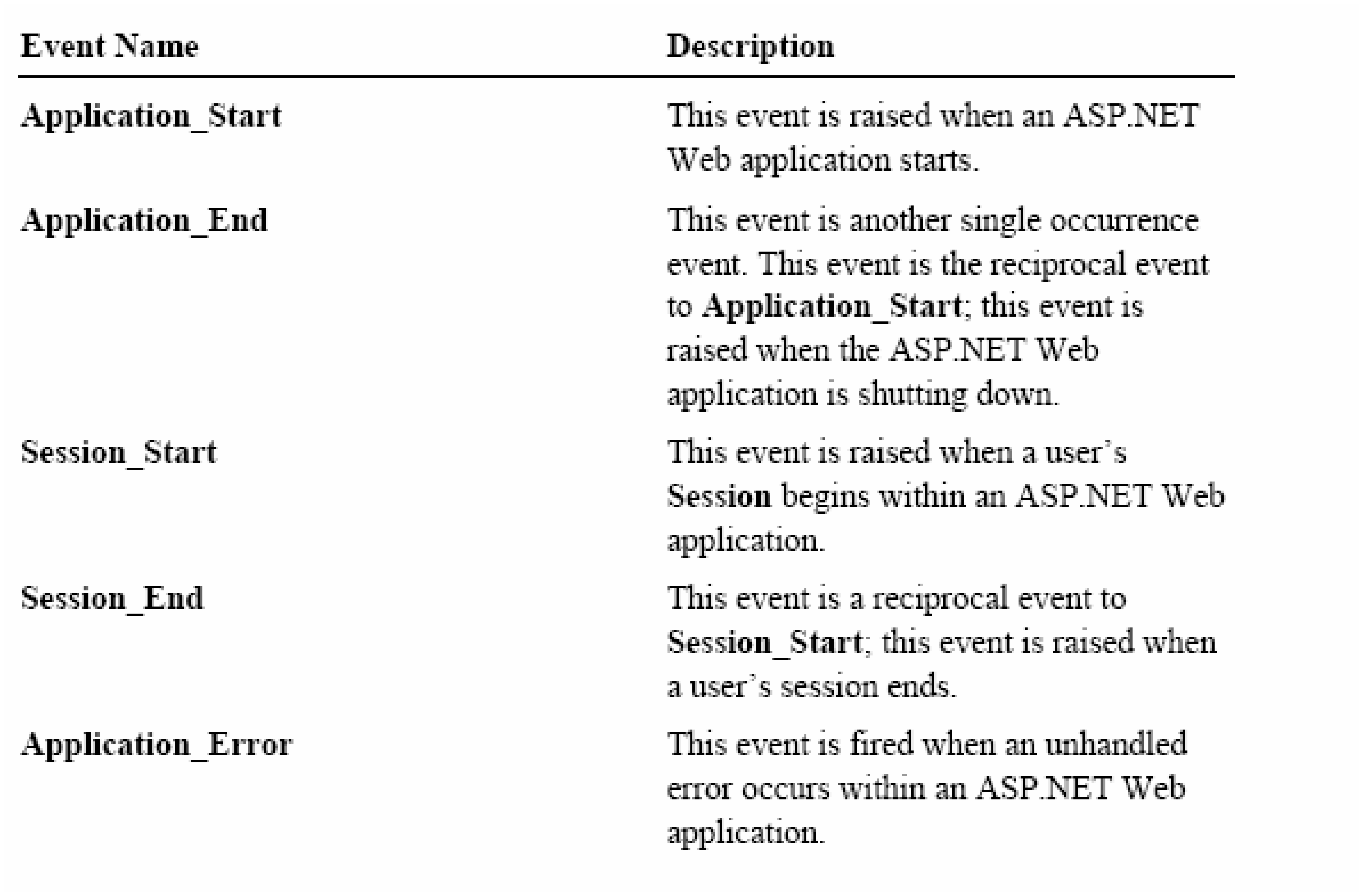
Application Object, Events and Code

- A web application refers to the collection of web pages and objects defined on the server as a virtual directory
- There is one instance of the Application object for each application running on the web server
- The application object
  - Stores information accessible to all clients
  - Stores information about the sessions active within a particular application
- Variables in Application object are defined in a special ASP.NET file – global.asax
  - Placed in the application's root directory
  - Each application can have only one global.asax

The Global.asax File

- Parsed and compiled, at runtime, into a dynamically generated .NET Framework class derived from the `HttpApplication` base class.
- Configured so that any direct URL request for it is automatically rejected; external users cannot download or view the code written within it.
  - A suitable place to place application-sensitive data
- The Global.asax file is optional.
- When the application receives the first user request, the `Application_Start` event is fired.
- If the `global.asax` file is edited and the changes are saved, then all current pending requests are completed, the `Application_End` event is fired, and the application is restarted.
- This sequence effectively reboots the application, flushing all state information.
  - The rebooting of the application is transparent to any users, however, since it occurs only after satisfying any pending requests and before any new requests are accepted.
- When the next request is received, the application starts over again raising another `Application_Start` event.

Application Events



Event Name	Description
Application_Start	This event is raised when an ASP.NET Web application starts.
Application_End	This event is another single occurrence event. This event is the reciprocal event to **Application_Start**; this event is raised when the ASP.NET Web application is shutting down.
Session_Start	This event is raised when a user's **Session** begins within an ASP.NET Web application.
Session_End	This event is a reciprocal event to **Session_Start**; this event is raised when a user's session ends.
Application_Error	This event is fired when an unhandled error occurs within an ASP.NET Web application.

Application Code – global.asax

```
<script language="c#" runat="server" >
 void Application_OnStart(Object obj, EventArgs e){
     Application[“timeKeeper"]= "";
     Application[“visitCounter"]= 0;
 }
</script>
```

- Here is a demo on how to use this file.

Application Configuration Settings

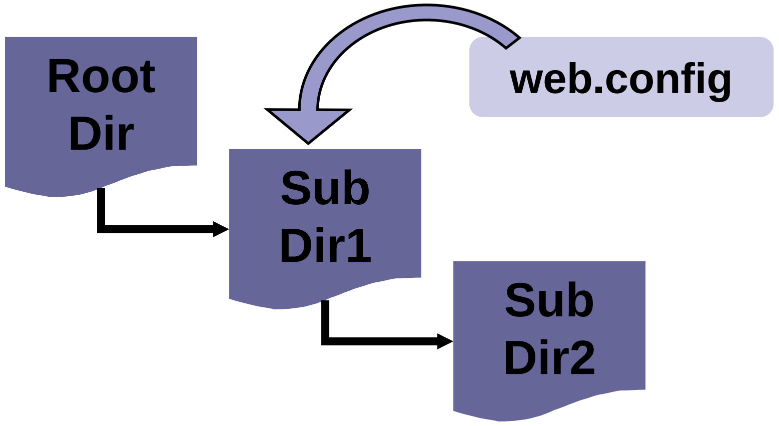
- In classic ASP all Web site related information was stored in the metadata of IIS.
  - Disadvantage: remote Web developers couldn't easily make Web-site configuration changes.
- Such configuration changes need to be done through the IIS admin tool
  - Your Web host will likely charge you a fee to do this for you.
- With ASP.NET, these settings are directly under developer control
  - Placed into an XML-formatted text file (Web.config) that resides in the Web site's root directory.
- Goal of ASP.NET configuration (web.config):
  - Provide extensible configuration for admins & developers to hierarchically apply settings for an application

Hierarchy of .config Files

- Multiple .config files can, and typically do, exist on a single system.
- System-wide configuration settings for the .NET Framework are defined in the Machine.config file.
  - Placed in

%SystemRoot%\Microsoft.NET\Framework\%VersionNumber%\CONFIG\ folder.

- Configuration files can be stored in application folders
  - Configuration system automatically detects changes
- Hierarchical configuration architecture
  - Applies to the actual directory and all subdirectories



Creating a web.config File

- At the root level of web.config is the <configuration> tag.
- Inside this tag you can add a number of other tags
  - The most common and useful one being the system.web tag, where you will specify most of the Web site configuration parameters.
- However, to specify application-wide settings you use the <appSettings> tag.
  - Inside of this tag you can specify zero to many settings by using the <add ... /> tag.
  - For example, if we wanted to add a database connection string parameter we could have a Web.config file like:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <appSettings>
       <add key="connectionString"
          value="Data Source=localhost;Initial Catalog=pubs;Integrated Security=True" />
    </appSettings>
   <system.web>
         ...
    </system.web>
</configuration>
```

- Retrieve as: **string str = ConfigurationSettings.AppSettings[“connectionString”];**

Forms Authentication

- Like IIS, ASP.NET has its own authentication methods

- When IIS receives a request for an ASP.NET resource, like .aspx file
  - It performs its own authentication (if the web app is configured in IIS to do so)
  - And then passes on the request and a security token to the ASP.NET runtime

- ASP.NET supports the following authentication modes
  - None – ASP.NET relies on IIS for authentication
  - Windows – treats the user identity supplied in the security token by IIS as the authenticated user
  - Forms – allows authentication via login forms of the Web Application
  - Passport – uses the Microsoft Passport system running on a separate Passport server for authentication

- Authentication mode is specified within the `authentication` element of the application's Web.config file:

```
<system.web>
   …
  <authentication mode="Windows" />
</system.web>
```

Forms Authentication: Example 1

- Consider the following configuration file:

```
<configuration>
…
<system.web>
    <authentication mode="Forms">
            <forms loginUrl="Login.aspx" >
            </forms>
    </authentication>
    <authorization>
            <deny users="?"/>
    </authorization>
   </system.web>
</configuration>
```

- Suppose you place the above in the folder containing your Web application files, then
  - An attempt by a user to access any file in the Web application now will be redirected to Login.aspx automatically
  - <**deny users="?" />** specifies that all unauthenticated users are denied access to ASP.NET resources in the site
  - Users information can be hard-coded in an event handler, inside a web.config file or, more appropriately, inside a database

Forms Authentication: Example 1

- The authentication logic can be hard-coded as follows:

```
protected void btnLogin_Click(object sender, EventArgs e)
{
   string user= txtUser.Text;
   string password = txtPassword.Text;

   if (IsValidUser(user, password))
      FormsAuthentication.RedirectFromLoginPage(user, true);
   else
      labError.Text = "User not found, try again";
}

private bool IsValidUser(string user, string password)
   {
      if (user == "sahl" && password == "abushabab")
         return true;
      else
         return false;
   }
```

Forms Authentication: Example 2

- Storing user credentials in a web.config file:

```
<configuration>
…
<system.web>
   <authentication mode="Forms">
          <forms loginUrl="Login.aspx" >
          <credentials passwordFormat="Clear">
                     <user name="sahl" password="abushabab"/>
                     <user name="ahmad" password="abuatfal"/>
                     <user name="ali" password="abulkhair"/>
                  </credentials>
          </forms>
   </authentication>
   <authorization>
          <deny users="?"/>
   </authorization>
   </system.web>
</configuration>
```

Forms Authentication: Example 2

- Since the credentials are now stored in web.config, we can use the built-in **`Authenticate`** method of **`FormsAuthentication`**:

```
protected void btnLogin_Click(object sender, EventArgs
   e)
{
   string user= txtUser.Text;
   string password = txtPassword.Text;

   if (FormsAuthentication.Authenticate(user,password))
      FormsAuthentication.RedirectFromLoginPage(user,
   true);
   else
      labError.Text = "User not found, try again";
}
```

Example 3: Customizing Authentication

- Suppose we want to allow everyone access to the main folder of the application and allow access to a MembersOnly folder only to authenticated users
- We place the following in the main folder

```
<configuration>
…
<system.web>
    <authentication mode="Forms">
             <forms loginUrl="Login.aspx" > <!– can add credentials here ! -->
             </forms>
    </authentication>
    <authorization>
             <allow users=“*"/>
    </authorization>
   </system.web>
</configuration>
```

- And place the following in the `MembersOnly` folder (there should not be `authentication` element here!):

```
<configuration >
    <system.web>
    <authorization>
             <deny users="?"/>
    </authorization>
   </system.web>
</configuration>
```

Example 4: Authentication using WAT

- The most versatile solution is to store user credentials in a database
- This can be done by creating authentication information using the WAT (Website Administration Tool) in Visual Studio 2005
    - Start the WAT:
        - Web Site > ASP.NET Configuration
    - Click the Security Table
    - Click the Create User link
    - Fill in the form and click the Create User button
    - Add two more users
- From the above steps, the WAT would have created an SQL server database with the information you entered added to a number of tables
    - Inspect Example 4 for details
    - Or better still see Chapter 13 of Randy Connolly's "Core Internet Application Development with ASP.NET 2.0", 2007

Stored Procedures

- A precompiled collection of SQL statements stored under a name and processed as a unit.
- They're stored in and deployed with the database
- They are usually written in a proprietary database language like PL/SQL for Oracle database or PL/PgSQL for PostgreSQL.
- Stored procedures are extremely similar to the constructs seen in other programming languages.
  - They accept data in the form of input parameters that are specified at execution time.
  - These input parameters (if implemented) are utilized in the execution of a series of statements that produce some result.
  - This result is returned to the calling environment through the use of a recordset, output parameters and a return code.
- Types
  - User-defined Stored Procedures
  - System Stored Procedures

Benefits of Stored Procedures

1. Precompiled execution.
   - SQL Server compiles each stored procedure once and then reutilizes the execution plan.
   - This results in tremendous performance boosts when stored procedures are called repeatedly.
2. Reduced client/server traffic.
   - Stored procedures can reduce long SQL queries to a single line thereby reducing network traffic.
3. Efficient reuse of code and programming abstraction.
   - Stored procedures can be used by multiple users and client programs.
   - Judicious use of stored procedures can reduce development time.
4. Enhanced security controls.
   - You can grant users permission to execute a stored procedure independently of underlying table permissions.

Stored Procedures: Example

- Consider the following `studentGrades` table:

ID	Name	Standing	Grades(%)
40232	Ahmad	P	50
40165	Khalid	G	50
40147	Qais	P	50
40244	Ibrahim	P	99
40284	Ali	G	84
40434	Amr	G	32

Example (cont'd)

- In Query:

```
SELECT Name, Grades
FROM studentGrades
WHERE Standing = ‘G’
```

- In Stored Procedure :

```
CREATE PROCEDURE sp_GetGrades
     @standing varchar(1)
     AS
     SELECT Name, Grades
     FROM studentGrades
     WHERE Standing = @standing
```

Example (cont'd)

- If we want to get the grades for good standing students:

```
EXECUTE sp_GetGrades ‘G'
```

- If we want to get the grades for probation students:

```
EXECUTE sp_GetGrades ‘P‘
```

- Show a demo.