

# ASP.NET Fundamentals

Them Pham

Dec 2011



# Course Objectives

- Get an overview of ASP.NET Architecture
- Access to key components of ASP.NET that are common to almost all Web applications
- Be able to build dynamic websites with ASP.NET using standard libraries shipped with .NET Framework

# Audience and Prerequisite

- Designed for professional programmers who need to create rich, interactive websites
- Fundamentals of .NET Framework, C#/VB.NET programming language.
- Background in ADO.NET-related data access would be helpful
- Attendance of “Web 101 – Web Development Fundamentals” course would be helpful

# Agenda

- ASP.NET Architecture
  - ASP.NET page request processing in IIS
  - Evolution of ASP.NET
  - Code behind model
  - Page life cycle and event handling
  - State management overview
- Standard controls & Validations
- Designing Websites with Master Pages & Themes
- Building Site Navigation

# Assessment Disciplines

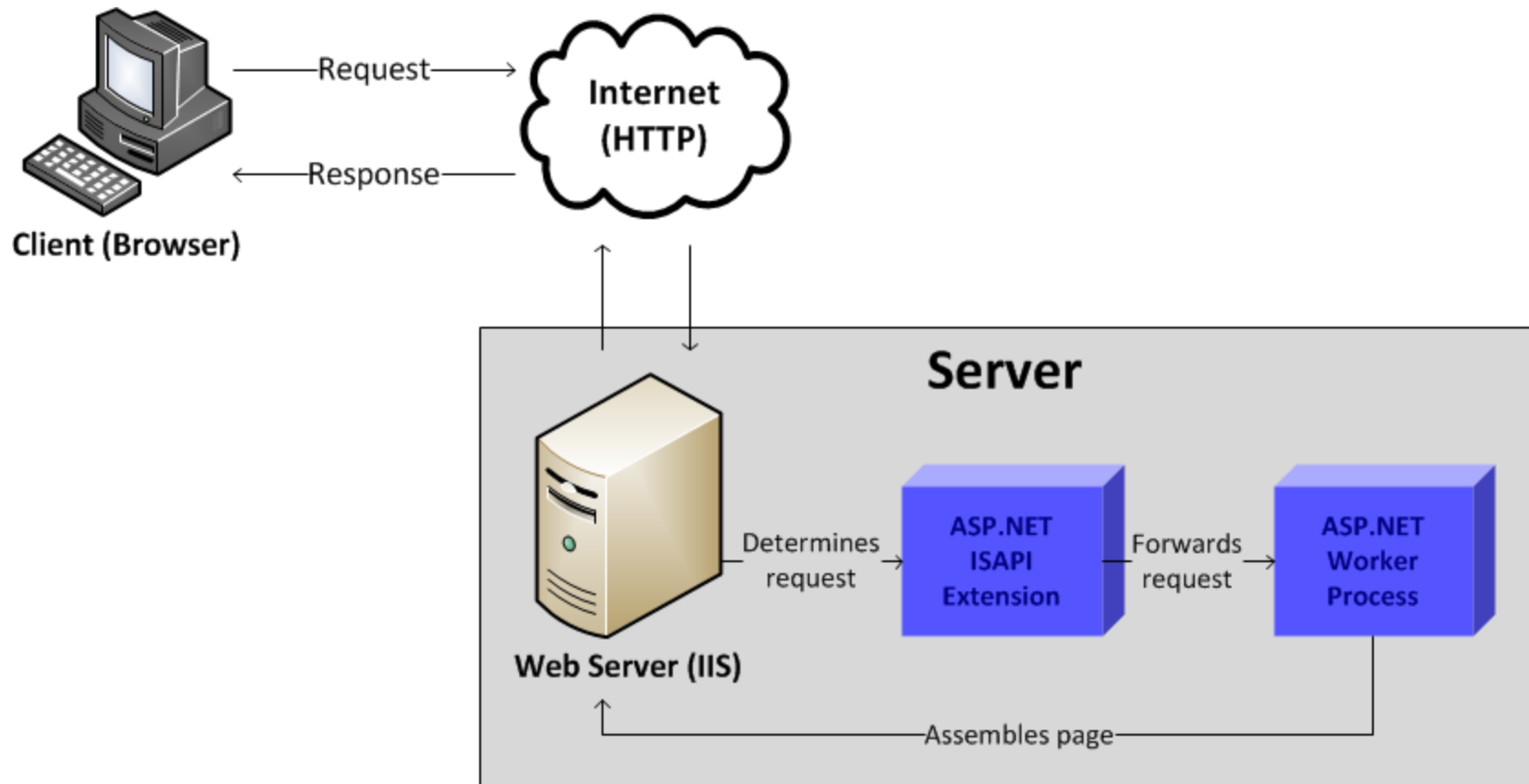
- Class Participation: 40%
- Final Exam: 60%
  - Duration: 3h
- Passing Score: 70%

# ASP.NET Architecture

# What is ASP.NET?

- A Web platform that provides all the services that are required to build enterprise-class server-based Web applications.
- Built on the .NET Framework → all .NET Framework features are available to ASP.NET applications.
- Don't need to paste together a jumble of HTML and script code in order to program the Web.
- Allow to use a full featured programming language such as C# or VB.NET to build web applications easily.
- Run inside IIS (Internet Information Services) server.

# How a web server processes ASP.NET pages?





# Evolution of ASP.NET

- Dynamic content in classic ASP
  - Interspersed server-side script notation
  - Maintainability, testability... problems

```
<h2>List of items:</h2>
<ul>
  <% for (int i = 0; i < _itemCount; i++) { %>
    <li><%=GetItem(i)%></li>
  <% } %>
</ul>

<% Response.Write("<h2>Number of items: " +
  _itemCount.ToString() + "</h2>"); %>
```

# Evolution of ASP.NET (cont.)

- Introducing server controls
  - Elements are marked with `runat="server"`
  - Clean alternative to the interspersed server-side script
  - Define member variables in class
  - Programmatically interact with object model of controls
  - State is retained implicitly

```
protected override void OnLoad(EventArgs e)
{
    bllItems.Items.Clear();
    for (int i = 0; i < _itemCount; i++)
    {
        bllItems.Items.Add(new ListItem(GetItem(i)));
    }
    base.OnLoad(e);
}
```

```
<asp:BulletedList runat="server" ID="bllItems">
    <asp:ListItem>Default item #1</asp:ListItem>
    <asp:ListItem>Default item #2</asp:ListItem>
</asp:BulletedList>
```

## Evolution of ASP.NET (cont.)

- Introducing data binding to list-typed server controls with server code
- Introducing declarative model

```
string[] _items = { "Item #1", "Item #2", "Item #3",  
                    "Item #4", "Item #5", "Item #6",  
                    "Item #7", "Item #8", "Item #9", "Item #10" };  
  
protected override void OnLoad(EventArgs e)  
{  
    bllItems.Items.Clear();  
  
    bllItems.DataSource = _items;  
    bllItems.DataBind();  
  
    base.OnLoad(e);  
}
```

# Code-behind model

- ASP.NET supports two methods to author pages: in-line code and code-behind
- Code-behind allows a clean separation of HTML from presentation logic
- Wire up the code-behind class to layout file by using **CodeFile** attribute (point to code-behind file) and **Inherits** attribute (class name with namespace)


```
<%@ Page Language="C#" CodeFile="~/Default.aspx.cs" Inherits="ASPNETFundamentals.SamplePage" %>
```

```
<head id="Head1" runat="server">
  <title>Simple Page</title>
</head>
<script runat="server" language="C#">
void btnSample_OnClick(Object sender, EventArgs e)
{
    lblSample.Text = txtSample.Text;
}
</script>
<body>
  <form id="MyForm" runat="server">
    <asp:TextBox id="txtSample" Text="Hello World!" runat="server">
    </asp:TextBox>
    <asp:Button id="btnSample" Text="Send" OnClick="btnSample_OnClick" runat="server">
    </asp:Button>
    <asp:Label id="lblSample" runat="server">
    </asp:Label>
  </form>
</body>
```

```
namespace ASPNETFundamentals
{
    public partial class SamplePage : Page
    {
        protected void btnSample_OnClick(Object sender, EventArgs e)
        {
            lblSample.Text = txtSample.Text;
        }
    }
}
```

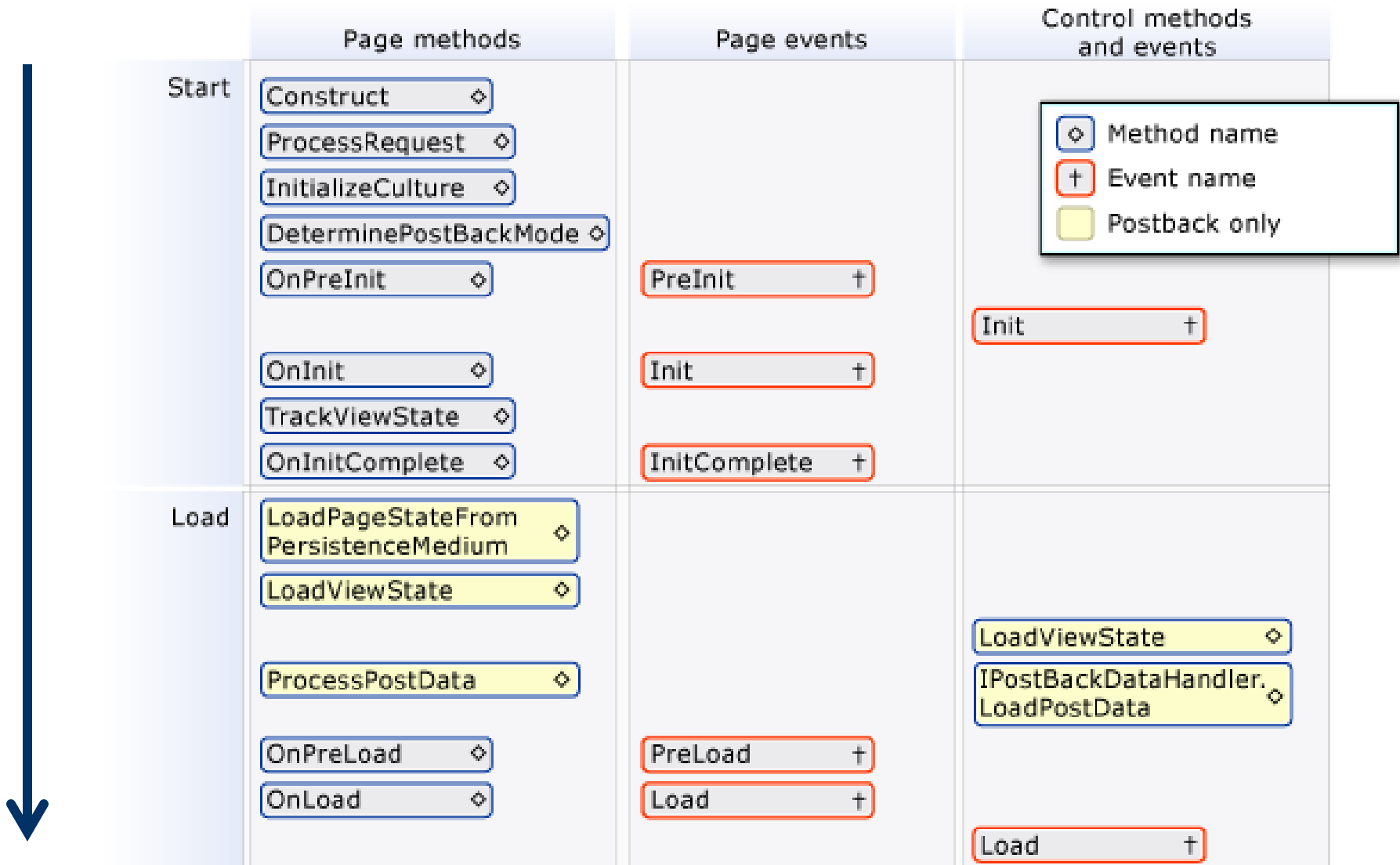
# Page life-cycle stages

- Allow to write code at the appropriate stage




Stage	Description
Page request	Begin the life of a page or load a cached version of the page.
Start	Page properties (e.g. <a href="#">Request</a> and <a href="#">Response</a> ) are set. If a new request → set <a href="#">IsPostBack</a> property.
Initialization	Controls on the page are available. Themes initialized, master pages applied, control skins applied.
Load	If IsPostBack, control properties are loaded with information recovered from view state and control state.
PostBack event handling	If IsPostBack, control event handlers are called → <a href="#">Validate</a> method of all validator controls is called → <a href="#">IsValid</a> flag is set.
Rendering	Control/View state saved. <a href="#">Render</a> method of controls is called.
Unload	Raised after the page has been fully rendered, sent to the client, and is ready to be discarded.

# Page life-cycle events



# Page life-cycle events (cont.)



	Page methods	Page events	Control methods and events
Event handling	RaisePostBackEvent ◊		Control-changed events
Validation	Validate ◊		
PreRendering	OnLoadComplete ◊	LoadComplete †	
	OnPreRender ◊	PreRender †	
			PreRender †
			Data binding events
	OnPreRenderComplete ◊	PreRenderComplete †	
	SaveViewState ◊		SaveViewState ◊
Rendering	SavePageStateToPersistenceMedium ◊		
	OnSaveStateComplete ◊	SaveStateComplete †	
	RenderControl ◊		
	Render ◊		
Unload	RenderChildren ◊		Render ◊
			Unload †
	OnUnload ◊		
	Dispose ◊		

# State management overview

- HTTP protocol is stateless
- Methods to maintain state:
  - View State
  - Transferring information between pages
  - Cookie
  - Session State
  - Application State
  - Profile Properties



# State management – View State

- View State uses hidden field to store information
- Used for multiple postbacks in a single web page
- Not limited to web controls
- Can be used to retain class member variables
- Can be used to store serializable objects
- EnableViewState property is, by default, always set to true to web controls

```
public partial class SamplePage : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        Employee emp = new Employee("Them", "Pham");
        ViewState["EmployeeInfo"] = emp;
        //get back
        emp = (Employee)ViewState["EmployeeInfo"];
    }
}

[Serializable]
public class Employee
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public Employee(string firstName, string lastName)
    {
        FirstName = firstName;
        LastName = lastName;
    }
}
```

# State management – Maintain information between pages

- Cross Page Posting:
  - Postback mechanism that sends user to another page by setting PostBackUrl property
  - Posted page accesses to previous page via its PreviousPage object

## Page1.aspx

```
<asp:TextBox ID="TextBox1" runat="server" />
<asp:Button ID="Button1" runat="server" Text="Button"
            PostBackUrl="~/Page2.aspx" />
```

## Page2.aspx

```
protected void Page_Load(object sender, EventArgs e)
{
    if (PreviousPage != null)
    {
        string sName = string.Empty;
        sName = ((TextBox) (PreviousPage.FindControl("TextBox1"))).Text;
        Response.Write("Hello: " + sName);
    }
}
```

# State management – Maintain information between pages (cont.)

- Transferring with Query String:
  - Query string is common to web regardless of server technology
  - A key/value pair which is appended to the URL after question mark
  - Release the pressure of the server
  - Limitations?
- Ensure query string value is encoded before appending them to the URL

```
//set query string and redirect
string url = "Page2.aspx?";
url += "fName=Them&";
url += "lName=" + Server.UrlEncode("Pham Van");
Response.Redirect(url);
//retrieve query string
string lName = Server.UrlDecode(Request.QueryString["lName"]);
```

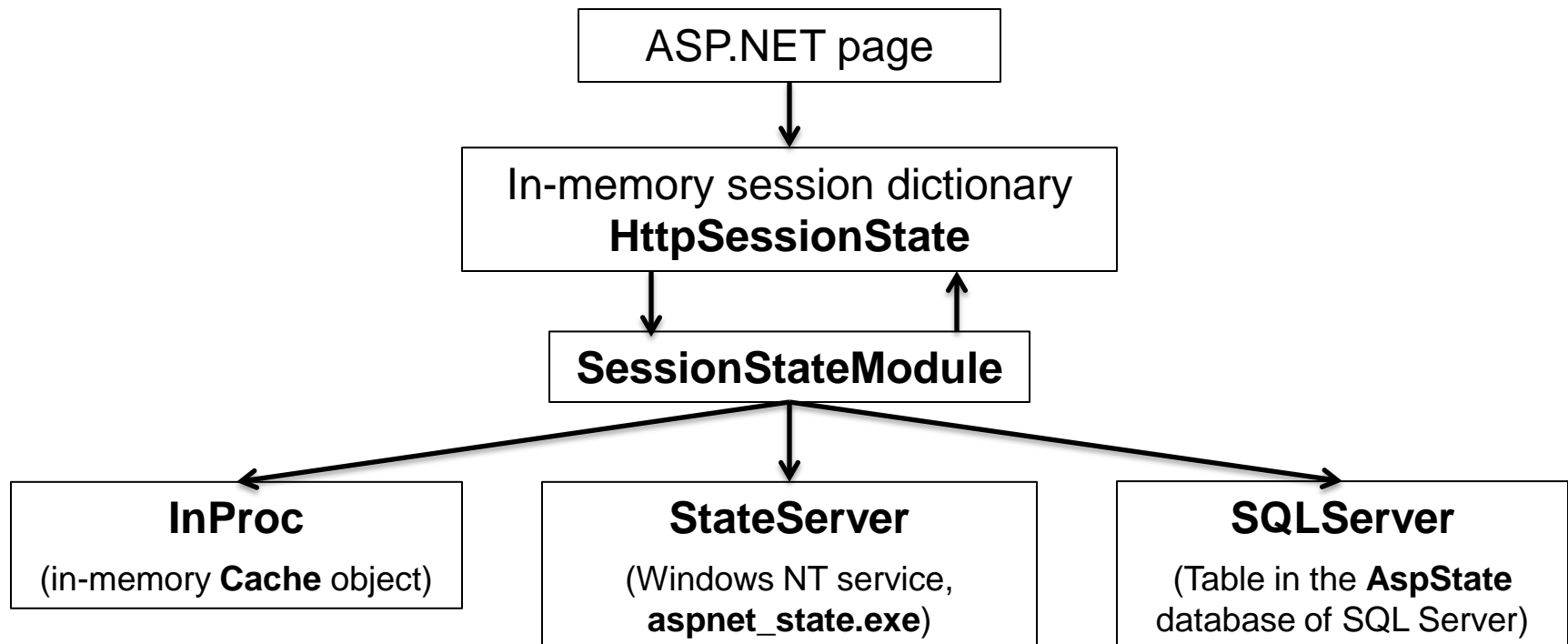
# State management – Cookie

- Cookie is stored at client machine
- Transient (or temporary) cookie
  - A cookie without expiration date
  - Stored in the web browser's memory
  - Cleared when user closes the browser
- Persistent (or permanent) cookie
  - A cookie with a specific expiration date set
  - Stored in the client's hard disk. For example:
    - For IE: \Documents and Settings\[user]\Cookies
  - Expired when the expiration date is exceeded

```
//create cookie and set a value
HttpCookie cookie = new HttpCookie("lang", "en-US");
cookie.Expires = DateTime.Now.AddDays(7d); //expired after 7 days
Response.Cookies.Add(cookie);
//retrieve cookie
string lang = Request.Cookies["lang"] != null ? Request.Cookies["lang"].Value : "n/a";
```

# State management – Session State

- Used to store complex information/objects per session in the server memory
- No size limitations
- Accessible across pages
- By default, session times out after 20 minutes of inactivity



# State management – Application State

- Store global objects that are accessible over the application
- Never time out
- Set exclusive access to the block of code

```
//set exclusive access
Application.Lock();
int counter = 0;
if (Application["counter"] != null)
{
    counter = (int)Application["counter"];
}
counter++;

Application["counter"] = counter;
// release exclusive access.
Application.UnLock();
```

# State management – Profiles

- Store information permanently using a back-end data source
- A profile is specific to an authenticated user
- Profile properties are strongly-type ones

```
<profile>
  <properties>
    <group name="MandatoryInfo">
      <add name="FirstName" type="System.String"/>
      <add name="LastName" type="System.String"/>
    </group>
    <group name="OptionalInfo">
      <add name="DateOfBirth" type="System.DateTime"/>
      <add name="Address" type="System.String"/>
    </group>
  </properties>
</profile>
```

```
Profile.MandatoryInfo.FirstName = "Them";
Profile.MandatoryInfo.LastName = "Pham";
Profile.OptionalInfo.Address = "My address";
```

# Standard controls & Validations



# HTLM Server Controls

- Provide an object interface for standard HTML elements by marking `runat="server"`
- Retain state automatically
- Convenient for manipulating elements in server-side code without changing markup significantly
- Useful when migrating ordinary HTML pages or ASP pages to ASP.NET

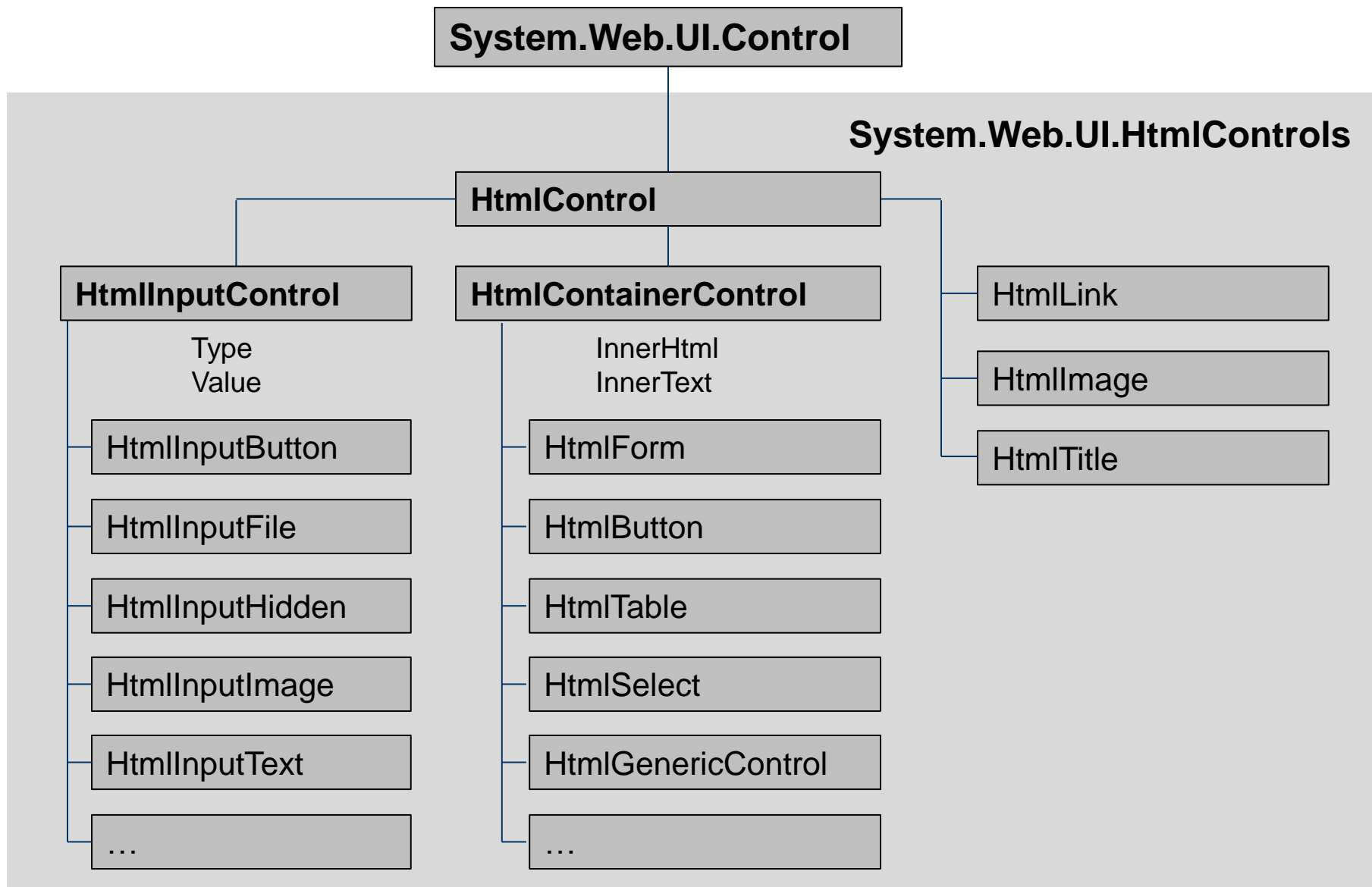
```
<div id="mainContainer" runat="server">  
    This content is from HTML.  
</div>
```

```
mainContainer.Attributes.Add("style", "color:red;");  
mainContainer.InnerHtml = "This content is from server-side";
```

# HTLM Control Classes

Class Name	HTML Element	Class Name	HTML Element
HtmlForm	<form>	HtmlTextArea	<textarea>
HtmlAnchor	<a>	HtmlInputImage	<input type="image">
HtmlImage	<img>	HtmlInputFile	<input type="file">
HtmlTable HtmlTableRow HtmlTableCell	<table>, <tr>, <th>, <td>	HtmlInputHidden	<input type="hidden">
HtmlInputButton HtmlInputSubmit HtmlInputReset	<input type="button"> <input type="submit"> <input type="reset">	HtmlInputText HtmlInputPassword	<input type="text"> <input type="password">
HtmlButton	<button>	HtmlSelect	<select>
HtmlInputCheckBox	<input type="checkbox">	HtmlHead HtmlTitle	<head> <title>
HtmlInputRadioButton	<input type="radio">	HtmlGenericControl	Other HTML element

# HTLM Controls Hierarchy



# Web Controls

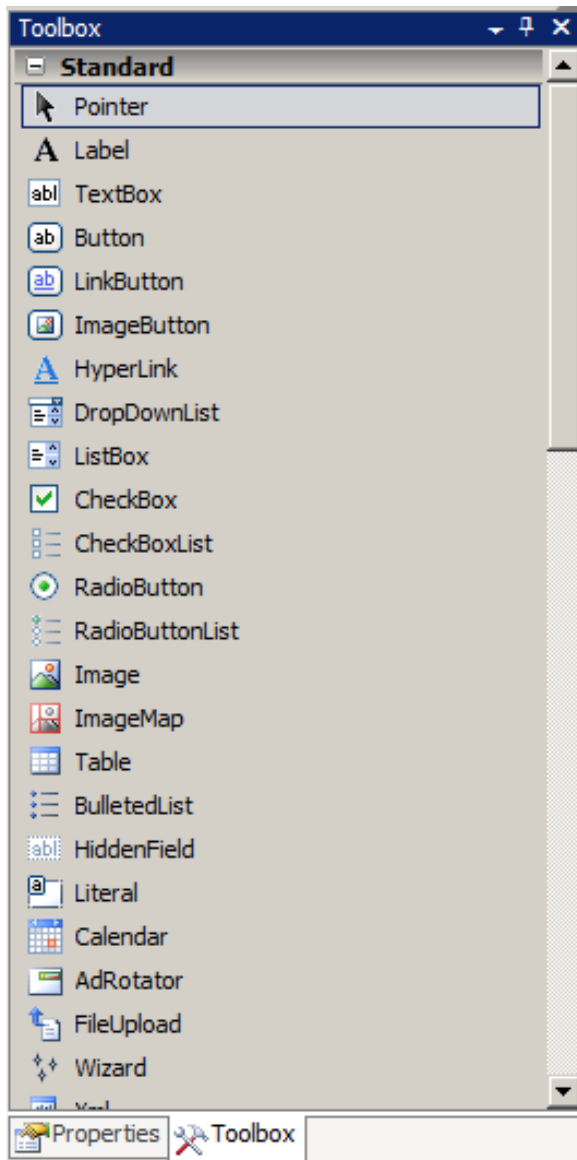
- Alternative to server-side HtmlControls
  - Do not have restriction that HtmlControls have
  - More consistent property naming
  - Consistent object model
- Provide a rich user interface, not just a single element
- Adaptive rendering
- Begin with **asp:** prefix in the tag. For example,  

```
<asp:TextBox ID="TextBox1" runat="server" />
```
- Question: when to choose html controls and when for web controls?

# Basic Web Controls Classes

Class Name	HTML Element	Class Name	HTML Element
Label	<span>	Image	<img>
Button	<input type="submit"> <input type="button">	ListBox	<select size="N">
CheckBox	<input type="checkbox">	DropDownList	<select>
TextBox	<input type="text"> <input type="password"> <textarea>	CheckBoxList	Multiple <input type="checkbox">
RadioButton	<input type="radio">	RadioButtonList	Multiple <input type="radio">
Hyperlink	<a>	BulletedList	An <ol> or <ul> list
LinkButton	<a> with a contained <img> tag	Panel	<div>
ImageButton	<input type="image">	Table TableRow TableCell	<table> <tr> <td> or <th>

# Web Controls Hierarchy



**System.Web.UI.Control**

**System.Web.UI.WebControls**

**WebControl**

Label

CheckBox

HyperLink

Image

Button

LinkButton

Panel

...

**BaseValidator**

**BaseCompareValidator**

CompareValidator

RangeValidator

CustomValidator

RegularExpressionValidator

RequiredFieldValidator

# Validation Controls

- Why validation?

Validation Control	Description
CompareValidator	Compares the value of one input control to the value of another input control or to a fixed value
RequiredFieldValidator	Makes an input control a required field
RangeValidator	Validates whether a value is within a range of values (e.g. specific numeric, alphabetic, or date range)
RegularExpressionValidator	Ensures that the value of an input control matches a specified pattern
CustomValidator	Validation is performed by a user-defined function
ValidationSummary	Displays a report of all validation errors occurred in a Web page

# Custom Validation Controls

- All built-in validators inherit BaseValidator class
  - Custom validation controls must also inherit the BaseValidator
- BaseValidator is an abstract class
  - **EvaluateIsValid**: true if form fields are validated as valid
  - GetControlValidationValue: get the value of the control being validated

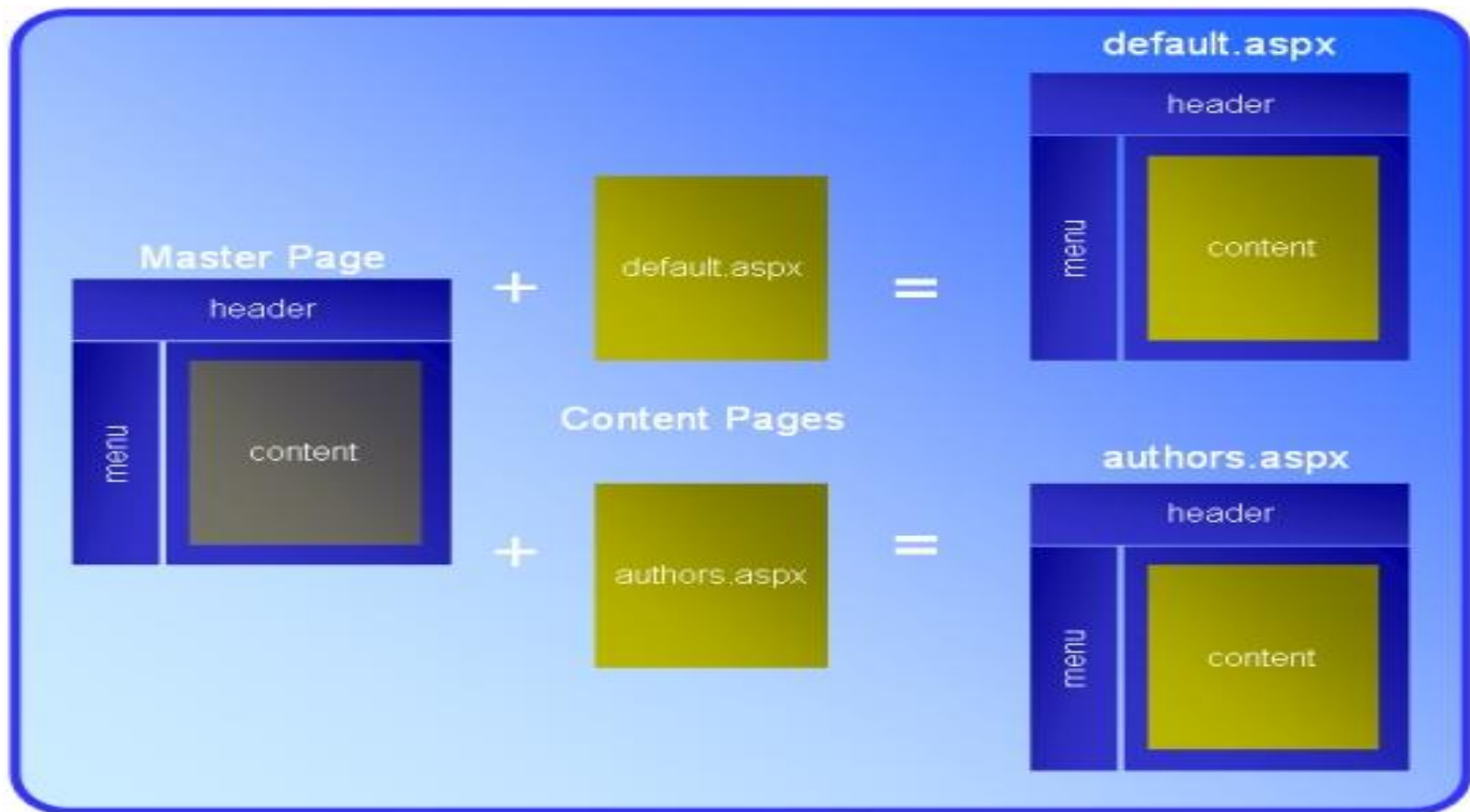


# Exercise 01

# Designing Websites with Master Pages & Themes

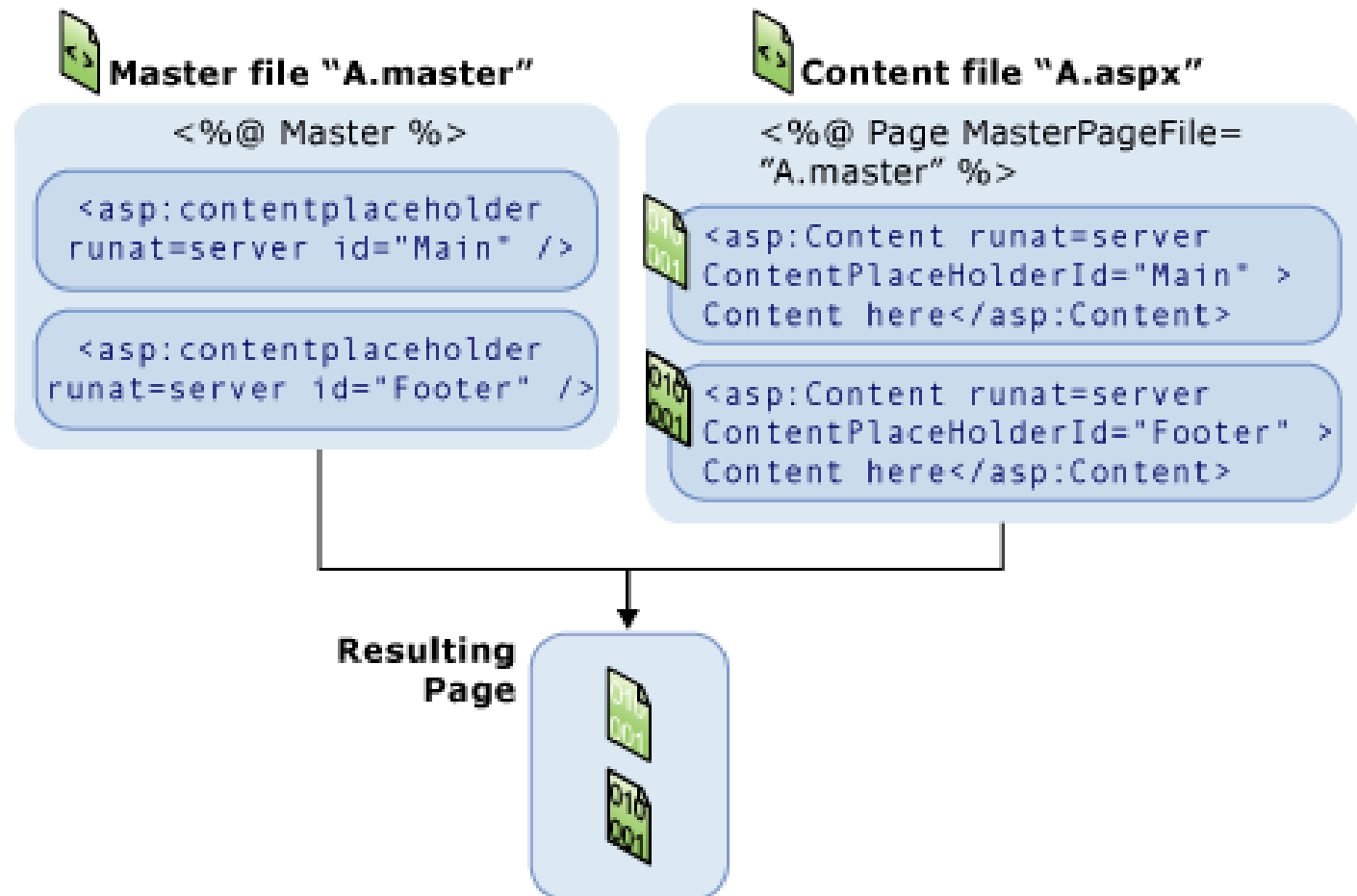
# Master Pages

- A “visual-inheritance” page-template based system
- A Master Page is just a type of file with .master extension



# How Master Pages Work?

- Master pages define common content and placeholders
- Content pages reference masters and fill placeholders with content

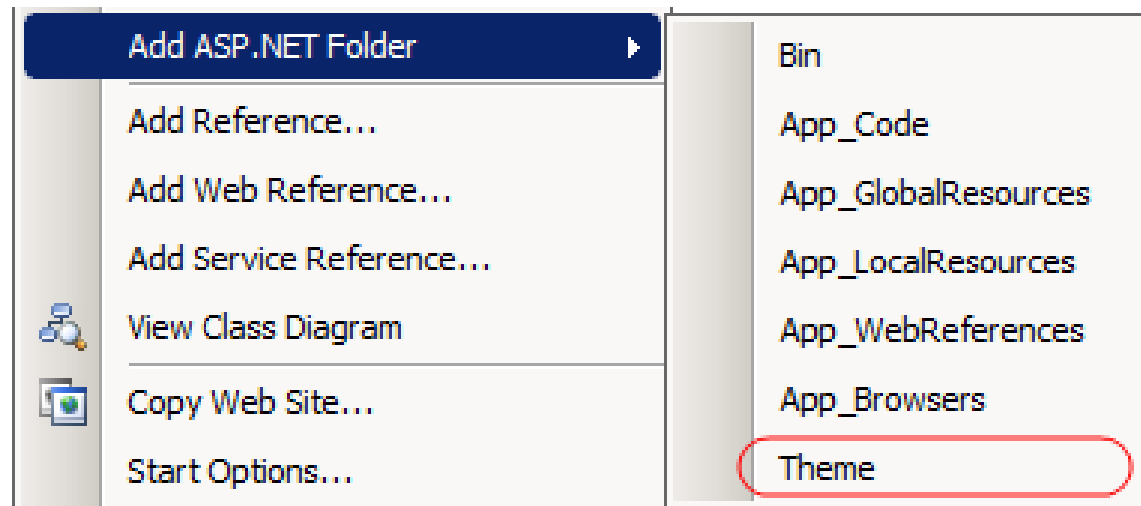


# Advanced Master Pages

- Style-based layout:
  - Html tables: break a portion of a page into columns and rows
  - CSS positioning: used with <div> tags (more standard and modern)
- Coding in Master Page:
  - Master pages can include code portion like a normal ASP.NET web page
- Interacting with Master Pages programmatically:
  - Content pages access to master page via Page.Master property
  - Controls on master page are accessible to content pages
  - Public properties are accessible

# Themes and Skins

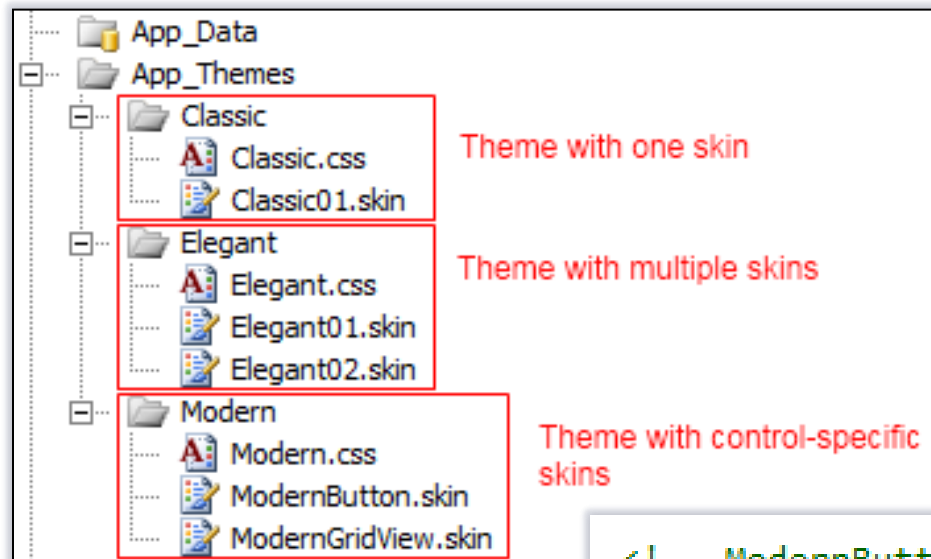
- What are themes?
  - Define a set of style details to be applied to controls in multiple pages
  - Themes are typically application specific
  - A theme is created as a folder beneath the special ASP.NET **App\_Themes** folder
  - Used together with skins



# Themes and Skins

- Skins

- Enable modifying properties of ASP.NET controls



```
<!-- ModernButton.skin -->
```

```
<asp:ImageButton runat="server" SkinID="OKButton"  
  ImageUrl="ButtonImages/buttonOK.jpg" />
```

```
<asp:ImageButton runat="server" SkinID="CancelButton"  
  ImageUrl="ButtonImages/buttonCancel.jpg" />
```

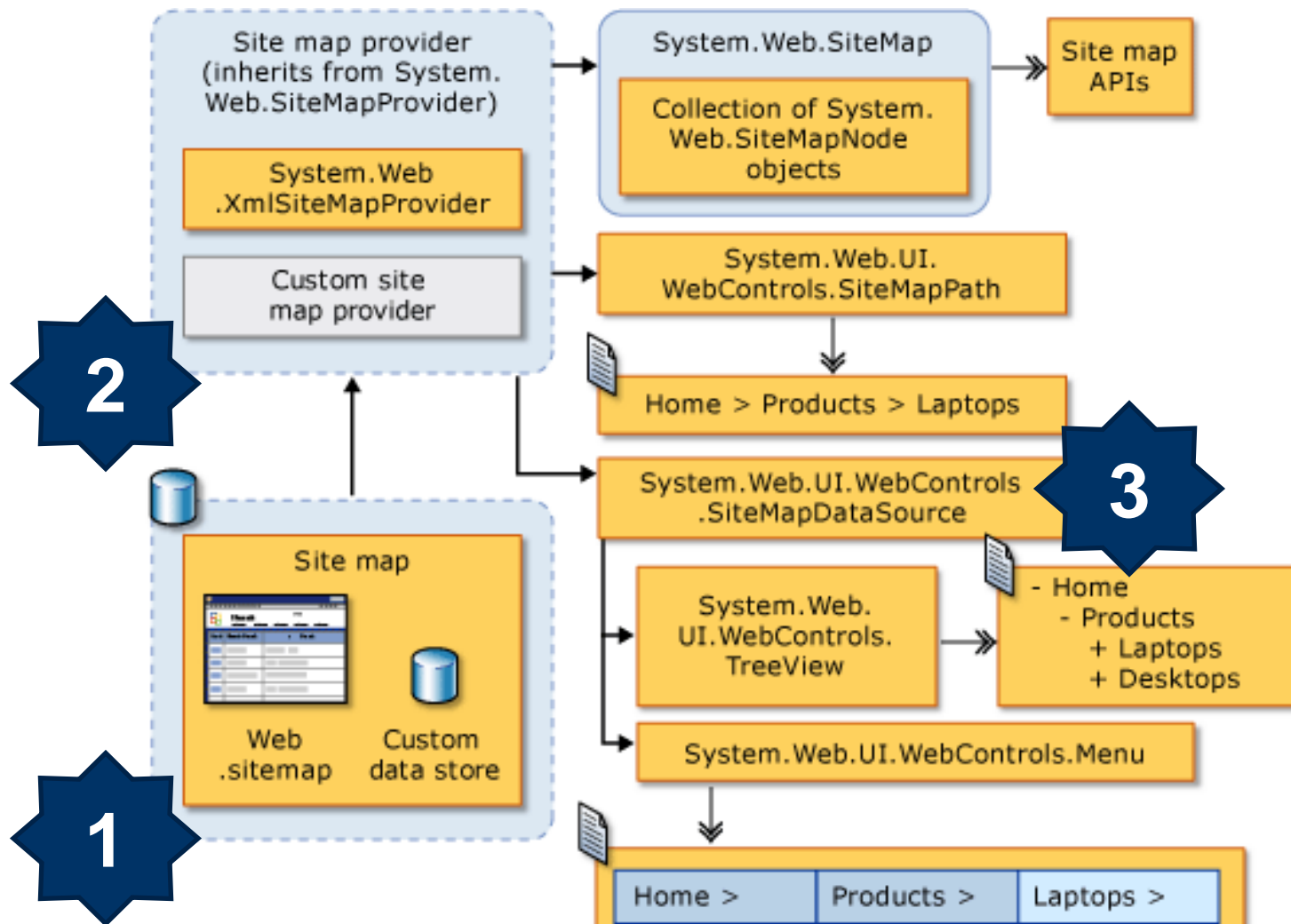
# Exercise 02



# Building Site Navigation

# Site Maps

- A consistent way to navigate from one page to the next



# SiteMap and SiteMapNode classes

- **SiteMap** represents an application's site map stored in different data sources: XML file, database or custom data source
- Pages/folders in a site map are represented by instances of **SiteMapNode**

```
void Application_Start(object sender, EventArgs e)
{
    SiteMap.SiteMapResolve += new
        SiteMapResolveEventHandler(SiteMapResolver)
}

SiteMapNode SiteMapResolver(object sender,
    SiteMapResolveEventArgs e)
{
    if (SiteMap.CurrentNode == null)
    {
        string url = e.Context.Request.Path;
        string title = Path.GetFileNameWithoutExtension(url);
        SiteMapNode newNode = new
            SiteMapNode(e.Provider, url, url, title);
        newNode.ParentNode = SiteMap.RootNode;
        return newNode;
    }
    return SiteMap.CurrentNode;
}
```

```
if (!Page.IsPostBack)
{
    SiteMapNode curNode = SiteMap.CurrentNode;
    string nodeTitle = curNode.Title;
    string nodeDesc = curNode.Title;
    string nodeUrl = curNode.Url;
}
```

# Navigation Controls

- SiteMapPath
  - Builds “breadcrumbs” based on site map content: indicate the current position
- Menu
  - Supports hierarchical data
  - Supports horizontal and vertical layouts
  - Support templates
- TreeView
  - Supports hierarchical data
  - Only vertical layout is supported
  - No template supported

# Remapping URLs

- Map a requested URL to different URL
- URL mappings are configured in web.config
  - Request.RawUrl: original URL (before being remapped)
  - Request.Path: current URL (after being remapped)

```
<system.web>
  <urlMappings enabled="true">
    <add url="/beverages.aspx"
         mappedUrl="/products.aspx?category=beverages" />
    <add url="/food.aspx"
         mappedUrl="/products.aspx?category=food" />
  </urlMappings>
```

# URL Routing Engine

- New in ASP.NET 4
- Cleaner alternatives for complicated-looking URLs
- Better SEO (Search Engine Optimization)
  - <http://mysite.com/products/beverages/cokes/pepsi>
- Flexibility of using URL parameters
  - No concerns about figuring out what the query string parameter names are
  - <http://mysite.com/blog/2011/12/08>
- Implemented using code, not configuring in web.config
  - Typically implemented in Application\_Start event

# Exercise 03



# Q&A



# Revision History

Date	Version	Description	Updated by	Reviewed and Approved By
12/08/2011	1.0	Ready for Review	Them Pham	