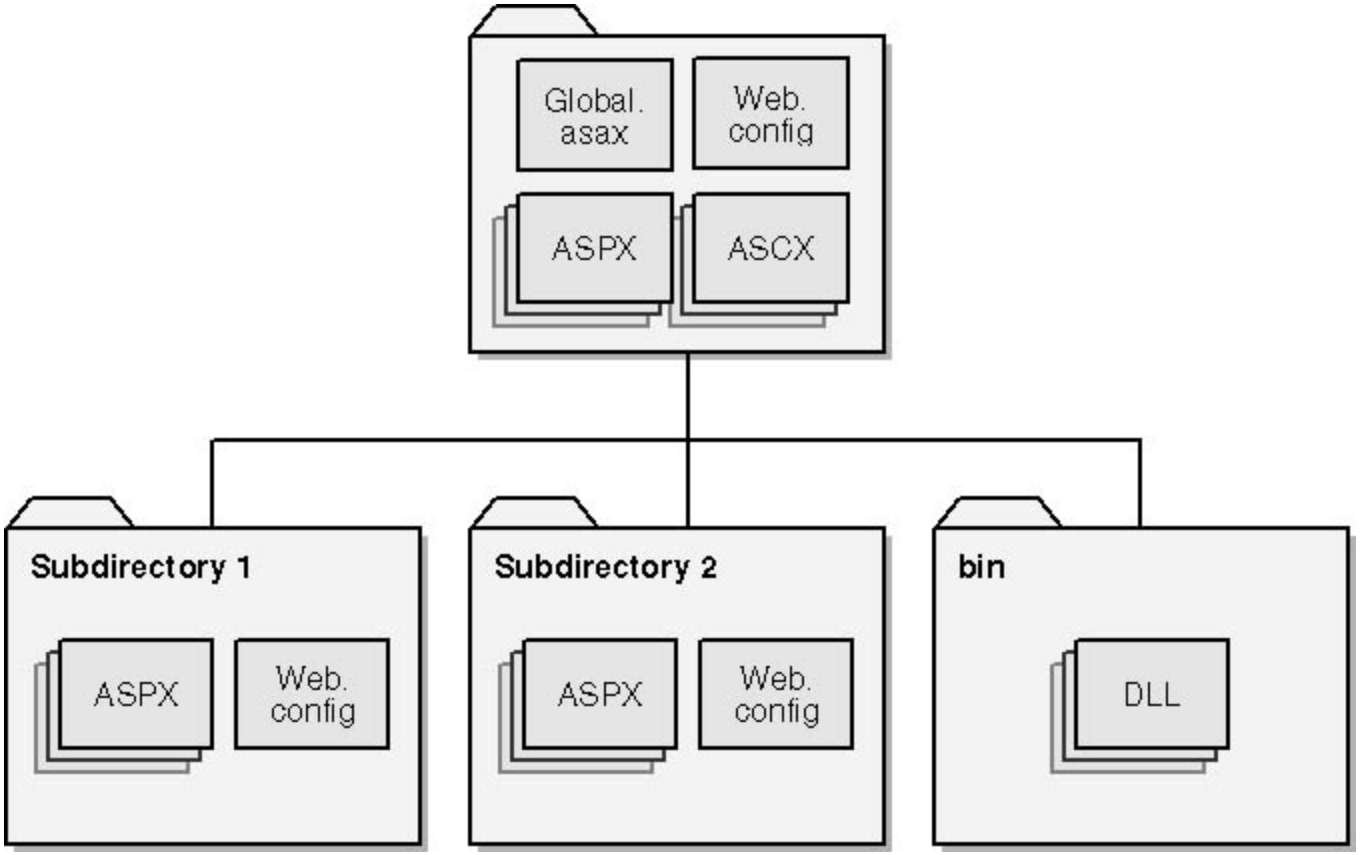
Web Applications

Structure of an ASP.NET Application

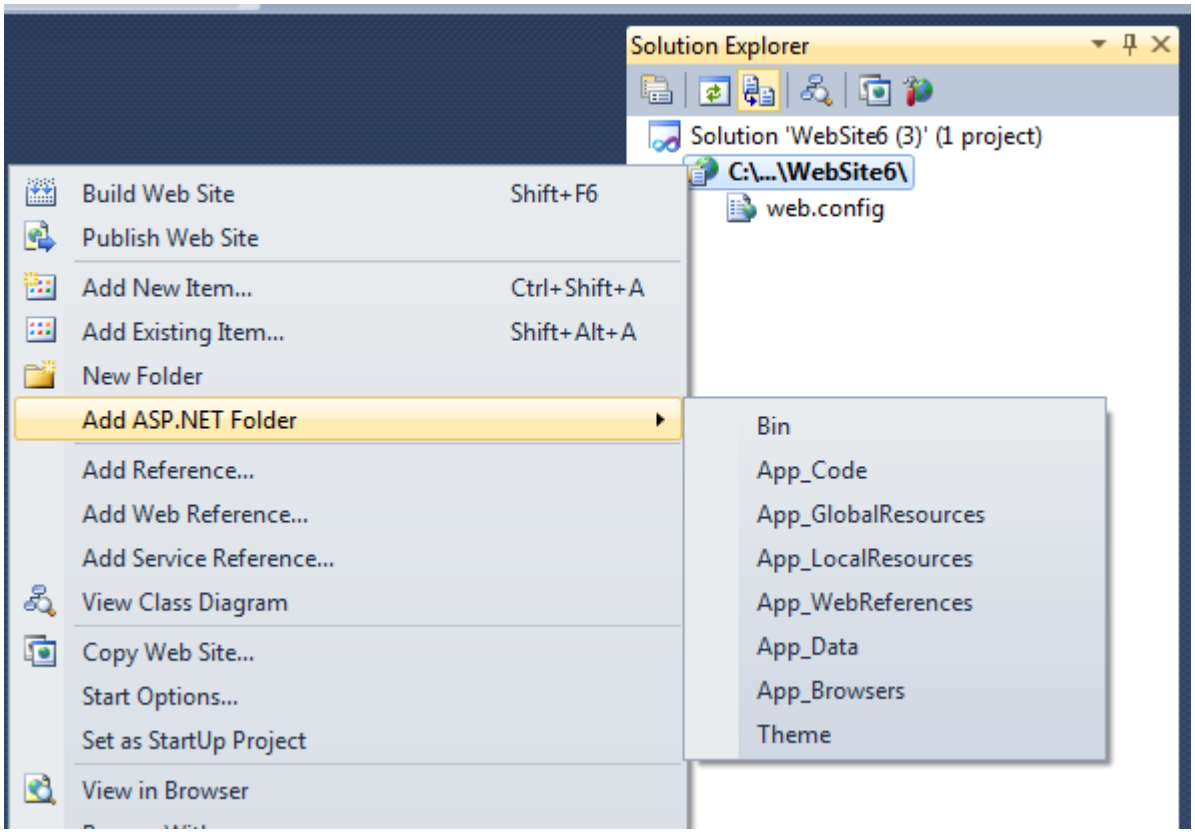
- ASPX files containing Web forms
- ASCX files containing user controls
- Web.config files containing configuration settings
- A Global.asax file containing global application elements
- Possible DLLs containing custom types employed by the application
- HTML files
- Other file misc. file types

Directory Organization



ASP.NET Folders

We can add special folders specifically for ASP.NET. You have already seen the use of App_Data.



The Web.config File

```
<configuration>
  <appSettings>
    <!-- appSettings values go here -->
  </appSettings>
  <system.web>
    <!-- ASP.NET configuration settings go here -->
  </system.web>
</configuration>
```

<appSettings>

- Essentially take the place of the registry or .ini files.
- Use Key/Value pairs.
- Both are strings.
- Consider a database application where the location of the database may change. We can place the connection string in the web .config file as you have already seen.
- Idea – store an application version number to be displayed on more than one page.

```
<configuration>
  <appSettings>
    <add key="MyConnectionString"
      value="server=hawkeye;database=pubs;uid=sa;pwd=" />
  </appSettings>
</configuration>
```

Page_Load can be rewritten to extract the connection string from Web.config:

```
string conn = ConfigurationSettings.AppSettings["MyConnectionString"];
SqlDataAdapter adapter = new SqlDataAdapter
    ("select * from titles where price != 0", conn);
DataSet ds = new DataSet ();
adapter.Fill (ds);
```

<system.web>

- Contains system settings to control ASP.NET.
- Some examples are:
  1. authentication
  2. authorization
  3. compilation
  4. security policy
  5. trace settings
- You have seen how web.config is used to allow debugging and control security.

Example

```
<configuration>
  <system.web>
    <trace enabled="true" />
    <sessionState
      mode="SQLServer"
      sqlConnectionString="server=localhost;uid=sa;pwd=" />
    <compilation debug="true" defaultLanguage="c#" />
    <pages enableViewStateMac="true" />
  </system.web>
</configuration>
```

Specifies the location of session state information

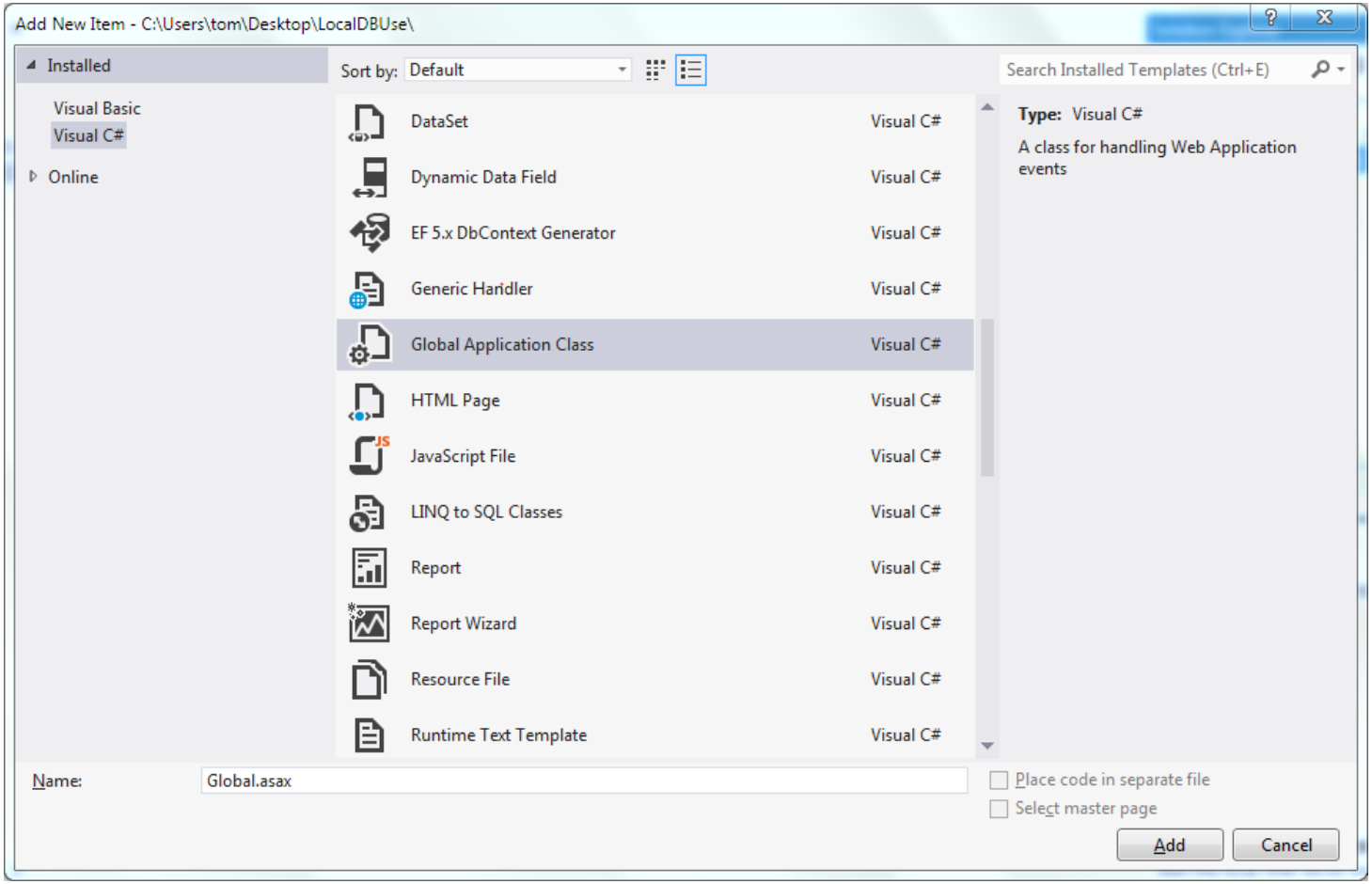
Configuration Inheritance

- Machine.config found in Windows\Microsoft.NET\Framework\vn.n.nnnn\Config
  Let's take a look….
- This is the master configuration file with systemwide defaults.
- Can be overridden in the web.config file in the application's root directory.
- Additional web.config files in subdirectories further override settings.

The Global.asax File

- Provides global application functionality
  - Global directives
  - Global event handlers
  - Global object tags
- Look over the HttpApplication class under System.Web.
  - This class is central to ASP.NET applications.

Adding Global.asax using VS 2013



Code Generated

```
<%@ Application Language="C#" %>

<script runat="server">

    void Application_Start(object sender, EventArgs e)
    {
        // Code that runs on application startup

    }

    void Application_End(object sender, EventArgs e)
    {
        //  Code that runs on application shutdown

    }

    void Application_Error(object sender, EventArgs e)
    {
        // Code that runs when an unhandled error occurs

    }
```

```
 void Session_Start(object sender, EventArgs e)
   {
      // Code that runs when a new session is started

   }

   void Session_End(object sender, EventArgs e)
   {
      // Code that runs when a session ends.
      // Note: The Session_End event is raised only
when the sessionstate mode
      // is set to InProc in the Web.config file. If session
mode is set to StateServer
      // or SQLServer, the event is not raised.

   }

</script>
```

Global Directives

- @ Application directives
  - Facilitates code behind for Global.asax
- @ Import directives
  - Imports a namespace (like for aspx files)
- @ Assembly directives
  - Specifies assemblies to be linked
- The key to the usefulness of these directives is event handlers.

Global Event Handlers

- Application wide events
- Most common are start and end events.
  - Note – an application ends 20 minutes after the last HTTP request. That's a relatively long time.
  - Default timeout can be changed
- Don't confuse application start/end with the concept of session start/end. An application can service many concurrent sessions from many users.

Example

```
<script language="C#" runat="server">
  void Application_Start ()
  {
    ...
  }

  void Application_End ()
  {
    ...
  }
```

Use for Session State

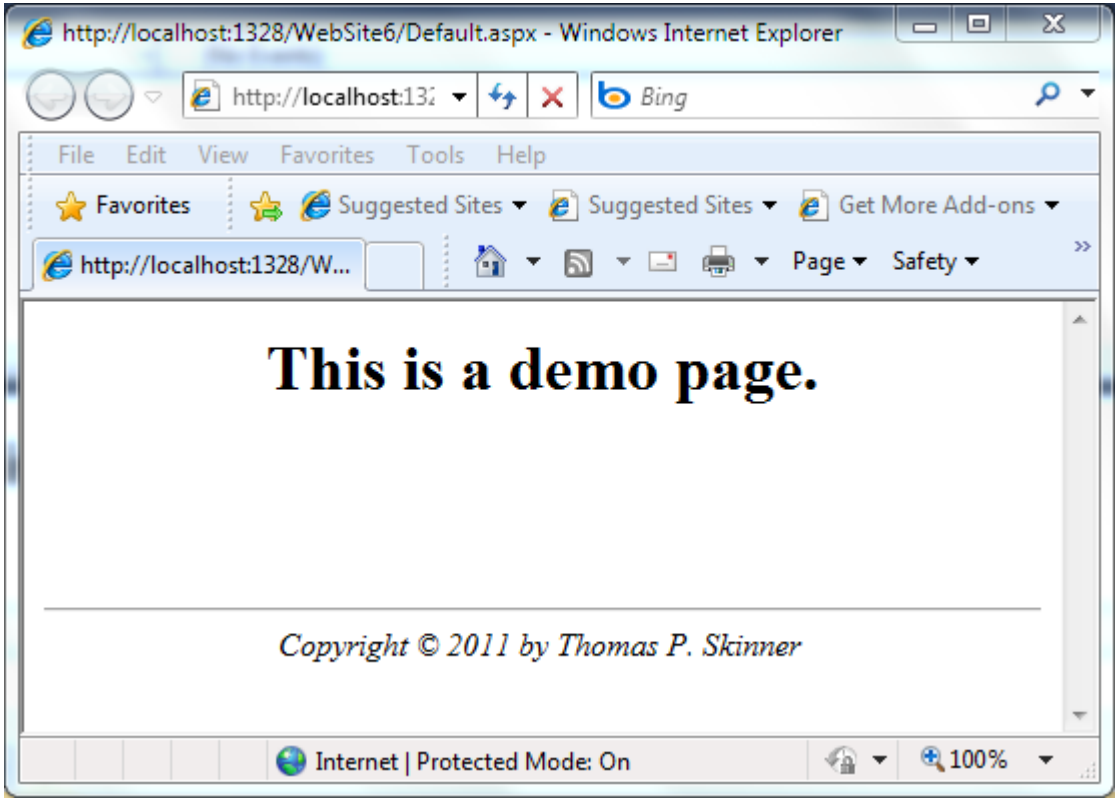
```
<script language="C#" runat="server">
  void Session_Start ()
  {
    ...
  }

  void Session_End ()
  {
    ...
  }
</script>
```

Per-Request Events

Used to customize ASP.NET

```
<script language="C#" runat="server">
  void Application_PostRequestHandlerExecute (Object   sender,
EventArgs e)
  {
      HttpApplication app = (HttpApplication) sender;  //cast
      app.Context.Response.Write ("<hr><center><i>" +
          "Copyright © 2013 by Thomas P. Skinner</i></center>");
  }
</script>
```

This code adds the copyright notice to all pages.

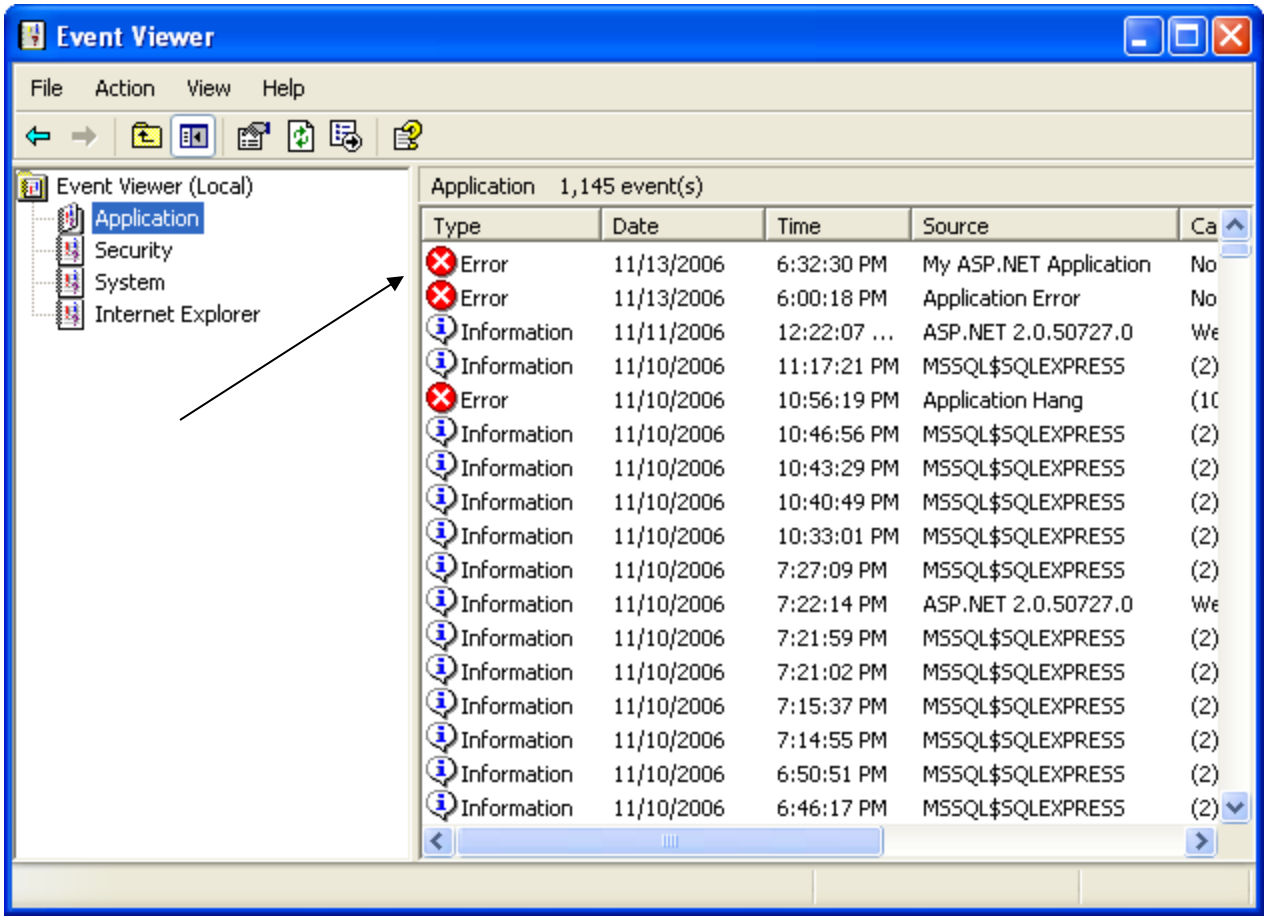

http://localhost:1328/WebSite6/Default.aspx - Windows Internet Explorer
http://localhost:13
Bing
File Edit View Favorites Tools Help
Favorites
Suggested Sites
Suggested Sites
Get More Add-ons
http://localhost:1328/W...
Page
Safety
This is a demo page.
Copyright © 2011 by Thomas P. Skinner
Internet | Protected Mode: On
100%

Error Events

- You can provide a custom error handler.
- Outputting a custom message, etc.

```
<%@ Import Namespace="System.Diagnostics" %>
<script language="C#“ runat="server">
   void Application_Error (Object sender, EventArgs e)
   {
       // Formulate a message to write to the event log
      string msg = "Error accessing " + Request.Path + "\n"
       +   Server.GetLastError ().ToString ();
      // Write an entry to the event log
      EventLog log = new EventLog ();
      log.Source = "My ASP.NET Application";
      log.WriteEntry (msg, EventLogEntryType.Error);
   } </script>
```

- The previous code outputs to the event log under the application tab.



Application State

- Store data that is available to all parts of an application. (note – not a session)
- Data is stored in memory so it is transient.
- Key/value pairs are stored.
- Data is anything derived from System.Object.
- Example:

```
Application[“Data1”] = 100;
Application[“Data2”] = 1000;
```

Application State

- You must cast the value when retrieving.

```
int i = (int)Application[“Data1”];
```

- Application.Lock() and Application.unlock() should be used to prevent concurrency related issues for multi-step updates.
- Individually, ASP.NET uses a reader/writer lock to synchronize access to individual pieces of state information.
- Demo – write an application to keep track of the number of sessions in progress. (ApplicationState web site example)

The Application Cache

- Better than application state.
- Can assign expiration policy.
- Can invoke a call-back method to refresh the data.
- Does not have lock and unlock methods.
- Instead use thread locking.
- Use of the cache can improve performance, e.g. when reading a file.
- See examples in Prosise book.

Session State

- Per user state information. (Normally what you want.)
- Default is to use a cookie.
- Sessions time out after 20 minutes of inactivity.
- A session object is created for each session.
- ASP Session state is stored in memory. ASP.NET has three options. (discussed later) that do not use memory.
- Memory is not good for large web sites running on server farms.

Session State – contd.

- Similar to application state.
- Use the indexer or Add method.

```
Session.Add(“Name”, “Tom”);
//or
Session[“Name”] = “Tom”;
```

- Semantically equivalent
- Both replace or add an item.
- Remember that objects are stored and casts are needed.
- A session cookie is saved on the client and expires when a new browser session is started.
- Demo (SessionState web site example)

Removing Items

- Use Remove, RemoveAt or RemoveAll methods.
- Clear is equivalent to RemoveAll.
- The SessionID property retrieves the unique identification of a session.
- Session state operations are thread safe.
  (Not sure how this works, but allegedly it does.)
- The Session.IsNewSession property can be used to determine the first time a page is loaded from the session. (see example in Prosise book)

Cookieless Session State

- A clever method to allow session state without cookies is possible.
- Set cookieless="true" to the sessionState element in web.config.
- URL munging is used. Extra characters are added to the URL that are stripped by ASP.NET at the server before accessing the page.
- Unfortunately there is no simple way to determine if cookies are supported. One method is to create a cookie and then see if it was stored on the client.

Memoryless Session State

- ASP.NET supports three process models:
  1. In-proc – stores in Aspnet_wp.exe (or whatever proces ASP runs in).
  2. State server – stores in a special state server process.
  3. SQL server – stores state in an SQL database.
  - These techniques are beyond the level of the course.

Session Lifetimes

- Default is 20 minutes.
- You can change this in web.config.

```
<sessionState … timeout="60"/>
```

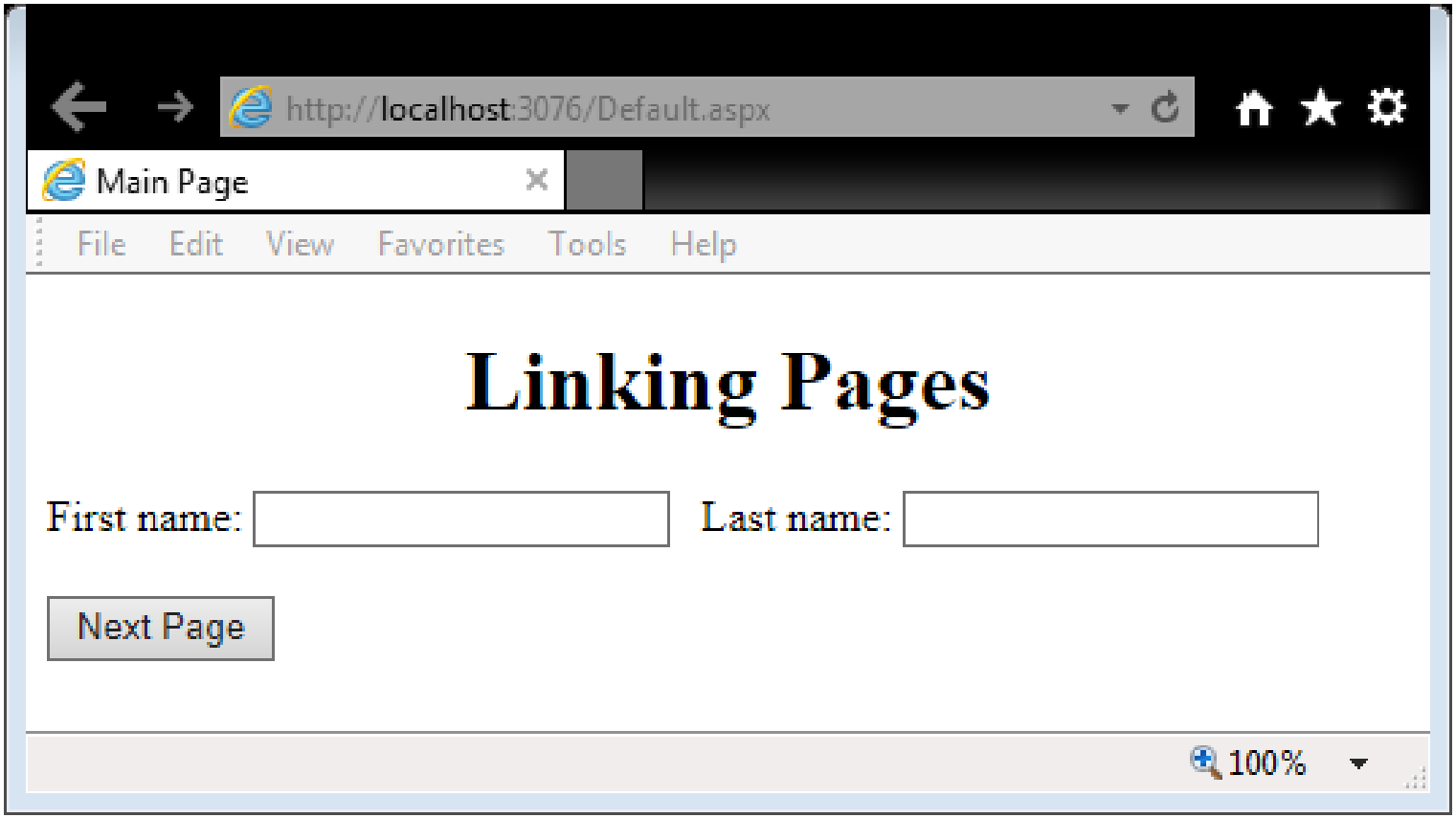
- The ellipses indicates other settings.
- You can also write it to the Session.Timeout property.

```
Session.Timeout = 60;
```

Linking Pages

- It is easy to pass data from one page to another.
- We use the CGI GET method to pass the data as key value pairs as part of the URL.
- We use the Response.Redirect and Request.QueryString methods.

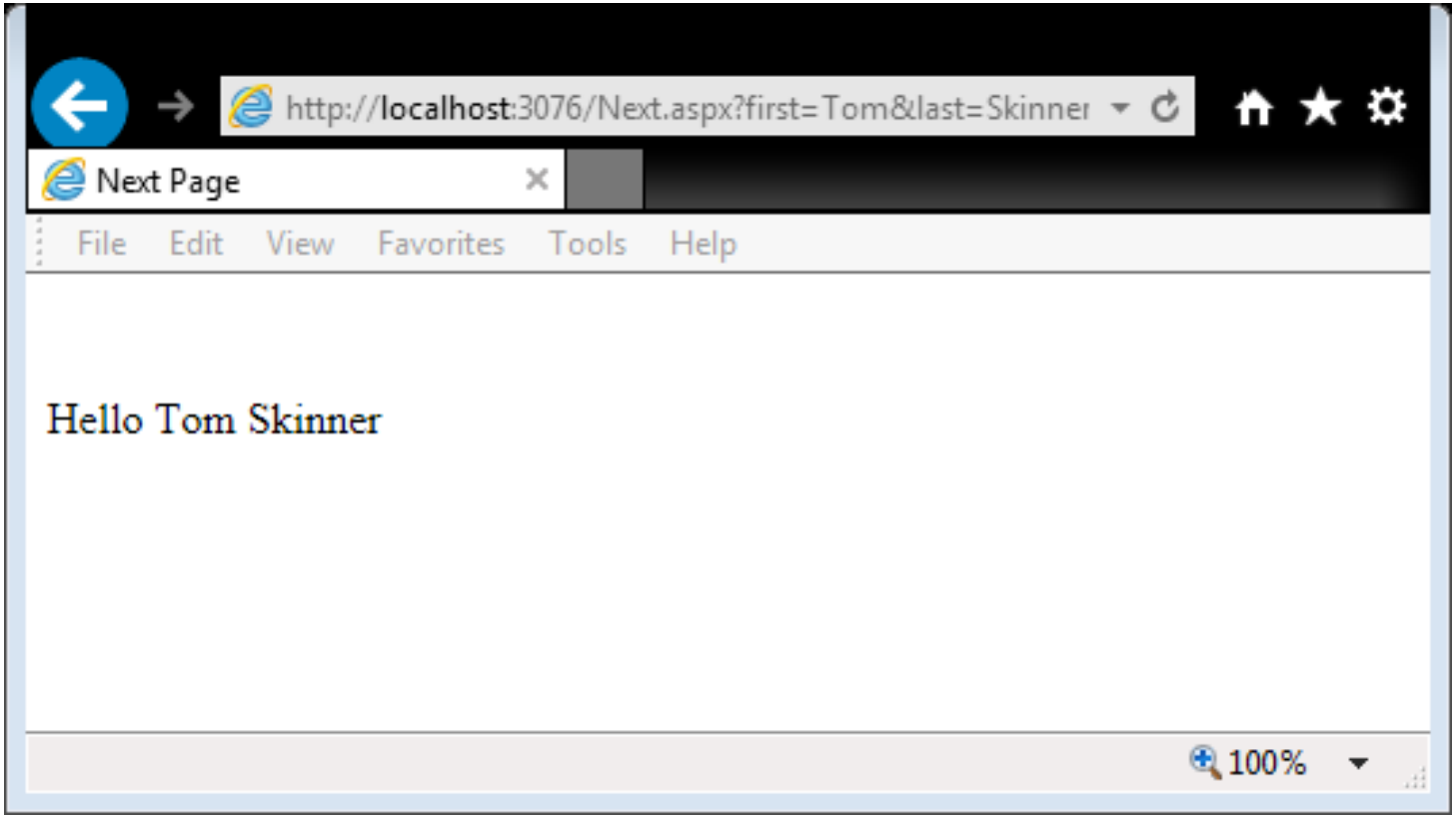
Default Page



Code Behind

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {

    }
    protected void Next_Click(object sender, EventArgs e)
    {
        string url = string.Format("Next.aspx?first={0}&last={1}",
first.Text, last.Text);
        Response.Redirect(url);
    }
}
```

The Linked Page

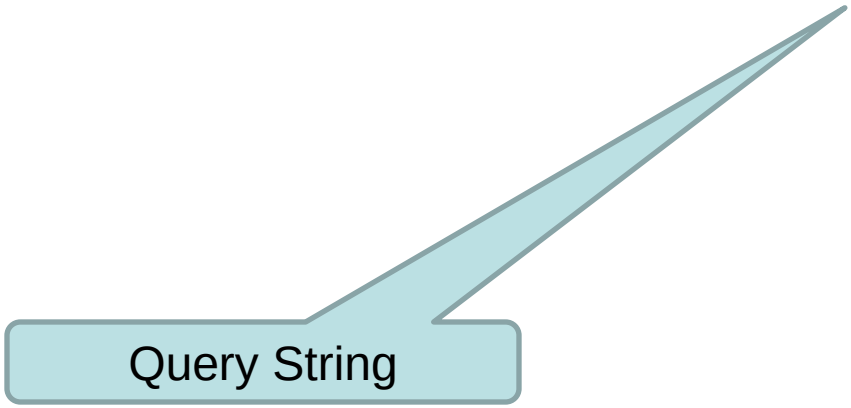


Code Behind

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

public partial class Next : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        string s = string.Format("Hello {0} {1}",
Request.QueryString["first"], Request.QueryString["last"]);
        greeting.Text=s;
    }
}
```

- The complete example is on the course web site under example code.
- Key is the URL

http://localhost:3076/Next.aspx?first=Tom&last=Skinner



Cookies

- Two types of cookies
  - Session
  - Persistent
- Very easy to use in ASP.NET
- Use HttpCookie object
- HttpCookie myCookie =
  new HttpCookie("name", "Tom");

Getting the cookie to the client

- The cookies collection in the Response object is used
- Response.Cookies.Add(myCookie);

Persistent Cookies

- By default cookies have session life
- We can set an expiration date to create a persistent cookie
- myCookie.Expires = new
  DateTime(2010, 12, 31);
- Expires 12/31/2010

Getting cookies from the client

- The Request object has a cookie collection

```
String name = "none";
HttpCookie myCookie = Request.Cookies["name"];
if(myCookie != null)
   name = myCookie.Value;
```

Multivalue Cookies

- Use the Values collection

  myCookie.Values["name"] = "Tom";
  myCookie.Values["sex"] = "male";
- Retrieve the values in the same way.