

## UNIT V

- 5.1 Web development and ASP.NET
- 5.2 Web applications and web servers
- 5.3 HTML form development
- 5.4 Client side scripting
- 5.5 GET and POST
- 5.6 ASP.NET application
- 5.7 ASP.NET namespaces
- 5.8 Creating sample C# web Applications.
- 5.9 Understanding Web Security
- 5.10 Windows authentication
- 5.11 Forms authentication
- 5.12 Web services
- 5.13 Web service clients
- 5.14 The CityView application.

### 5.1 Web development and ASP.NET

ASP.NET is a development framework for building web pages and web sites with HTML, CSS, JavaScript and server scripting.

ASP.NET supports three different development models: Web Pages, MVC (Model View Controller), and Web Forms:

.NET framework to be the most robust and flexible technology available for developing web applications. It is quick to develop, runs very fast and can also be used with other technologies which makes it very powerful.

#### **Web applications :**

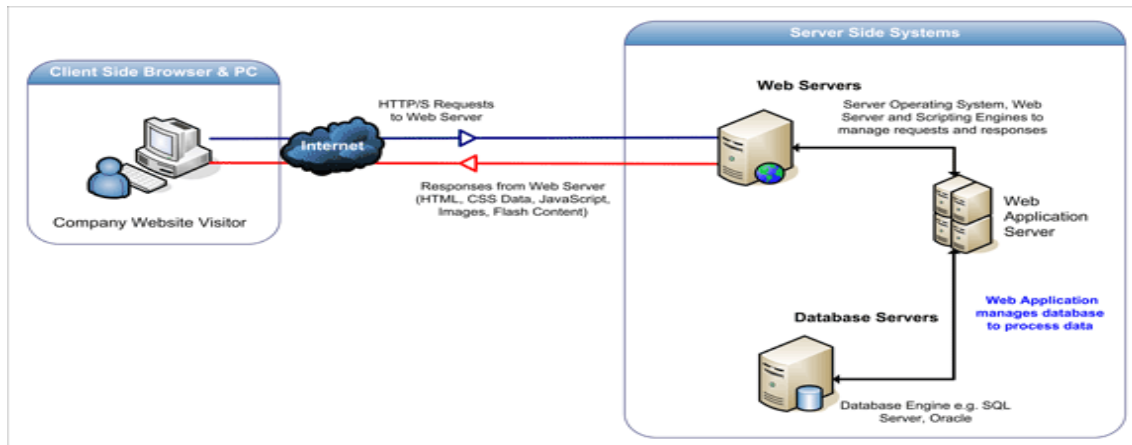
A web application or web app is any application software that runs in a web browser or is created in a browser-supported programming language (such as the combination of JavaScript, HTML and CSS) and relies on a common web browser to render the application.

#### **What is a Client?**

The 'client' is used in client-server environment to refer to the program the person uses to run the application. A client-server environment is one in which multiple computers share information such as entering information into a database. The 'client' is the application used to enter the information, and the 'server' is the application used to store the information.

Web applications commonly use a combination of server-side script (ASP, PHP, etc) and client-side script (HTML, Javascript, etc.) to develop the application. The client-side script deals with the presentation of the information while the server-side script deals with all the hard stuff like storing and retrieving the information.

## How web Application Work :



### Web Pages : Single Pages Model

- Simplest ASP.NET model.
- Similar to PHP and classic ASP.
- Built-in templates and helpers for database, video, graphics, social media and more.

### MVC

Model View Controller : MVC separates web applications into 3 different components:

- Models for data
- Views for display
- Controllers for input

### WebForms

Event Driven Model :

- The traditional ASP.NET event driven development model:
- Web pages with added server controls, server events, and server code.

## 5.2 Web applications and web servers

### Web application

web application or "web app" is a software program that runs on a web server. Unlike traditional desktop applications, which are launched by your operating system, web apps must be accessed through a web browser.

Web apps have several advantages over desktop applications. Since they run inside web browsers, developers do not need to develop web apps for multiple platforms. For example, a single application that runs in Chrome will work on both Windows and OS X. Developers do not need to distribute software

updates to users when the web app is updated. By updating the application on the server, all users have access to the updated version.

From a user standpoint, a web app may provide a more consistent user interface across multiple platforms because the appearance is dependent on the browser rather than the operating system. Additionally, the data you enter into a web app is processed and saved remotely. This allows you to access the same data from multiple devices, rather than transferring files between computer systems.

While web applications offer several benefits, they do have some disadvantages compared to desktop applications. Since they do not run directly from the operating system, they have limited access to system resources, such as the CPU, memory, and the file system. Therefore, high-end programs, such as video production and other media apps generally perform better as desktop applications. Web apps are also entirely dependent on the web browser. If your browser crashes, for example, you may lose your unsaved progress. Also, browser updates may cause incompatibilities with web apps, creating unexpected issues.

## **Web Servers**

A Web server is a program that, using the client/server model and the World Wide Web's Hypertext Transfer Protocol, serves the files that form Web pages to Web users (whose computers contain HTTP clients that forward their requests). Every computer on the Internet that contains a Web site must have a Web server program.

### **Examples:**

**Microsoft's Internet Information Server, which comes with the Windows NT server;**

**Netscape FastTrack and Enterprise servers;**

**Apache, a Web server for UNIX -based operating systems.**

Web servers often come as part of a larger package of Internet- and intranet-related programs for serving e-mail, downloading requests for File Transfer Protocol files, and building and publishing Web pages. Considerations in choosing a Web server include how well it works with the operating system and other servers, its ability to handle server-side programming, and publishing, search engine, and site building tools that may come with it.

## **Application Server**

Application servers are software that help enterprises develop, deploy and manage large numbers of applications that are mostly distributed in nature.

**A Web server exclusively handles HTTP requests, whereas an application server serves business logic to application programs through any number of protocols.**

## **5.3 HTML form development**

HTML forms are used to pass data to a server.

A form can contain input elements like text fields, checkboxes, radio-buttons, submit buttons and more. A form can also contain select lists, textarea, fieldset, legend, and label elements.

The <form> tag is used to create an HTML form:

```
<form>
.
  input elements
.
</form>
```

## HTML Forms - The Input Element

The most important form element is the input element.

The input element is used to select user information.

An input element can vary in many ways, depending on the type attribute. An input element can be of type text field, checkbox, password, radio button, submit button, and more.

The most used input types are described below :

### Text Fields

<input type="text" /> defines a one-line input field that a user can enter text into:

```
<form>
First name: <input type="text" name="firstname" /><br />
Last name: <input type="text" name="lastname" />
</form>
```

HTML code above looks in a browser as:

First name:  Last name:

**Note:** Default width of a text field is 20 characters.

### Password Field

<input type="password" /> defines a password field:

```
<form>
Password: <input type="password" name="pwd" />
</form>
```

HTML code above looks in a browser as:

Password: \$\$\$\$\$\$\$\$\$\$

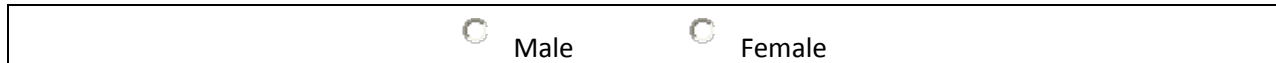
**Note:** The characters in a password field are masked (shown as asterisks or circles).

## Radio Buttons

`<input type="radio" />` defines a radio button. Radio buttons let a user select ONLY ONE of a limited number of choices:

```
<form>
<input type="radio" name="sex" value="male" /> Male<br />
<input type="radio" name="sex" value="female" /> Female
</form>
```

HTML code above looks in a browser as:

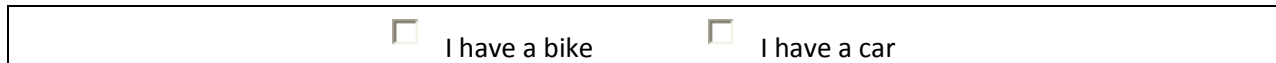


## Checkboxes

`<input type="checkbox" />` defines a checkbox. Checkboxes let a user select ONE or MORE options of a limited number of choices.

```
<form>
<input type="checkbox" name="vehicle" value="Bike" /> I have a bike<br />
<input type="checkbox" name="vehicle" value="Car" /> I have a car
</form>
```

HTML code above looks in a browser as:



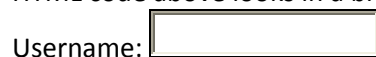
## Submit Button

`<input type="submit" />` defines a submit button.

A submit button is used to send form data to a server. The data is sent to the page specified in the form's action attribute. The file defined in the action attribute usually does something with the received input:

```
<form name="input" action="html_form_action.asp" method="get">
Username: <input type="text" name="user" />
<input type="submit" value="Submit" />
</form>
```

HTML code above looks in a browser as:



If you type some characters in the text field above, and click the "Submit" button, the browser will send your input to a page called "html\_form\_action.asp". The page will show you the received input.

## 5.4 Client side scripting

JavaScript is Netscape's cross-platform, object-oriented scripting language. Core JavaScript contains a core set of objects, such as Array, Date, and Math, and a core set of language elements such as operators, control structures, and statements. Core JavaScript can be extended for a variety of purposes by supplementing it with additional objects.

**Client-side JavaScript** extends the core language by supplying objects to control a browser (Navigator or another web browser) and its Document Object Model (DOM). For example, client-side extensions allow an application to place elements on an HTML form and respond to user events such as mouse clicks, form input, and page navigation.

**Server-side JavaScript** extends the core language by supplying objects relevant to running JavaScript on a server. For example, server-side extensions allow an application to communicate with a relational database, provide continuity of information from one invocation to another of the application, or perform file manipulations on a server.

### Example for Client Side Scripting :

```
<HTML> <center> <h1> Using JavaScript and HTML Forms </h1> </center>
```

```
<br> <br>
```

```
<form action="Z:\sathyabama\SATHYA\FLDR\IP\example\html\ex1.html" method="get" name="form1"
action="javascript:void(0)">
```

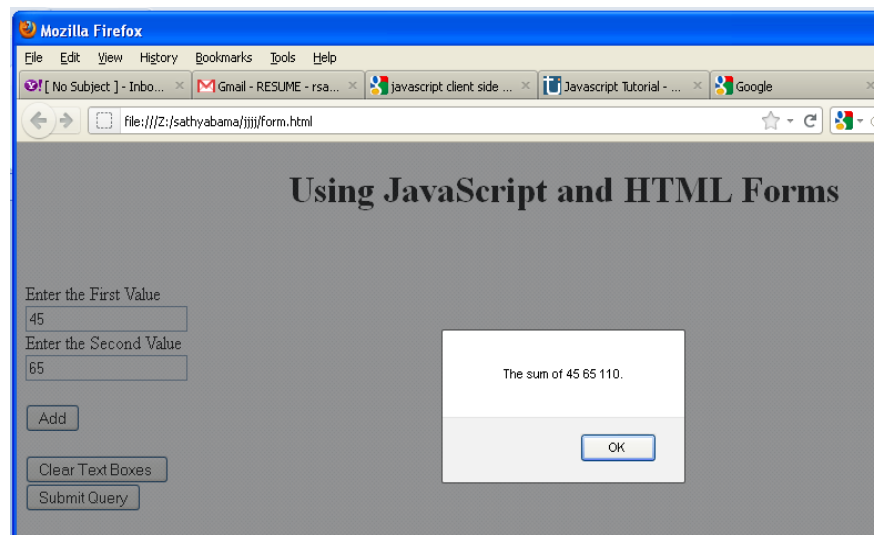
```
Enter the First Value <br> <input type="TEXT" name="first" size="20"
value=""> <br>
```

```
Enter the Second Value <br> <input type="TEXT" name="second" size="20"
value="" > <br> <br>
```

```
<input type="button" value="Add" NAME="btnbutton" onclick="add()">
```

```
<br> <br> <input type="RESET" value="Clear Text Boxes " > <br>
```

```
<input type="submit" NAME="ss"> </form>
```



## 5.5 GET and POST

The method attribute specifies how to send form-data (the form-data is sent to the page specified in the action attribute).

The form-data can be sent as URL variables (with method="get") or as HTTP post (with method="post").

### Notes on the "get" method:

- This method appends the form-data to the URL in name/value pairs
- This method is useful for form submissions where a user want to bookmark the result
- There is a limit to how much data you can place in a URL (varies between browsers), therefore, you cannot be sure that all of the form-data will be correctly transferred
- Never use the "get" method to pass sensitive information! (password or other sensitive information will be visible in the browser's address bar)

### Notes on the "post" method:

- This method sends the form-data as an HTTP post transaction
- Form submissions with the "post" method cannot be bookmarked
- The "post" method is more robust and secure than "get", and "post" does not have size limitations

### Syntax

`<form method ="get | post">`

### Attribute Values

Value	Description
get	Default. Appends the form-data to the URL in name/value pairs: URL? name=value&name=value
post	Sends the form-data as an HTTP post transaction

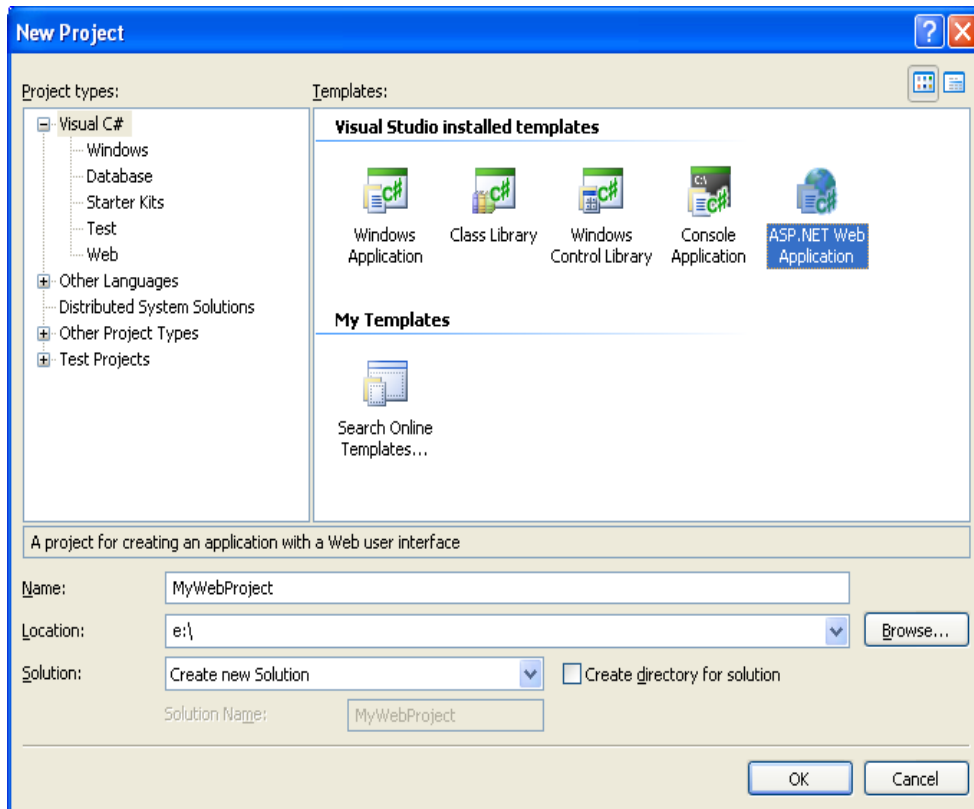
```
<form action="form_action.asp" method="get">  
  First name: <input type="text" name="fname" /><br />  
  Last name: <input type="text" name="lname" /><br />  
  <input type="submit" value="Submit" />  
</form>
```

## 5.6 ASP.NET Application

Creating, building and executing first web app using C# and the ASP.NET Web Application Project.

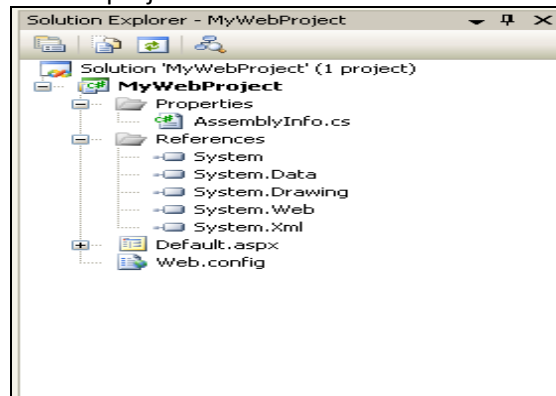
### Creating a New Project

Select File->New Project within the Visual Studio IDE. This will bring up the New Project dialog. Click on the "Visual C#" node in the tree-view on the left hand side of the dialog box and choose the "ASP.NET Web Application" icon:



Choose where you want the project to be created on disk (note that there is no longer a requirement for web projects to be created underneath the inetpub\wwwroot directory -- so you can store the project anywhere on your filesystem). Then name it and hit ok.

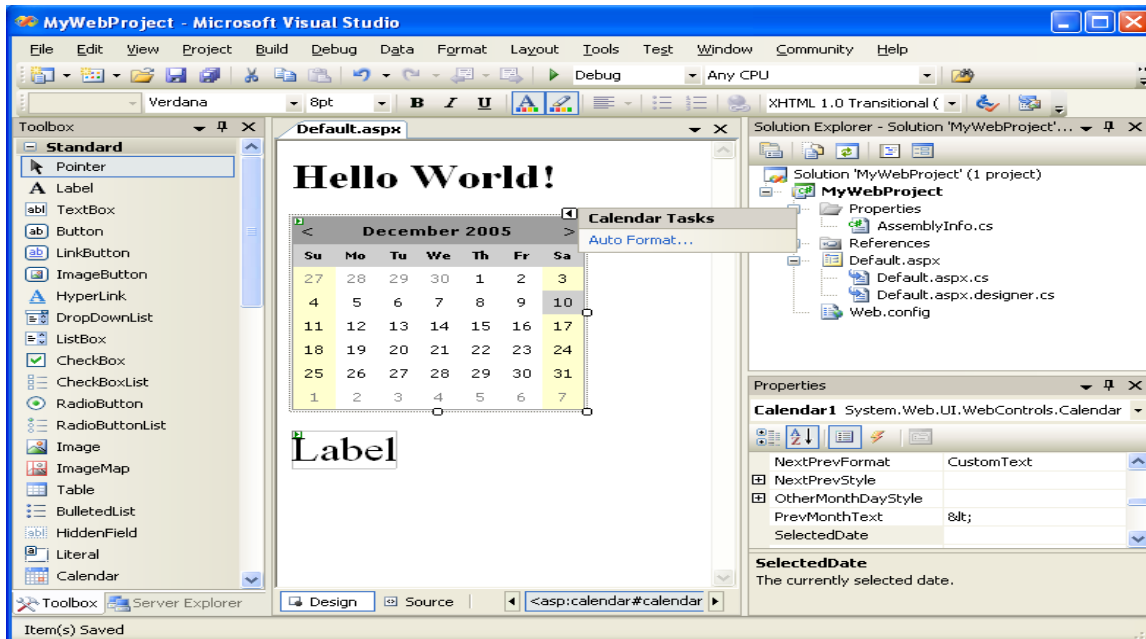
Visual Studio will then create and open a new web project within the solution explorer. By default it will have a single page (Default.aspx), an AssemblyInfo.cs file, as well as a web.config file. All project file-meta-data is stored within a MSBuild based project file.



## Opening and Editing the Page

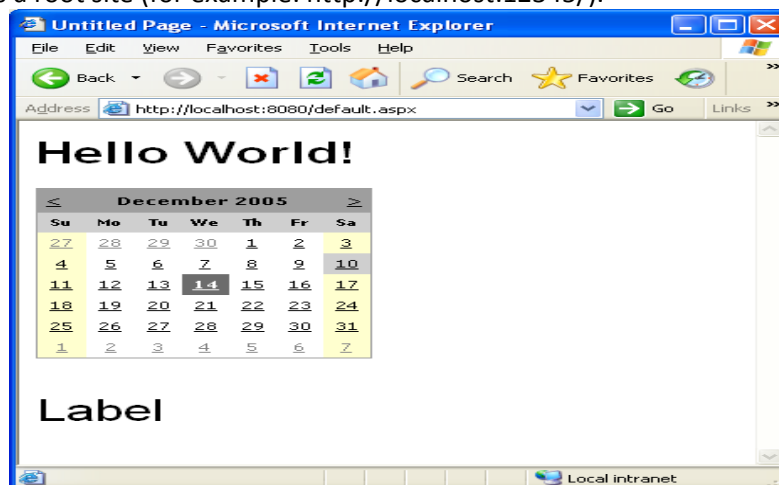
Double click on the Default.aspx page in the solution explorer to open and edit the page. You can do this using either the HTML source editor or the design-view. Add a "Hello world" header to the page, along with a calendar server control and a label control.





## Build and Run the Project

Hit F5 to build and run the project in debug mode. By default, ASP.NET Web Application projects are configured to use the built-in VS web-server (aka Cassini) when run. The default project templates will run on a random port as a root site (for example: <http://localhost:12345/>).



You can end the debug session by closing the browser window, or by choosing the Debug->Stop Debugging (Shift-F5) menu item.

## 5.7 ASP.NET Namespaces

Asp .net3.5 there are well over 30 web-centric namespace in the base class libraries.

From a high level, these namespace can be grouped into several major categories.

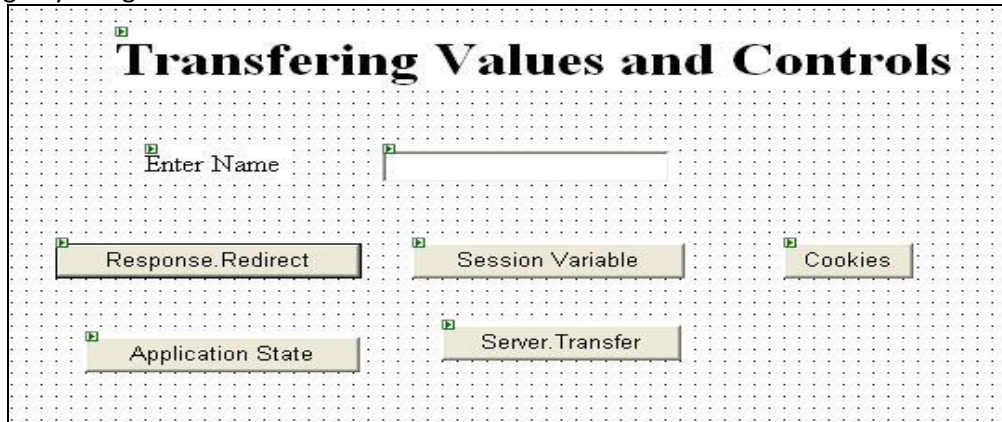
### 1.Core functionality

- 2.Web form and HTMLcontrols
- 3.Mobile web development
- 4.Silverlight development
- 5.Ajax development
- 6.XML web service

NAMESPACE	MEANING
1. System.Web	defines type that enable browser/web server communication.
2. System.Web.Caching	defines type that facilitate caching support for a web.
3. System.Web.Hosting	defines type that allow to build custom host for ASP.NET runtime.
4. System.Web.Management	defines type for monitoring & managing the health of an ASP.NET web application.
5. System.Web.Profile	defines type that are used to implement ASP.NET user profile.
6. System.Web.Security	defines type that allow you to programmatically secure your site
7. System.Web.SessionState	defines type that allow you to maintain stateful information on a per-user basis.
8. System.Web.UI, System.Web.UI.WebControls, System.Web.UI.HtmlControls.	defines a number of type that allow you to build a GUI front end for your own web application.

### **5.8 Creating sample C# web Applications.**

We always come into situations in which we need to transfer values from one page to another page. Some different ways of transferring values from page to page. The page created in this example is really simple which consists of a text field and a few buttons. The data entered in the text field will be transferred to another page by using different methods labeled on the buttons.



### **Introduction**

We always come into situations in which we need to transfer values from one page to another page. In this article, I will show you some ways of transferring values from page to page. The page I created in this example is really simple which consists of a text field and a few buttons. The data entered in the text field will be transferred to another page by using different methods labeled on the buttons.

## Response.Redirect

Let's first see how to transfer using Response.Redirect method. This maybe the easiest of them all. You start by writing some data in the text field, and when you finish writing the data, you press the button labeled 'Reponse.Redirect'. One tip that I would like to share with you is, sometimes we want to transfer to another page inside the catch exception, meaning exception is caught and we want to transfer to another page. If you try to do this, it may give you a System.Threading exception. This exception is raised because you are transferring to another page leaving behind the thread running. You can solve this problem using:

```
Response.Redirect("WebForm5.aspx",false);
```

This tells the compiler to go to page "WebForm5.aspx", and "false" here means that don't end what you were doing on the current page. You should also look at the System.Threading class for threading issues. Below, you can see the C# code of the button event. "txtName" is the name of the text field whose value is being transferred to a page called "WebForm5.aspx". "Name" which is just after "?" sign is just a temporary response variable which will hold the value of the text box.

```
private void Button1_Click(object sender, System.EventArgs e)
{ // Value sent using HttpResponse
  Response.Redirect("WebForm5.aspx?Name="+txtName.Text);
}
```

Okay, up till this point, you have send the values using Response. But now, where do I collect the values, so in the "WebForm5.aspx" page\_Load event, write this code. First, we check that the value entered is not null. If it's not, then we simply display the value on the page using a Label control. Note: When you use Response.Redirect method to pass the values, all the values are visible in the URL of the browser. You should never pass credit card numbers and confidential information via Response.Redirect.

```
if (Request.QueryString["Name"]!= null)
```

```
Label3.Text = Request.QueryString["Name"];
```

## Cookies

Cookies are created on the server side but saved on the client side. In the button click event of 'Cookies', write this code:

```
HttpCookie cName = new HttpCookie("Name");
cName.Value = txtName.Text;
Response.Cookies.Add(cName);
Response.Redirect("WebForm5.aspx");
```

First, we create a cookie named "cName". Since one cookie instance can hold many values, we tell the compiler that this cookie will hold "Name" value. We assign to it the value of the TextBox and finally add it in the Response stream, and sent it to the other page using Response.Redirect method. Let's see here how we can get the value of the cookie which is sent by one page.

```
if (Request.Cookies["Name"] != null )
  Label3.Text = Request.Cookies["Name"].Value;
```

As you see, it's exactly the same way as we did before, but now we are using Request.Cookies instead of Request.QueryString. Some browsers don't accept cookies.

## Session Variables

Session variables which are handled by the server. Sessions are created as soon as the first response is being sent from the client to the server, and session ends when the user closes his browser window or some abnormal operation takes place. Here is how you can use session variables for transferring values. Below you can see a Session is created for the user and "Name" is the key, also known as the Session key, which is assigned the TextBox value.

```
// Session Created
Session["Name"] = txtName.Text;
Response.Redirect("WebForm5.aspx");
// The code below shows how to get the session value.
// This code must be placed in other page.
if(Session["Name"] != null)
    Label3.Text = Session["Name"].ToString();
```

## Application Variables

Sometimes, we need to access a value from anywhere in our page. For that, you can use Application variables. Here is a small code that shows how to do that. Once you created and assigned the Application variable, you can retrieve its value anywhere in your application.

```
// This sets the value of the Application Variable
Application["Name"] = txtName.Text;
Response.Redirect("WebForm5.aspx");
// This is how we retrieve the value of the Application Variable
if( Application["Name"] != null )
    Label3.Text = Application["Name"].ToString();
```

## HttpContext

You can also use HttpContext to retrieve values from pages. The values are retrieved using properties or methods. It's a good idea to use properties since they are easier to code and modify. In your first page, make a property that returns the value of the TextBox.

```
public string GetName
{ get { return txtName.Text; } }
```

We will use Server.Transfer to send the control to a new page. Note that Server.Transfer only transfers the control to the new page and does not redirect the browser to it, which means you will see the address of the old page in your URL. Simply add the following line of code in 'Server.Transfer' button click event:

```
Server.Transfer("WebForm5.aspx");
Now, let's go to the page where the values are being transferred, which in this case is "webForm5.aspx".
Collapse | Copy Code
// You can declare this Globally or in any event you like
WebForm4 w;
// Gets the Page.Context which is Associated with this page
w = (WebForm4)Context.Handler;
// Assign the Label control with the property "GetName" which returns string
Label3.Text = w.GetName;
```

## Sample coding for server side script: [study your lab asp coding or the following code]

### Default.aspx.cs

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

namespace WebApplication3
{
    public partial class _Default : Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
        }

        protected void DropDownList1_SelectedIndexChanged(object sender, EventArgs e)
        {
        }

        protected void Button1_Click(object sender, EventArgs e)
        {
            if (t2.Text == "hello")
            {
                Session["NAME"] = t1.Text;
                Session["PASSWORD"] = t2.Text;
                Session["GENDER"] = r1.SelectedItem.Text;
                Session["cal"] = cd.SelectedDate;

                if (c1.Checked == true)
                {
                    Session["CRICKET"] = l1.Text;
                }
                else
                {
                    Session["CRICKET"] = " ";
                }
                if (c2.Checked == true)
                {
                    Session["FOOTBALL"] = l2.Text;
                }
                else
                {
                    Session["FOOTBALL"] = " ";
                }
                if (c3.Checked == true)
                {
                    Session["CHESS"] = l3.Text;
                }
                else
                {
                }
            }
        }
    }
}
```

```

        Session["CHESS"] = " ";
    }

    Session["COURSE"] = cb.SelectedItem.Text;
    Server.Transfer("WebForm1.aspx");
}
else
{
    b1.Attributes.Add("onclick", "return confirm('Enter valid password')");
}
}

protected void CheckBox1_CheckedChanged(object sender, EventArgs e)
{
}

protected void c3_CheckedChanged(object sender, EventArgs e)
{
}
}
}

```

#### WebForm1.aspx.cs

```

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

namespace WebApplication3
{
    public partial class WebForm1 : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            t11.Text = Session["NAME"].ToString();
            t12.Text = Session["Password"].ToString();
            t13.Text = Session["GENDER"].ToString();
            t14.Text = Session["COURSE"].ToString();
            tt.Text = Session["cal"].ToString();
            tt1.Text = Session["CRICKET"].ToString();
            tt2.Text = Session["FOOTBALL"].ToString();
            tt3.Text = Session["CHESS"].ToString();
        }
    }
}

```

## Default.aspx

USERNAME

PASSWORD

GENDER

MALE

FEMALE

HOBBY

CRICKET       FOOTBALL       CHESS

SELECT YOUR COURSE

CSE

March 2012						
Mon	Tue	Wed	Thu	Fri	Sat	Sun
27	28	29	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	1
2	3	4	5	6	7	8

SUBMIT

## Webform1.cs

NAME: Bibin

PASSWORD: hello

GENDER: MALE

COURSE: CSE

HOBBY: CRICKET  
FOOTBALL

DATE: 15-03-2012 00:00:00

## 5.9 Understanding Web Security

### ASP.NET Authentication

ASP.NET implements additional authentication schemes using authentication providers, which are separate from and apply only after the IIS authentication schemes. ASP.NET supports the following authentication providers:

- Windows (default)
- Forms
- Passport
- None

To enable an authentication provider for an ASP.NET application, use the authentication element in either machine.config or Web.config as follows:

## **5.10 Windows Authentication**

### **Configuring Windows Authentication**

To configure your application to use Integrated Windows authentication, you must use IIS Manager to configure your application's virtual directory security settings and you must configure the <authentication> element in the Web.config file.

To configure Windows authentication

- 1.Start Internet Information Services (IIS).
- 2.Right-click your application's virtual directory, and then click Properties.
- 3.Click the Directory Security tab.
- 4.Under Anonymous access and authentication control, click Edit.
- 5.Make sure the Anonymous access check box is not selected and that Integrated Windows authentication is the only selected check box.

In your application's Web.config file or in the machine-level Web.config file, ensure that the authentication mode is set to Windows as shown here.

```
<system.web>
  <!-- mode=[Windows|Forms|Passport|None] -->
  <authentication mode="Windows" />
</system.web>
```

Each ASP.NET authentication provider supports an OnAuthenticate event that occurs during the authentication process, which you can use to implement a custom authorization scheme. The primary purpose of this event is to attach a custom object that implements the [IPrincipal Interface](#) to the context.

Which ASP.NET authentication provider you use typically depends upon which IIS authentication scheme you choose. If you are using any of the IIS authentication schemes other than Anonymous, you will likely use the Windows authentication provider. Otherwise, you will use Forms, Passport, or None.

### **Windows**

The Windows authentication provider relies upon IIS to perform the required authentication of a client. After IIS authenticates a client, it passes a security token to ASP.NET. ASP.NET constructs and attaches an



object of the `WindowsPrincipal` Class to the application context based on the security token it receives from IIS.

- Authenticates using Windows accounts, so you do not need to write any custom authentication code.

### Con

- May require the use and management of individual Windows user accounts.

In addition, each IIS authentication scheme has its own associated pros and cons, which you should consider when choosing a security model.

### Authorizing Windows Users

When you use Windows authentication to authenticate users, you can use the following authorization options:

- File authorization provided by the `FileAuthorizationModule`.
- URL authorization provided by the `UrlAuthorizationModule`.

**Note** File authorization requires Windows authentication. The other authorization options are available with other authentication mechanisms.

```
<authorization>
  <deny users="DomainName\UserName" />
  <allow roles="DomainName\WindowsGroup" />
</authorization>
```

When you use Windows authentication, user names take the form `domainName\userName`. Windows groups are used as roles and they take the form `domainName\windowsGroupName`. Well known local groups such as Administrators and Users are referenced by using the "BUILTIN" prefix as shown here.

```
<authorization>
  <allow users="DomainName\Bob, DomainName\Mary" />
  <allow roles="BUILTIN\Administrators, DomainName\Manager" />
  <deny users="*" />
</authorization>
```

## 5.11 Forms Authentication

The Forms authentication provider is an authentication scheme that makes it possible for the application to collect credentials using an HTML form directly from the client. The client submits credentials directly to your application code for authentication. If your application authenticates the client, it issues a cookie to the client that the client presents on subsequent requests. If a request for a protected resource does not contain the cookie, the application redirects the client to the logon page. When authenticating credentials, the application can store credentials in a number of ways, such as a configuration file or a SQL Server database.

**Note** An ISAPI server extension only handles those resources for which it has an application mapping. For example, the ASP.NET ISAPI server extension only has application mappings for particular resources, such as `.asax`, `.ascx`, `.aspx`, `.asmx`, and `.config` files to name a few. By default, the ASP.NET ISAPI server

extension, and subsequently the Forms authentication provider, does not process any requests for non-ASP.NET resources, such as .htm, .jpg or .gif files.

### Pros

- Makes it possible for custom authentication schemes using arbitrary criteria.
- Can be used for authentication or personalization.
- Does not require corresponding Windows accounts.

### Cons

- Is subject to replay attacks for the lifetime of the cookie, unless using SSL/TLS.
- Is only applicable for resources mapped to Aspnet\_isapi.dll.

### Implementation

To implement forms authentication you must create your own logon page and redirect URL for unauthenticated clients. You must also create your own scheme for account authentication. The following is an example of a Web.config configuration using Forms authentication:

```
<!-- Web.config file -->
<system.web>
  <authentication mode="Forms">
    <forms forms="401kApp" loginUrl="/login.aspx" />
  </authentication>
</system.web>
```

Because you are implementing your own authentication, you will typically configure IIS for Anonymous authentication.

### Passport

The Passport authentication provider is a centralized authentication service provided by Microsoft that offers a single logon and core profile services for member sites. Passport is a forms-based authentication service. When member sites register with Passport, the Passport service grants a site-specific key. The Passport logon server uses this key to encrypt and decrypt the query strings passed between the member site and the Passport logon server.

### Pros

- Supports single sign-in across multiple domains.
- Compatible with all browsers.

### Con

- Places an external dependency for the authentication process.

## Implementation

To implement Passport, you must register your site with the Passport service, accept the license agreement, and install the Passport SDK prior to use. You must configure your application's Web.config file as follows:

```
<!-- Web.config file -->
<system.web>
  <authentication mode="Passport" />
</system.web>
```

## None (Custom Authentication)

Specify "None" as the authentication provider when users are not authenticated at all or if you plan to develop custom authentication code. For example, you may want to develop your own authentication scheme using an ISAPI filter that authenticates users and manually creates an object of the GenericPrincipal Class.

### Pros

- Offers total control of the authentication process providing the greatest flexibility.
- Provides the highest performance if you do not implement an authentication method.

### Cons

- Custom-built authentication schemes are seldom as secure as those provided by the operating system.
- Requires extra work to custom-build an authentication scheme.

## Implementation

To implement no authentication or to develop your own custom authentication, create a custom ISAPI filter to bypass IIS authentication. Use the following Web.config configuration:

```
<!-- Web.config file -->
<system.web>
  <authentication mode="None" />
</system.web>
```

## 5.12 Web services

### What are Web Services?

- Web services are application components
- Web services communicate using open protocols
- Web services are self-contained and self-describing
- Web services can be discovered using UDDI
- Web services can be used by other applications
- XML is the basis for Web services.

## **How Does it Work?**

### **The basic Web services platform is XML + HTTP.**

XML provides a language which can be used between different platforms and programming languages and still express complex messages and functions.

The HTTP protocol is the most used Internet protocol.

Web services platform elements:

- SOAP (Simple Object Access Protocol)
- UDDI (Universal Description, Discovery and Integration)
- WSDL (Web Services Description Language)

## **Why Web Services?**

A few years ago Web services were not fast enough to be interesting.

### **Interoperability has Highest Priority**

When all major platforms could access the Web using Web browsers, different platforms could interact. For these platforms to work together, Web-applications were developed.

Web-applications are simply applications that run on the web. These are built around the Web browser standards and can be used by any browser on any platform.

### **Web Services take Web-applications to the Next Level**

By using Web services, your application can publish its function or message to the rest of the world.

Web services use XML to code and to decode data, and SOAP to transport it (using open protocols).

With Web services, your accounting department's Win 2k server's billing system can connect with your IT supplier's UNIX server.

### **Web Services have Two Types of Uses**

#### **Reusable application-components.**

There are things applications need very often. So why make these over and over again?

Web services can offer application-components like: currency conversion, weather reports, or even language translation as services.

#### **Connect existing software.**

Web services can help to solve the interoperability problem by giving different applications a way to link their data.

With Web services you can exchange data between different applications and different platforms.

## Web Services Platform Elements

Web Services have three basic platform elements: SOAP, WSDL and UDDI.

### What is SOAP?

SOAP (Simple Object Access Protocol) is a messaging protocol that allows programs that run on disparate operating systems (such as Windows and Linux) to communicate using Hypertext Transfer Protocol (HTTP) and its Extensible Markup Language (XML). SOAP is a protocol for accessing a Web Service.

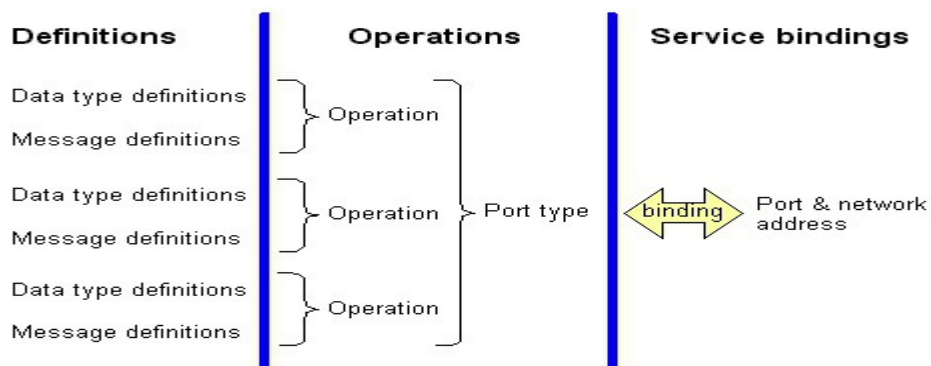
- SOAP stands for Simple Object Access Protocol
- SOAP is a communication protocol
- SOAP is a format for sending messages
- SOAP is designed to communicate via Internet
- SOAP is platform independent
- SOAP is language independent
- SOAP is based on XML
- SOAP is simple and extensible
- SOAP allows you to get around firewalls
- SOAP is a W3C standard

### What is WSDL?

Web Services Description Language (WSDL) is a format for describing a Web Services interface. It is a way to describe services and how they should be bound to specific network addresses. WSDL has three parts:

Definitions  
Operations  
Service bindings

The following figure shows the relationship of the basic parts of WSDL:



WSDL is an XML-based language for locating and describing Web services.

- WSDL is based on XML
- WSDL is used to describe Web services
- WSDL is used to locate Web services
- WSDL is a W3C standard

## What is UDDI?

Universal Description, Discovery, and Integration (UDDI) provides the definition of a set of services supporting the description and discovery of (1) businesses, organizations, and other Web Services providers, (2) the Web Services they make available, and (3) the technical interfaces which may be used to access those services. The idea is to "discover" organizations and the services that organizations offer, much like using a phone book or dialing information.

UDDI is a directory service where companies can register and search for Web services.

- UDDI is a directory for storing information about web services
- UDDI is a directory of web service interfaces described by WSDL
- UDDI communicates via SOAP
- UDDI is built into the Microsoft .NET platform

## Using the Web Service Example

These functions will send an XML response like this:

```
<?xml version="1.0" encoding="utf-8" ?>  
<string xmlns="http://tempuri.org/">38</string>
```

## Put the Web Service on Your Web Site

Using a form and the HTTP POST method, you can put the web service on your site, like this:

Fahrenheit to Celsius:	<input type="text"/>
	<input type="submit" value="Submit"/>
Celsius to Fahrenheit:	<input type="text"/>
	<input type="submit" value="Submit"/>

## How To Do It

Here is the code to add the Web Service to a web page:

```
<form action='tempconvert.aspx/FahrenheitToCelsius'  
method="post" target="_blank">  
<table>  
<tr>  
<td>Fahrenheit to Celsius:</td>  
<td>  
<input class="frmInput" type="text" size="30" name="Fahrenheit">  
</td>  
</tr>  
<tr>  
<td></td>  
<td align="right">  
<input type="submit" value="Submit" class="button">  
</td>  
</tr>  
</table>
```

```

</form>

<form action='tempconvert.asmx/CelsiusToFahrenheit'
method="post" target="_blank">
<table>
<tr>
<td>Celsius to Fahrenheit:</td>
<td>
<input class="frmInput" type="text" size="30" name="Celsius">
</td>
</tr>
<tr>
<td></td>
<td align="right">
<input type="submit" value="Submit" class="button">
</td>
</tr>
</table>
</form>

```

Substitute the "tempconvert.asmx" with the address of your web service like:  
<http://www.example.com/webservices/tempconvert.asmx>.

### **5.13 Web service clients**

One of the most simplest ways of adding a web service to a client application is by adding a web reference to the web service by specifying the URL of the *.asmx* file. This generates the required proxy object, that's what VS.NET takes care of. However, it may happen that after adding the web reference, the web service is moved to some other location. In such cases, the most easy way is to recreate the proxy object. But what if the same thing happens after you deploy your web service client. It would be nice to allow a configurable URL so that even if the original web service is moved, your client applications need not be recompiled.

Two web services are created here, one of which will have direct call and another web service will look into the web.config file for the reference.

#### **Create web service**

For our example, we will develop a simple web service that has only one method. The following steps will show you how to proceed.

- Create a new C# Web Service project in VS.NET.
- Open the default *.Asmx* file and add the following code to it:

```

using System;
using System.Web.Services;

namespace HWWebService
{

```

```

public class HWClass : System.Web.Services.WebService
{
    [WebMethod]
    public string HelloWorld()
    {
        return "Hello Application, from Web Service";
    }
}

```

As shown above, this web service class (HWClass) contains a single method called HelloWorld() that returns a string. Add another .asmx file to the project. Open the file and modify it as shown below:

```

using System;
using System.Web.Services;

namespace HWWebService
{
    public class AnotherService : System.Web.Services.WebService
    {
        [WebMethod]public string AnotherHelloWorld()
        {
            return "Hello World, from Another Service";
        }
    }
}

```

This class is similar to the previous one but its name is AnotherService. Also, it returns a different string from AnotherHelloWorld() method so that you can identify the method call. Now that we have both the web services ready, compile the project.

### Creating the web service client

Let us build a simple web client for our web service.

- Create a new ASP.NET web application in VS.NET.
- The application will have a default web form. Before writing any code, we need to add a web reference to our web service. Right click on the References node and select Add web reference. Follow the same procedure as you would have while developing normal web services. Adding a web reference will generate code for the proxy web service object.
- Place two text boxes. Name them as txtReturnWS and txtAnotherWS.
- Place two buttons on the web form and name them btnWS and btnAnotherWS. Add the following code in the Click event of the button:

```

private void btnWS_Click(object sender, System.EventArgs e)
{
    HelloWorld.HWServiceClass objProxy = new HelloWorld.HWServiceClass();
    objProxy.Url = GetHWServiceURL();
    txtReturnWS.Text = objProxy.HelloWorld();
}

```



- The above code shows how you will normally call a web service. The web reference contains information about the location of the web service.
- If you move the .asmx file after you deploy this client, it is bound to get an error. To avoid such a situation, modify the above code as shown below:

```
private void btnAnotherWS_Click(object sender, System.EventArgs e)
{
    AnotherWorld.AnotherService objProxy = new AnotherWorld.AnotherService();
    txtAnotherWS.Text = objProxy.AnotherHelloWorld();
}
```

- In above code, we have explicitly set the **Url** property of the **objProxy** class to the required .asmx file.
- You can store this URL in the **<appSettings>** section of the *web.config* file and retrieve it at run time. Now, even if you move your web service, all you need to do is change its URL in the *web.config*.
- You can add the URL of the webservice in the *web.config* file in two ways:
  1. directly by adding the required tag to the *web.config* file and
  2. by right-clicking the service used in the application, set its property to dynamic. This will automatically add the tag required for the service, you can change the name of the service to "HWServiceURL".

The following code shows this:

```
private void btnWS_Click(object sender, System.EventArgs e)
{
    localhost.HWClass objProxy=new localhost.HWClass;
    objProxy.Url=GetHWServiceURL();
    Response.Write(objProxy.HelloWorld());
}
public string GetHWServiceURL()
{
    return
        System.Configuration.ConfigurationSettings.AppSettings["HWServiceURL"];
}
```

- The *web.config* looks like this:

```
<appSettings>
  <add key="HWServiceURL"
    value="http://localhost/HWWebService/HWService.asmx" />
</appSettings>
```

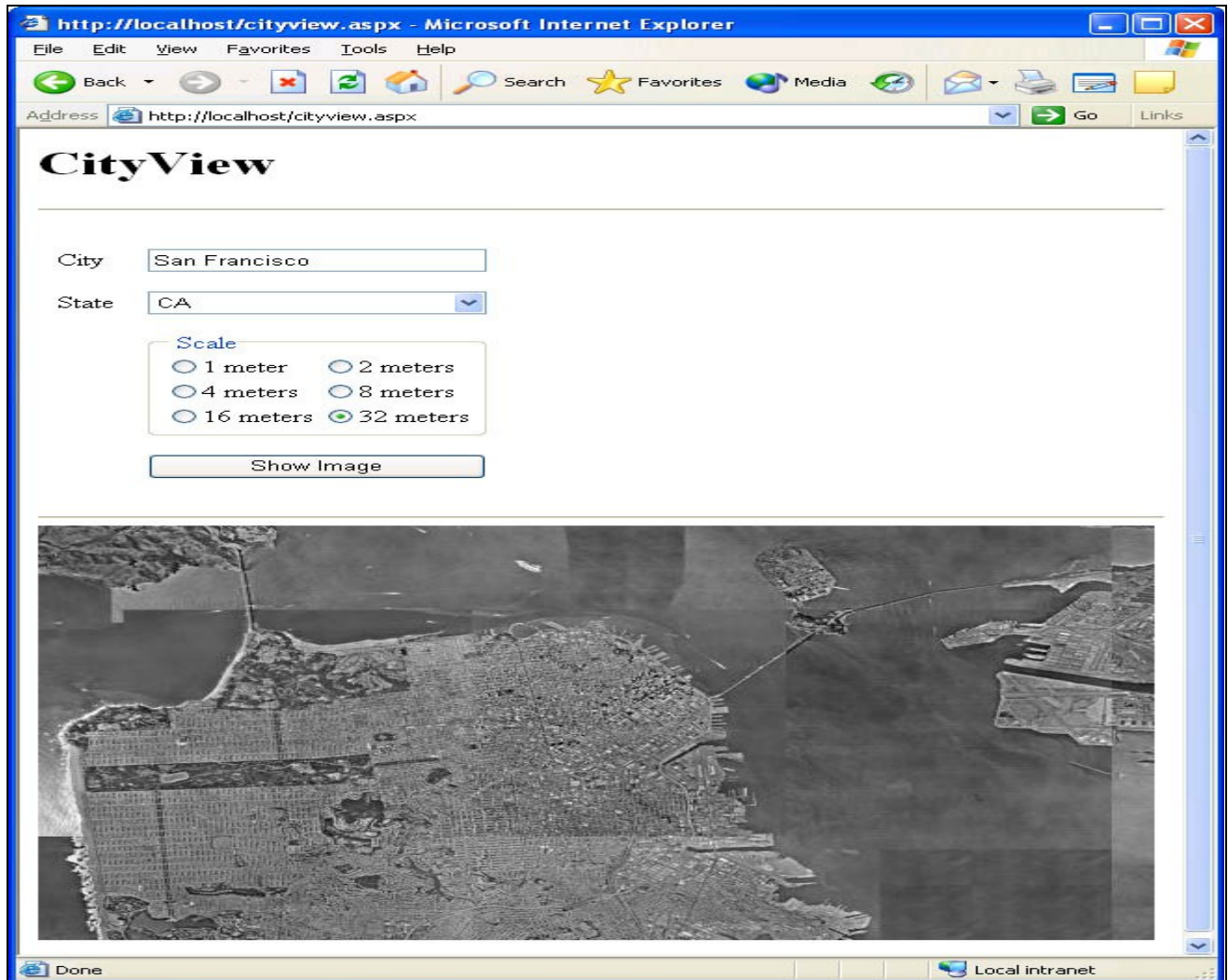
Note that in order for the above code to work correctly, both web services should have exactly same web method signatures.

## **5.14 The CityView application**

The Web application pictured in Figure CityView is a novel and graphic example of a Web service client. The Web service that it connects to is the Microsoft TerraService

(<http://terraservice.net/terraservice.aspx>), which is a Web service front end to the Microsoft TerraServer database. You can read all about TerraServer and TerraService at <http://terraservice.net/>.

TerraServer is one of the worlds largest online databases. Inside it are photographs and maps of much of Earths surface, made available to the public through a partnership between Microsoft and the U.S. Geological Survey. CityView showing an aerial view of San Francisco.



Now that CityView is installed, try it out by calling up CityView.aspx in your browser. Enter a city name (for example, New York) and pick a state. Then click Show Image. After a brief pause, the specified city appears at the bottom of the page. If CityView is unable to fetch the image you requested, it responds by displaying Image not available in place of the image. That could mean you entered a city that doesnt exist. Or it could mean that TerraService is temporarily down or your Internet connection is taking a nap.

You can zoom in and out by selecting different scales. The default scale is 8 meters. Choose a smaller number to zoom in or a larger number to zoom out. For a great aerial view of the San Francisco peninsula that includes an overhead shot of the Golden Gate Bridge, enter San Francisco, CA and choose a scale of 32 meters.

How CityView Works

CityView consists of three files:

- CityView.aspx
- CityView.ashx
- TerraService.dll

CityView.aspx is a Web form that defines CityViews user interface. Its source code appears in Figure 11-12. The user interface consists of a TextBox for typing city names, a DropDownList for selecting states, a RadioButtonList for choosing scales, and a Button for posting back to the server and fetching images. It also includes an Image control whose ImageUrl property is programmatically initialized following each postback. Heres the code that assigns a URL to the Image control:

```
MyImage.ImageUrl=?builder.ToString?();
```

If you enter Redmond, WA, and accept the default scale of 8 meters, the string assigned to ImageURL looks like this:

```
CityView.ashx?city=Redmond&state=WA&scale=8
```

which sets the stage perfectly for a discussion of the second component of CityView: namely, CityView.ashx.

CityView.ashx is an HTTP handler. Specifically, its an HTTP handler that generates and returns a bitmap image of the location named in a query string. When we deployed an HTTP handler in Chapter 8, we coded the handler in a CS file, compiled it into a DLL, and dropped the DLL into the application roots bin directory. We also registered the handler using a Web.config file. CityView.ashx demonstrates the other way to deploy HTTP handlers. You simply code an IHttpHandler-derived class into an ASHX file and include an @ WebHandler directive that identifies the class name and the language in which the class is written: `<%@?WebHandler?Language="C#" Class="CityViewImageGen" %>`

When a client requests an ASHX file containing a WebHandler class, ASP.NET compiles the class for you. The beauty of deploying an HTTP handler in an ASHX file is that you dont have to register the handler in a CONFIG file or in the IIS metabase; you just copy the ASHX file to your Web server. The downside, of course, is that you must test the handler carefully to make sure ASP.NET can compile it.

The CityViewImageGen class inside CityView.ashx (Figure 11-12) generates the images that CityView.aspx displays. Its heart is the ProcessRequest method, which is called on each and every request. ProcessRequest calls a local method named GetTiledImage to generate the image. Then it returns the image in the HTTP response by calling Save on the Bitmap object encapsulating the image:

```
bitmap.Save?(context.Response.OutputStream,?format);
```

Should GetTiledImage fail, ProcessRequest returns a simple bitmap containing the words Image not available instead of an aerial photograph. It also adjusts the format of the bitmap to best fit the type of content returned: JPEG for photographs, and GIF for bitmaps containing text.

GetTiledImage uses three TerraService Web methods:

- ConvertPlaceToLonLatPt, which converts a place (city, state, country) into a latitude and longitude
- GetAreaFromPt, which takes a latitude and longitude and an image size (in pixels) and returns an AreaBoundingBox representing the image boundaries
- GetTile, which takes a tile ID (obtained from the AreaBoundingBox) and returns the corresponding tile

A tile is a 200-pixel-square image of a particular geographic location. To build larger images, a TerraService client must fetch multiple tiles and stitch them together to form a composite. That's how GetTiledImage generates the 600 x 400 images that it returns. It starts by creating a Bitmap object to represent the image. Then it uses Graphics.DrawImage to draw each tile onto the image. The logic is wholly independent of the image size, so if you'd like to modify CityView to show larger (or smaller) images, find the statement

```
Bitmap?bitmap?=?GetTiledImage?(city,?state,?res,?600,?400);
```

in CityView.ashx and change the 600 and 400 to the desired width and height.

The third and final component of CityView is TerraService.dll, which contains the TerraService proxy class named TerraService. CityView.ashx's GetTiledImage method instantiates the proxy class and uses the resulting object to call TerraServices Web methods:

```
TS.TerraService?ts?=?new?TS.TerraService?();
```

TerraService.dll was compiled from TerraService.cs, which I generated with the following command:  
wsdl?/namespace:TS?http://terraservice.net/terraservice.asmx

The namespace was necessary to prevent certain data types defined in TerraServices WSDL contract from conflicting with data types defined in the .NET Framework Class Library. Once TerraService.cs was created, the command

```
csc?/t:library?terraservice.cs
```

compiled it into a DLL.

TerraService exposes TerraServers content via Web methods. There are 16 Web methods in all, with names such as ConvertPlaceToLonLatPt and GetTile. As you might expect, TerraService was written with the Microsoft .NET Framework. Its WSDL contract is available at <http://terraservice.net/terraservice.asmx?wsdl>.