

# **ASP.NET**

**Course Designer and Acquisition Editor**

**Centre for Information Technology and Engineering**

**Manonmaniam Sundaranar University**

**Tirunelveli**

---

# ASP.NET

---

---

## CONTENTS

Lecture 1	Introduction to ASP.NET	1
	<ul style="list-style-type: none"><li>o Webservice</li><li>o Virtual Directory</li><li>o Introduction to ASP.NET</li><li>o ASP.NET Architecture</li><li>o ASP.NET Features</li><li>o Advantages of ASP.NET</li><li>o Dataflow in ASP.NET</li><li>o A Simple ASP.NET Application</li></ul>	
Lecture 2	HTML Server-Side Controls	16
	<ul style="list-style-type: none"><li>o Basics of coding in ASP.NET</li><li>o Structure of code Blocks</li><li>o HTML Server-Side Controls</li></ul>	
Lecture 3	ASP.NET Web Controls – I	50
	<ul style="list-style-type: none"><li>o Web Controls</li><li>o Web Controls Hierarchy</li></ul>	
Lecture 4	ASP.NET Web Controls –II	62
	<ul style="list-style-type: none"><li>o ASP.NET Image Controls</li><li>o ASP.NET List Controls</li><li>o Link Button</li><li>o Panel</li><li>o AdRotator</li><li>o Calendar</li></ul>	
Lecture 5	Validation Controls	84
	<ul style="list-style-type: none"><li>o Validation in ASP.NET</li><li>o Required Field Validation</li><li>o Compare Validation</li><li>o Range Validator</li><li>o Regular Expression Validation</li><li>o Custom Validator</li><li>o Validation Summary Controls</li></ul>	
Lecture 6	Http Request	101
	<ul style="list-style-type: none"><li>o HTML Form</li><li>o HttpRequest Properties</li><li>o Cookies</li></ul>	
Lecture 7	HttpResponse	119

---

o	HttpResponse Properties	
o	HttpResponse Methods	
Lecture 8	Application, Session State Management and Cookies	127
o	HttpApplication	
o	Session	
o	Cookies	
o	Server Object	
Lecture 9	ADO.NET-I	145
o	Database	
o	DBMS	
o	DBMS Standardization	
o	Structured Query Language	
o	Using SQL as a DataQuery Language	
o	Manipulation of database in ASP.NET	
Lecture 10	ADO.NET-II	163
o	Managed Providers	
o	Connection Object	
o	Command Object	
o	DataReader	
Lecture 11	ADO.NET-II	179
o	DataSet	
o	DataTable	
o	Dataview	
Lecture12	Data Aware Control	197
o	Databinder	
o	Repeater Control	
Lecture13	DataGrid and DataList Server Controls	208
o	DataGrid Server Control	
o	DataList Server Controls	
Lecture14	Tracing, Error-Handling and Debugging	225
o	Tracing	
o	Error Handling	
o	Debugging	
Lecture15	Introduction to User Controls	248
o	Advantages of User Controls	
o	Creating a User Control	

---

---

Lecture16 ASP.NET Components	259
------------------------------	-----

---

- Disadvantages of COM
- ASP.NET Components
- Components and System Architecture
- Deploying Components in ASP.NET Applications

Syllabus	270
----------	-----

---

❧❧❧

## Lecture 1

---

# Introduction to ASP.NET

---

## Objectives

In this lecture you will learn the following

- + Knowing about Webserver & Virtual Directory
- + Learning the ASP.NET Architecture
- + Features of ASP.NET
- + Advantages of ASP.NET
- + Advantages of ASP.NET
- + A simple ASP.NET Application

---

## Coverage Plan

---

### Lecture 1

- 1.1 Snap Shot
- 1.2 Web Server
- 1.3 Virtual Directory
- 1.4 Introduction to ASP.NET
- 1.5 ASP.NET Architecture
- 1.6 ASP.NET Features
- 1.7 Advantages of ASP.NET
- 1.8 Dataflow in ASP.NET
- 1.9 A simple ASP.NET Application
- 1.10 Short Summary
- 1.11 Brain Storm

## 1.1 Snap Shot

This session deals with webservers, the difference between static and dynamic web pages, how to create a virtual directory. This session then proceeds on to ASP.NET, its architecture, its features and advantages. A Web server is a server, which uses the Hypertext Transfer Protocol (HTTP) and sends the files which consists of Web pages to Web users when requested. All the web pages are stored in a web server. ASP.NET is the next generation of Microsoft's Active Server Page (ASP), a feature of their Internet Information Server (IIS). ASP.NET allows to create Web pages dynamically by inserting queries to a relational database in the Web page.

## 1.2 Web Server

Generally, all the machines on the Internet can be categorized into two types namely servers and clients. Machines that provide services, like Web servers or FTP servers, to other machines are called as servers. Machines that are used to connect to those services are known as clients.

The term Server can be used for any of the following:

- A computer on a network that sends files to or runs applications for other computers on the network
- A software that runs on the server computer and performs the work of serving files or running applications
- A code that exchanges information with another upon request

**Radiant™**  
RAY OF HOPE

**ASP.NET**

- ⊕ Web Server is a program to server content over the Internet using HTML
- ⊕ Web server accepts requests from browser and then returns the appropriate HTML documents
- ⊕ Every computer on the Internet that contains a web site must have a web server program
- ⊕ Web servers are available for both Internet and Intranet related programs

Web server is a server to serve content over the Internet using the Hypertext Markup Language (HTML). The Web server accepts requests from browsers like Netscape and Internet Explorer and then returns the appropriate HTML documents. Every computer on the Internet that contains a web site must have a Web server program.

Client machine running a web browser

Server machine running a web server

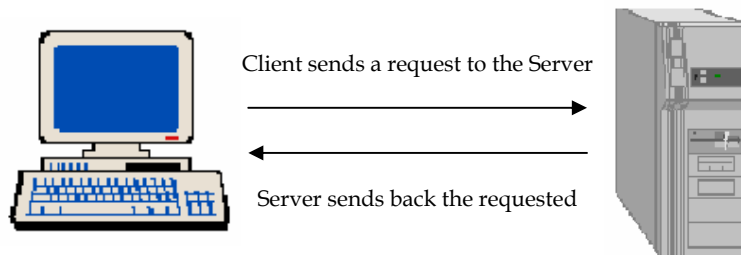


Figure 1.1



When the client asks for information such as a file, the Web server gets the file and sends it to the client. In most cases, the Web server does not read or process this file, but sends it to the client.

A number of Server-Side technologies can be used to increase the power of the server beyond its ability to deliver standard HTML pages, which include:

- CGI scripts
- Server-Side Includes (SSI)
- Active Server Pages (ASP)

The most popular Web servers are Microsoft's Internet Information Server (Internet Information Server), which comes with the Windows NT server, Netscape FastTrack and Enterprise servers and Apache server, a Web server for UNIX-based operating systems. Other Web servers include Novell's Web Server for users of its NetWare operating system and IBM's family of Lotus Domino servers, primarily for IBM customers.

Web servers are available for both Internet and Intranet related programs. Some of the functionalities include:

- Serving E-Mail
- Downloading requests for File Transfer Protocol files
- Building and publishing Web pages

Considerations in choosing a Web Server include the ability to work with the operating system and other servers. Selection should also be based on the ability to handle Server-Side programming, publishing, search engine, and site building tools that accompany it.

### **Web Page**

Web page is a World Wide Web document. A Web page typically consists of an HTML file, with associated files for graphics and scripts, in a particular directory on a particular computer (and thus identifiable by a URL). Web pages can be classified as static or dynamic web pages.

A **static web page** is a page whose content is changed before being sent over the Internet to the user's computer. Such pages appear to the user to be **static**. An example of this kind of a web page is a page that is created by a search engine. The static web page will not have any animation or lively user interactions.

A **dynamic web page** is a page that contains animated GIFs, Java applets, ActiveX Controls, or DHTML. It is a Web page created automatically based on information provided by the user, or generated with ASP.

### 1.3 Virtual Directory

**Radiant™**  
RAY OF HOPE

**ASP.NET**

- ⊕ Virtual directory is created in order to publish from any directory not contained within the home directory
- ⊕ Virtual directory is not contained in the home directory
- ⊕ Virtual directory has an *alias*, a name that web browsers use to access that directory

All the documents of a web site are stored in directories. The Web server cannot publish documents that are not within these specified directories. The first step in deploying a web site is to determine how the files need to be organized.

If a special directory is not created to store the documents, there is a default home directory in which all the documents can be published by copying the Web files into the default home directory.

**Home directory** is the root directory for a web site, where the content files are stored. It is also called as a document root or Web root. In Internet Information Services, the home directory and all its subdirectories are available to users by default. Normally, the home directory for a site contains the home page.

A **Virtual directory** is created in order to publish from any directory not contained within the home directory. A virtual directory is not contained in the home directory but appears to client browsers as though it were. A virtual directory has an *alias*, a name that Web browsers use to access that directory. Since an alias is usually shorter than the path of the directory, it is more convenient for users to type the alias.

An alias is more secure and advantageous as:

- Users do not know where the files are physically located on the server
- Users cannot utilize that information to modify the files
- Makes it easier to move directories in the site
- Instead of changing the URL for the directory, it is possible to change the mapping between the alias and physical location of the directory

#### Creating a Virtual Directory

**Radiant™**  
RAY OF HOPE

**ASP.NET**

Creating a Virtual Directory

- ⊕ In the Internet Information Services snap-in, the web site must be selected to which a directory is added
- ⊕ Click the Action button, point to New, and select the Virtual Directory
- ⊕ Use the New Virtual Directory Wizard to complete the task

As seen in the previous section, if the web site contains files that are located on a drive different from the home directory, or on other computers, virtual directories must be created to include those files in the web site. To use a directory on another computer, the directory's Universal Naming Convention (UNC) name must be specified and the user must be provided with a user name and password for access permission.

The following steps are used to create a virtual directory.

- In the Internet Information Services snap-in, select the web site or FTP site to which a directory has to be added
- Click the **Action** button, and then point to **New**, and select **Virtual Directory**
- Use the **New Virtual Directory Wizard** to complete the task

**Note :** While using NTFS, a virtual directory can be created by right-clicking a directory in the Windows Explorer, clicking **Sharing**, and then selecting the **Web Sharing** property sheet.

To delete a virtual directory the following steps have to be followed:

- In the Internet Information Services snap-in, select the virtual directory that is to be deleted
- Click the **Action** button, and select **Delete**

An ASP.NET application can be stored either in the Home or Virtual directory.

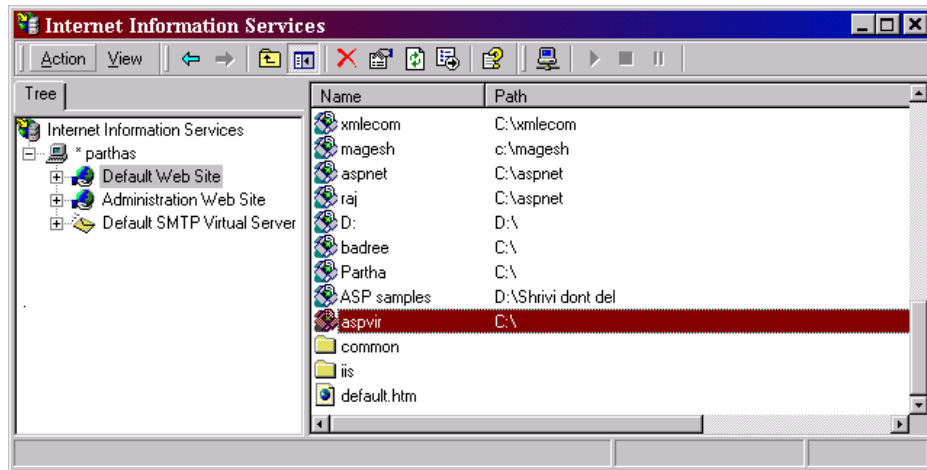


Figure 1.2

## 1.4 Introduction to ASP.NET

ASP.NET is an entirely new framework from Microsoft for building Next Generation Web Applications. ASP.NET is component-based and modularized. Every page, object, and HTML element in ASP.NET can be accessed at runtime as a component.

The key component of the new .NET framework has the potential to save organizations time and money by allowing them to establish object-oriented frameworks for the web applications. Microsoft has done an excellent job by automating common web development tasks. But more importantly they have created a tool that gives developers the ability to handle any business problem.

**What is ASP.NET?**

Radiant™

ASP.NET

RAY OF HOPE

What is ASP.NET?

- ⊕ A revolutionary, new programming framework
- ⊕ Enables the rapid development of powerful web applications and services
- ⊕ Most scalable way to build, deploy and run distributed web applications
- ⊕ Can target and browser or device

ASP.NET is a revolutionary new programming framework that enables the rapid development of powerful web applications and services. Part of the emerging Microsoft .NET Platform, it provides the easiest and most scalable way to build, deploy and run distributed web applications that can target any browser or device.

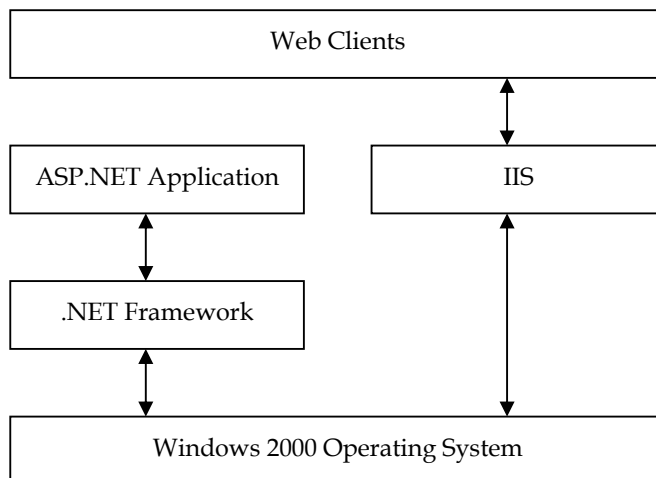
**How ASP.NET uses .NET framework**

ASP.NET is a compiled .NET-based environment. The developer can author applications in any .NET compatible language, including Visual Basic, C# and Jscript.NET. Additionally, the entire .NET Framework is available to any ASP.NET application. Developers can easily access the benefits of these technologies, which include a managed Common Language Runtime environment, type safety, inheritance, and so on.

ASP.NET has been designed to work seamlessly with WYSIWYG HTML editors and other programming tools, including Microsoft Visual Studio.NET. Not only does this make Web development easier, but also provides all the benefits that these tools have to offer, including a GUI that developers can use to drop server controls onto a Web page, as well as fully integrated debugging support.

**1.5 ASP.NET Architecture**

This section provides an overview of the ASP.NET infrastructure and the subsystem relationships. These relationships are shown in the following illustration:

*Figure 1.3*

As shown, all Web clients communicate with ASP.NET applications through IIS. IIS deciphers the request, authenticates, and finds the requested resource (such as an ASP.NET application). If authorized, it returns the appropriate resource to the client. In addition to the built-in ASP.NET security features, an ASP.NET application can use the Common Language Runtime low-level security features.

### Integrating with IIS

This release of ASP.NET uses IIS 5.0 as the primary host environment. When considering ASP.NET authentication, the interaction with IIS authentication services must be understood.

There are three different kinds of authentication in IIS 5.0: Basic, Digest, and Integrated Windows Authentication. The type of authentication used can be set in the IIS administrative services. If a URL containing an ASP.NET application is requested, both the request and information about authentication get handed over to the application. ASP.NET provides two additional types of authentication: Cookie authentication and Passport authentication.

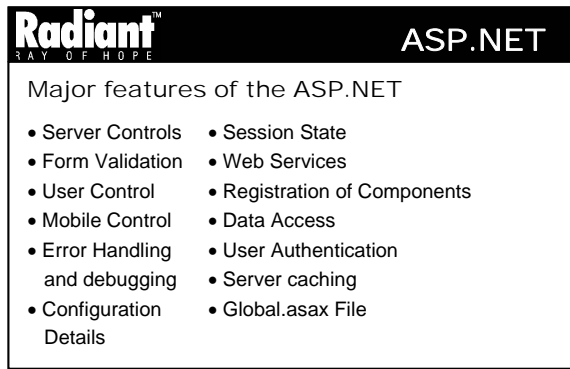
### Cookie-Based Authentication

Cookie authentication is a process that allows the application to collect credentials directly from the client requestor (usually name and password), and determine their authenticity. IIS authentication is not used by the application however, the IIS authentication settings are important to the ASP.NET cookie authentication process. Unless the IIS "Anonymous Access" setting is enabled, requests that do not meet the criteria for IIS authentication will be rejected and never reach the ASP.NET application.

### Passport Authentication

Passport authentication is a centralized authentication service provided by Microsoft that offers a single sign-in and core profile services for member sites.

## 1.6 ASP.NET Features



The major features of the ASP.NET are as follows:

### Server Controls

ASP.NET introduces the concept of Server Controls, which enable the values of the controls to be manipulated on the Server-Side.

An amazing feature of the Server-Side controls is the ability of these controls to fire Server-Side events. Thus, the control appearing at the client, with runat attribute set to server can fire Server-Side events. This is particularly useful when validating and dynamically generating forms.

The Server-Side controls can be subdivided into the following categories based on the functionalities that they offer.

### **Intrinsic Controls**

These controls create HTML elements on the client browser with the added capability of maintaining the state.

### **List Controls**

These Controls are used to display a wide variety of lists on the Client-Side.

### **Rich Controls**

These are used to output HTML in a rich format to the client. The calendar control is an example of this category.

### **Validation Controls**

These make the art of Server-Side validation a cakewalk.

### **Session State**

In ASP.NET, the Session state is automatically maintained by using a hidden field called view state. Further, ASP.NET offers the flexibility to turn the Session state on or off at both the page level as well as the control level. It is also possible to store the Session state directly in the SQL Server database and retrieve at a later time.

### **Form Validation**

ASP.NET offers a rich set of server controls to achieve the task of Form Validation. These controls are popularly known as Validators or validation controls and can be used to validate all types of validations.

### **Web Services**

ASP.NET offers a new concept called Web Services by which a web application can expose its functionality to others on the Internet by employing a protocol such as Simple Object Access Protocol (SOAP) and in XML format. This is entirely a new concept. Web Services can be employed for displaying news headlines, weather reports etc.

### **User Controls**

ASP.NET offers code reusability by introducing the concept of "User Controls." A User Control is a pseudo control, wrapped with common code that can expose properties, methods and events. These can be employed for opening and closing a database connection, which happen quite frequently in all the pages. User Controls can be employed to wrap up the most frequently used HTML code and reuse it across several Web pages.

### **Registration of Components**

In web development, it is a common practice to apply the business logic as COM Components. This concept has been received very well by the developers, as there is a performance boost when such business objects are employed. This performance boost is due to the fact that the components are compiled. But the maintenance poses a great problem. If a component has to be unregistered from use by the web server, the server has to be restarted.

ASP.NET has the solution for this problem. If a component is to be registered in ASP.NET, it has to be copied in the web application's "bin" directory. The file has to be deleted to unregister the component.

### **Mobile Controls**

ASP.NET is in tune with M-Commerce by providing .NET Mobile Web SDK that allows applications to be built covering a wide range of mobile devices ranging from cellular phones to Personal Digital Assistants.

### **Data Access**

ASP.NET has introduced the next generation of ADO known as ADO.NET with respect to data access. ADO.NET places more emphasis on disconnected recordsets by employing XML as a medium of communication between these recordsets and the DataStore.

### **Error Handling and Debugging**

ASP.NET offers a cleaner approach to error handling and debugging as compared to its predecessor ASP. It is now possible to specify individual error pages for each ASP.NET page by employing the new ErrorPage directive.

```
<%@page Errorpage = "/myErrorPage.aspx"%>
```

In the above example, if any error occurs on the page, the user is directed to myErrorPage.aspx. This type of error handling enables developers to circumvent the IIS error pages altogether.

### **User Authentication**

ASP.NET offers several kinds of authentication for checking the validity of the users accessing a web site. In addition to support for Basic, Digest and Windows NT(NTLM), ASP.NET also supports Passport authentication (explained later).

### **Server-Side Caching**

The process of retaining most frequently visited web pages in memory is called caching. Caching of ASP.NET Page leads to an increase in its performance. ASP.NET supports the following two types of caching

- Output caching
- Data Caching

Output caching is the process of caching the entire page, this kind of caching is advisable for sites with heavy traffic. Data caching is the process of caching a part of the page that includes objects and data.

### **Configuration Details**

ASP.NET maintains all the configuration details pertaining to a web application in a file called Config.web which is in a human readable format (XML). The whole range of configuration details including which request to handle on specific situation, tracing options etc.

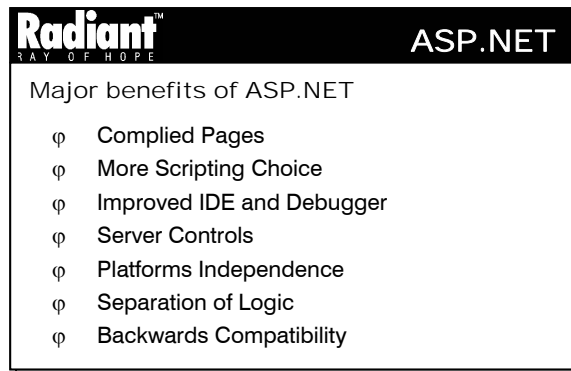
### **Global.asax File**

In ASP.NET it is possible to specify the event-handlers that need to be invoked on the occurrence of particular events. These event-handlers are defined in a global file by name Global.asax. The Global.asax file of ASP.NET has a provision for the following event-handlers

- Application\_OnStart
- Application\_OnEnd
- Session\_OnStart
- Session\_OnEnd
- Application\_BeginRequest
- Security\_OnAuthenticate

Finally, ASP.NET provides a powerful framework for developing and deploying Next Generation Web Applications. ASP.NET will have a significant impact on the web application development in future.

## 1.7 Advantages of ASP.NET



The greatly improved ASP.NET runtime handles many routine tasks in the code. Although there are major benefits of ASP.NET, here are some that will prove beneficial to the majority of the development tasks.

### Compiled Pages

A page is compiled into a .NET class when it is requested the first time. The runtime engine will detect if any changes have been made to the source code and recompile it if necessary. The user can optionally specify a class file with the page, otherwise it will be created by the ASP runtime.

### More Scripting Choice

The user can code pages with Visual Basic, C#, or JScript.NET. VBScript has been incorporated into the VB syntax.

### Improved IDE and Debugger

The entire .NET family shares the same IDE and debugging tools. The user will benefit from the feature rich debugging tools of ASP.NET.

### Server Controls

Server-Side components automate many common development tasks. These controls can detect the version of the browser and generate the proper HTML and JavaScript code.

### Browser Independence



The ASP.NET runtime is capable of detecting the browser type of the requesting client and generate the proper HTML or JavaScript code. This has the potential of making ASP.NET applications browser independent.

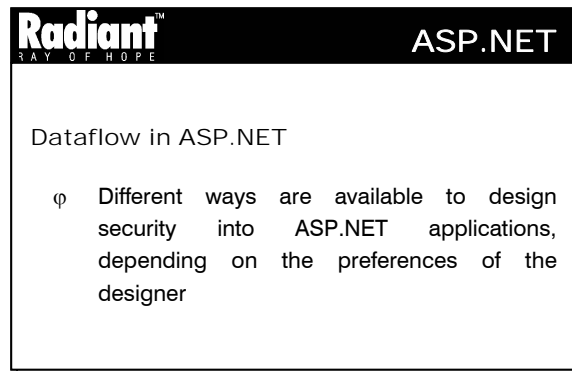
### Separation of Logic

ASP.NET has been designed to separate business logic and presentation by utilizing ASP and COM. The task of encapsulating the logic from the user's presentation has been made easier. Application separation is the key to productivity. By creating .NET components and class files, the user can maximize code re-use.

### Backward Compatibility

Organizations will be able to leverage their existing investments in COM and COM+. ASP.NET applications can interact with legacy COM objects.

## 1.8 Dataflow in ASP.NET



The data flow in this scenario is shown in the following illustration:

As shown in the figure 1.4, the sequence of events is as follows:

1. A client generates request for a protected resource.
2. IIS receives the request, and if the requestor is authenticated by IIS - or if IIS "Anonymous Access" is enabled, it is passed on to the ASP.NET application. Because the authentication mode in the ASP.NET application is set to "cookie", any IIS authentication is not used.
3. ASP.NET checks to see if a valid authentication cookie is attached to the request. If the identity's credentials have been confirmed previously, the request is tested for authorization. The authorization test is performed by ASP.NET and accomplished by comparing the credentials contained in the request's authorization cookie with the authorization settings in the application configuration files (config.web). If the user is authorized, access is granted to the protected resource - or the application may require an additional test of the credentials before authorizing access to the protected resource, depending on the design of the application.

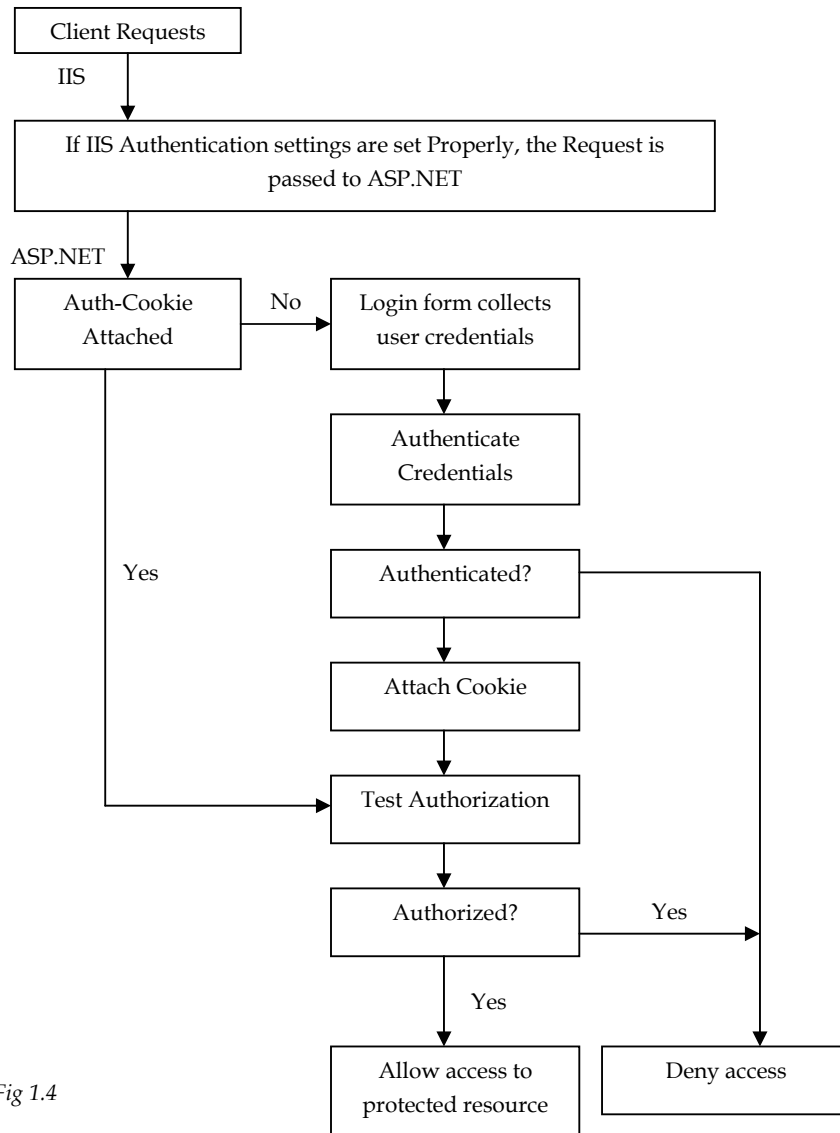


Fig 1.4

4. If there is no cookie attached to the request, ASP.NET redirects the request to a login page (the path of which resides in the application's configuration file) where the client user enters the required credentials (usually a name and password).
5. The application code checks the credentials to confirm their authenticity (usually in an event handler) and if authenticated, attaches a cookie ticket containing the credentials to the request. If authentication fails, the request is usually returned with an "Access Denied" message.
6. If the user is authenticated, ASP.NET checks authorization as in step 3, and can either allow access to the originally requested, protected resource or redirect the request to some other page, depending on the design of the application. It can otherwise, direct the request to a custom form of authorization where the credentials are tested for authorization to the protected resource. Usually if authorization fails, the request is returned with an "Access Denied" message.

## 1.9 A Simple ASP.NET Application

An ASP.NET application can be saved either in a Home directory or Virtual directory.



### Practice 1.1

The following example illustrates how an ASP.NET application can be saved.

```
<HTML>
<HEAD>
<TITLE>ASP Script</TITLE>
<script language="C#" runat="server">
  void Page_Load(Object s,EventArgs x)
  {
Response.Write("Welcome to Radiant");
  }
</script>
</HEAD>
</HTML>
```

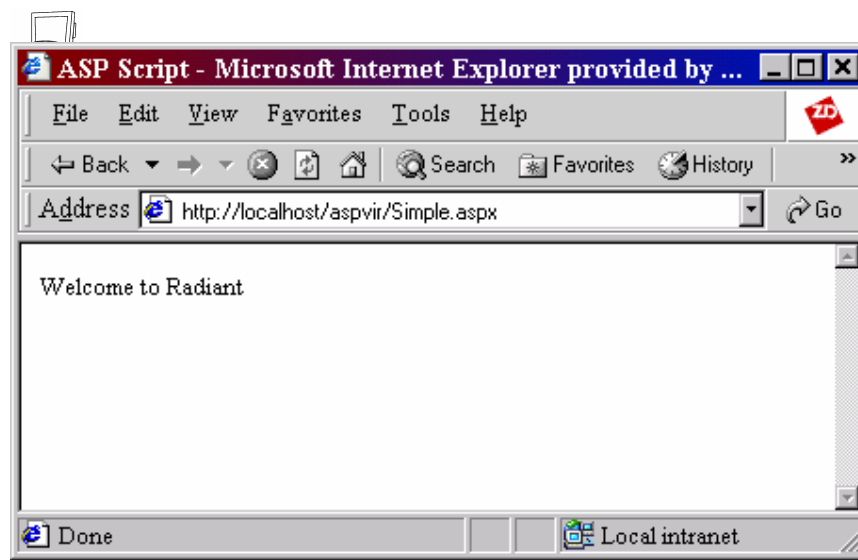
Save this file as Simple.aspx in the **Inetpub\wwwroot** directory, which is the **home directory**. (If no alteration has been made in the Internet Information Server's Service Manager, this will be the home directory for the default web site).

(or)

Place the Simple.aspx file in any desired directory, and create a **virtual directory** (see Creating a Virtual Directory) to map to the physical directory in which the Simple.aspx file has been stored.

Open Simple.aspx in the Web browser by specifying its URL as in the following:

`http://machine_name/ Simple.aspx`



As it can be seen from the above output, the application is saved in a virtual directory called "aspvir".

### 1.10 Short Summary

- Web server is a program that is used to serve content over the Internet using the Hypertext Markup Language (HTML)
- When the client asks for information such as a file, the Web server gets the file and sends it to the client
- Web servers are available for both Internet and Intranet related programs
- A Web page typically consists of an HTML file with associated files for graphics and scripts in a particular directory on a particular computer (and thus identifiable by a URL)
- A static web page is a page whose content is changed before being sent over the Internet to the user's computer. Such pages appear to the user to be static
- A dynamic web page is a page that contains animated GIFs, Java applets, ActiveX Controls, or DHTML
- Home directory is the root directory for a web site, where the content files are stored
- A virtual directory is not contained in the home directory
- ASP.NET is an entirely new framework from Microsoft for building Next Generation Web Applications.
- The major features of the ASP.NET are server control, session state, form validation, web services, user control, mobile control, data access, error handling and debugging, user authentication and server-side caching etc.
- The various advantages provided by the ASP.NET environment are compiled pages, more scripting charges, server controls, platform independence and separation of logic etc.
- There are different ways of securing ASP.NET applications, depending on the preferences of the designer
- An ASP.NET application can be saved either in the Home directory or Virtual directory

### 1.11 Brain Storm

1. What is a web server? What are the various Server-Side technologies that are available to deliver standard web pages.
2. What is the difference between a static and dynamic web page?
3. What is a virtual directory and in what way does it differ from a home directory?
4. What is ASP.NET? How does it use the .NET framework?
5. What are the features of ASP.NET?
6. What are the advantages of ASP.NET?



---

## HTML Server-Side Controls

---

### Objectives

In this lecture you will learn the following

- + Basic coding in ASP.NET
- + Learning various server-side controls

---

## Coverage Plan

---

Lecture 2
2.1 Snap Shot
2.2 Basics of coding in ASP.NET
2.3 Structure of code Blocks
2.4 HTML Server-side controls
2.5 Short Summary
2.6 Brain Strom

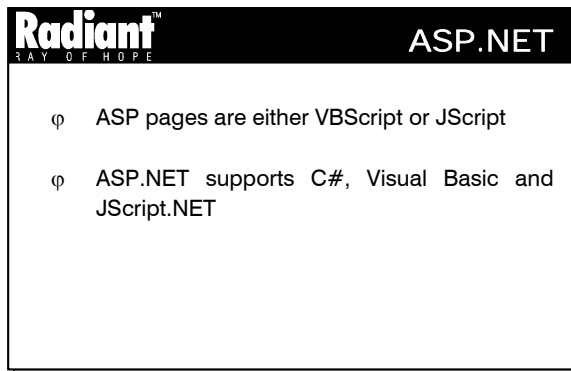
## 2.1 Snap Shot

This session is focussed on the basics of coding in ASP.NET, structure of code blocks and various HTML Server-Side Controls.

ASP executes components on the server and generates sections of the page that are returned to the user. This concept is extended in ASP.NET through server controls. The requirement to convert an HTML element into a server control is the addition of an extra attribute: **runat="server"**. Any HTML element in a page can be marked using this method and ASP.NET processes the elements on the server and generates output which is suitable to a specific client.

One of the most tedious tasks in creating interactive Web sites and applications is managing the values passed to the server from HTML form controls, and maintaining the values between page requests. Hence, one of the aims of ASP.NET is to simplify this programming task that involves no extra effort on the part of the programmer, and works well with any browser that supports basic HTML.

## 2.2 Basics of Coding in ASP.NET



Most of the existing ASP pages use either VBScript or Jscript. Since both of these are being interpreted languages, they are no longer supported by ASP.NET.

ASP.NET, currently offers built-in support for three languages: C#, Visual Basic, and JScript.NET. Component-based applications can be created using any of the Visual Studio .NET languages: Managed Extensions (produced using Visual C++ .NET), Visual Basic, and C#.

An ASP.NET page is restricted to code written in a SINGLE language. The default language is C#, but any other compliant language can be declared as the default for the page through a directive like:

```
<%@ Language = VB%>
```

**The language may also be declared within <script> blocks as shown below:**

```
<script language = VB>
```

If different languages are declared in separate script blocks on the same page, an error will be thrown.

“Render functions” are not supported in ASP.NET. It is possible to insert literal HTML within a procedure body in the earlier versions of ASP:

```
<% Sub SomeProcedure() %>
<H3> Render this line in bold text. </H3>
<% End Sub %>
```

**In an ASP.NET page, this can be written as:**

```
<script runat=server>
Sub SomeProcedure()
Response.Write("<H3> Render this line in bold text.</H3>")
End Sub
</script>
```

### Postback events

A Web page must be static to be viewed through the widest range of browsers. This means any interaction with the user (such as updating the display in response to a button click) must be resolved by calling the Web server and sending a new page.

The simplest way of implementing a request-response dialog is through the HTML form element. When a form is submitted, it invokes its target URL, sends the page and data from its data entry controls. A well-written ASP page submits the form to it, using hidden fields to cache the data needed to reconstruct the current state in the refreshed client display.

A postback event occurs on the client, but handled by the code in a copy of the page running on the server. For example, in an ordinary page a pushbutton can have an OnClick event, which is handled by the client-side code. In ASP.NET pushbutton can have an OnServerClick event, which causes a round-trip back to the server. When accessed by the server-side event handler, the values of the form's control properties reflect the data submitted by the client-side form, and any updates performed by the handler are reflected in the refreshed client display. Postback events are the key to the advanced features of DataGrid and DataList controls. They simplify development by hiding the difficulties of the stateless HTTP model.

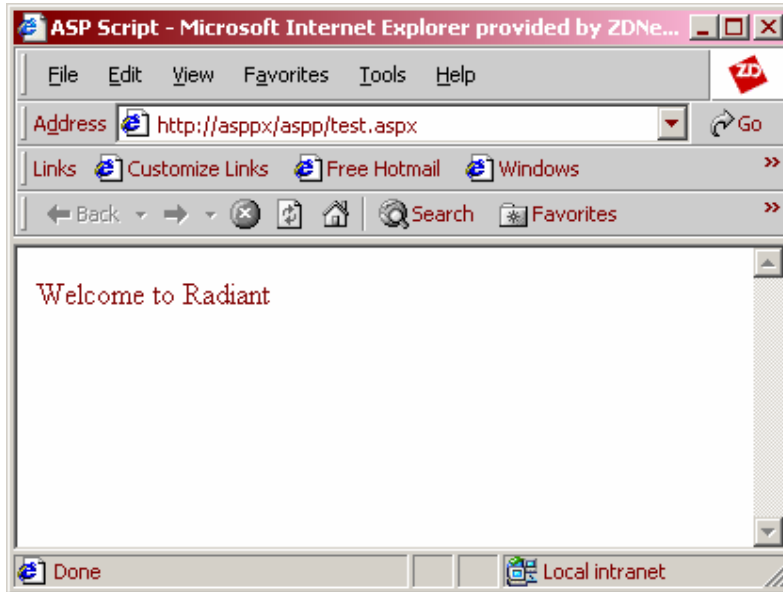


### Practice 2.1

The following code shows a simple ASP.NET application.

```
<HTML>
<HEAD>
<TITLE>ASP Script</TITLE>
<script language="C#" runat="server">
void Page_Load(Object sender,EventArgs e)
{
    Response.Write("Welcome to Radiant");
}
</script>
</HEAD>
<HTML>
```





In the above program, the **Response.Write** method is used in **Page\_Load** to display the message in the browser.

### 2.3 Structure of Code Blocks

In the previous versions of ASP, the **<script>** element or the script delimiters **<%...%>** were used to enclose script code in a page. In ASP.NET all functions and subroutines must be placed within the **<script>** section. The script delimiters **<%...%>** cannot be used.

#### Page Directives

Radiant™ RAY OF HOPE	ASP.NET
<ul style="list-style-type: none"> <li>⊘ Generally used to define the script language</li> <li>⊘ Can also set the transition state for the page, the character encoding and several other properties</li> <li>⊘ Possible to accept multiple directive statements in a page</li> </ul>	

In the previous versions of ASP, a single page can contain only one page directive. The directive is generally used to define the script language that is used to code. For example,

```
<%@Language="JScript"%>
```

The transition state for the page, the character encoding and several other properties can also be set by using this directive. In ASP.NET, it is possible to accept multiple directive statements in a page. The default directive has the name **Page**.

```
<%@Page Language="JScript"%>
```

The above directive supports the existing ASP attributes like **Transition="Required"**. The other directives supported are listed in Table 2.1.

Attribute	Values supported	Description
BUFFER	True or False	Specifies if response buffering is on.
CONTENTTYPE	Any content type string	HTTP content type for the output.
CODEPAGE	Valid codepage value	Locale code page of the file.
CULTURE	Valid COM+ culture string	Culture setting of the page.
RESPONSEENCODING	GetEncoding() values	Encoding of response content.
LCID	Valid LCID identifier	Locale identifier for the code.
DESCRIPTION	String description	Text description of the page.
ENABLESESSIONSTATE	True or False (read-only)	Session state requirements.
ERRORPAGE	Valid HTTP URL	URL for unhandled errors.
INHERITS	Page-derived COM+ class	"Code-behind" class for the page to inherit from.
LANGUAGE	VB, JScript, C# or {other}	Language used when compiling all <%...%> blocks in the page.
MAINTAINSTATE	True or False	Indicates whether viewstate should be maintained across page requests.
TRANSACTION	Not supported, Supported, Required or RequiresNow	Indicates the transaction semantics.
TRACE	True or False	Indicates whether tracing is enabled.
SRC	Source file name	'Code-behind' class to be compiled.
WARNINGS	True or False	Indicates whether compiler warnings should be treated as errors or ignored.

**Table 2.1**

In order to maintain backward compatibility, the directive statements in a page having no recognized directive name like Page, Register etc. will be considered as the **Page** directive statement. For example,

```
<%@Language="JScript"%>
```

is taken as

```
<%@Page Language="JScript"%>
```

### Web Forms Server Controls

Server controls are specifically designed to work with Web Forms. They differ from Windows controls since they work within the ASP.NET framework. Due to this, server controls feature unique design considerations. The different types of controls are:

- HTML Server Controls - HTML elements are exposed to the server. They expose an object model that maps very closely to the HTML elements that they render
- ASP.NET Web Controls - Controls having more built-in features than HTML server controls. ASP.NET server controls include controls like buttons, text boxes and also special-purpose controls like calendar. ASP.NET server controls does not necessarily reflect HTML syntax
- Validation Controls - Controls that incorporate logic to test the user input. A validation control is attached to an input control to test data entered by the user for that input control. Validation controls allows to check for a required field, to test against a specific value or pattern of characters, between ranges, and so on
- User Controls - Controls that are created by the user as Web Forms pages. Web Forms User Controls can be embedded in other Web Forms pages and hence menus, toolbars, and other reusable elements can be created easily

HTML controls offer Web developers the power of the Web Forms page framework while retaining the familiarity and easier use of HTML elements. These controls look exactly like HTML, except they have a `runat="server"` attribute/value pair in the opening tag of an HTML element. For example, the following HTML would not only create a text box on a Web page, but would also create an instance of the **HtmlInputText** server control:

```
<input id="myTextBox" type="text" runat="server">
```

**Note:** The `runat` and `id` properties are mandatory to all the `HtmlServerControls`.

Without `runat="server"`, this line of HTML would be parsed into a standard text box. This model allows developers to import an HTML page authored by a Web designer and program against selected elements on the page. In addition, once an element is converted to a control, a Microsoft .NET Framework class is created for the element and its attributes are exposed as properties on the HTML control.

To enable programmatic referencing of the control, developers can include a unique id attribute. In the above example, the `id="myTextBox"` defines this programmatic ID, allowing developers to manipulate the contents of this text box with events and other code that they write.

ASP.NET offers direct object model support for the most commonly used HTML elements. For those elements that are not directly supported, there is the `HtmlGenericControl`, which supports the `<span>`, `<div>`, `<body>`, and `<font>` elements, among others.

In addition, HTML controls offer the following features:

- A set of events for which handlers can be written in much the same way as it is done in a client-based form, except that the event is handled in the server code
- The ability to handle events in client script
- None of them emit ECMAScript to the client when responding to a request
- Automatic management of the values of the form's controls. If the form makes a round trip to the server, HTML controls are automatically populated with the values they had when the form was submitted to the server

- Interaction with validation controls helps to verify that a user has entered appropriate information into a control
- Databinding to a single field, property, method, or expression (simple data binding) or a collection (list databinding)
- Support for HTML 4.0 styles
- Pass-through of custom attributes. The required attributes can be added to HTML control. The Web Forms framework will read them and render them without any change in functionality. This allows the user to add browser-specific attributes to the controls
- All controls that post back events must be nested within an HtmlForm control. For example, the following input elements would be parsed correctly as HTML server controls:

```
<form runat="server" id="myForm">
  <input type="text" id="myTextBox" runat="server">
  <input type="submit" id="mySubmitButton"
    OnServerClick="SubmitBtn_Click" runat="server">
</form>
```

- All HTML controls must be well formed and should not overlap. Unless noted, elements must be closed, either with an ending slash within the tag, or with a closing tag. For example,

```
<span id="mySpan" runat="server" />
or
<span id="mySpan" runat="server">Span text to be displayed here</span>
```

## 2.4 HTML Server-Side Controls

The various controls explained under this section are:

- HtmlInputButton
- HtmlInputFile
- HtmlInputImage
- HtmlInputText
- HtmlForm
- HtmlSelect
- HtmlTable
- HtmlTableCell
- HtmlTableRow
- HtmlTextArea
- HtmlInputCheckBox
- HtmlInputHidden
- HtmlInputRadioButton
- HtmlAnchor
- HtmlGeneric
- HtmlImage

The hierarchy of the controls is given in the following figure.

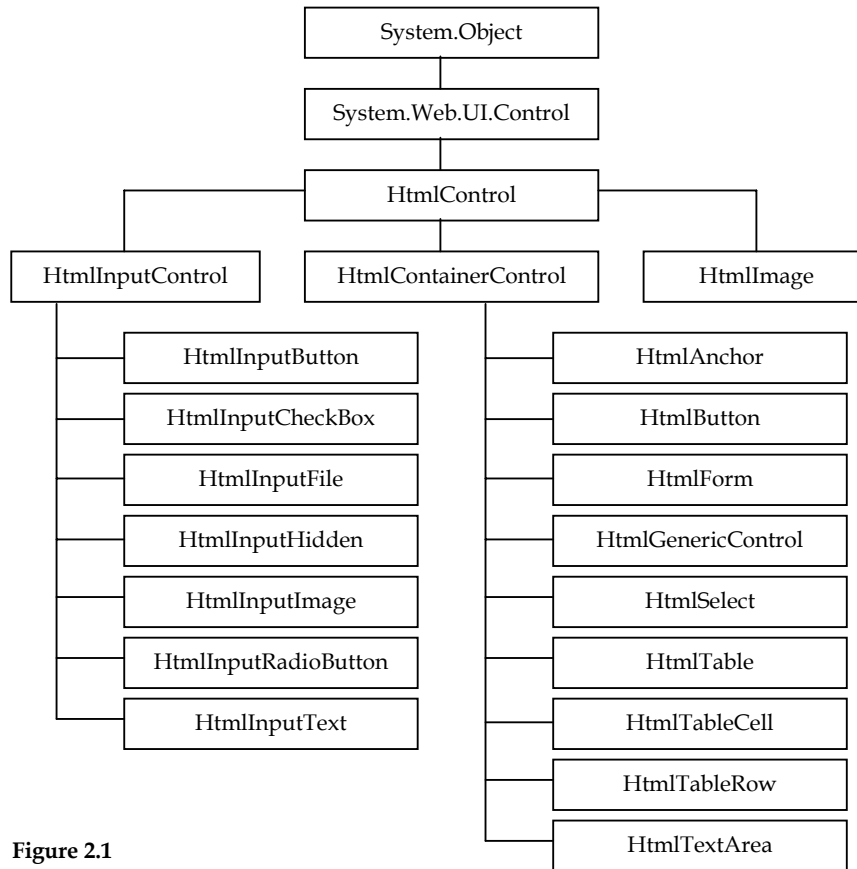


Figure 2.1

### HtmlInputButton Control

<b>Radiant™</b> <small>RAY OF HOPE</small>	<b>ASP.NET</b>
<ul style="list-style-type: none"> <li>☐ Allows to program against the HTML &lt;input type=button&gt;, &lt;input type=submit&gt;, and &lt;input type=reset&gt; elements</li> <li>☐ Processing is done in the server and a response is sent back to the requesting browser</li> <li>☐ Used in conjunction with the HtmlInputText and HtmlTextArea controls</li> </ul>	

When a user clicks on HTML Input Button control, input from the form that contains the embedded control is posted to the server. The processing is done in the server and a response is sent back to the requesting browser.

The HtmlInputButton is used in conjunction with the HtmlInputText and HtmlTextArea controls to create user input or authentication pages that can be processed on the server.

### Syntax



The above program displays a text box and a button using the input tag. When the button is clicked after entering the text in the textbox, the Button\_Click1 function is called and the text is displayed using span.

### HtmlInputFile Control

The HtmlInputFile control is used to design a page that allows uploading a file to a directory designated on the Web Server. The control allows to program against the HTML <input type=file> element. This control allows uploading of a binary or text file from a browser to the server. File upload is enabled in HTML.

### Syntax

```
<input
  type=file
  runat="server"
  id="programmaticID"
  accept="MIMEencodings"
  maxlength="maxfilepathlength"
  size="widthoffilepathtextbox"
  postedfile="uploadedfile"
>
```

- **type** is the file type (text, binary, etc)
- **postedfile** is the name of the file that is uploaded
- **maxlength** is the maximum length of the file path



### Practice 2.3

The following program illustrates how to upload a file to the Web server.

```
<html>
<head>

<script language="C#" runat="server">

void Button1_Click(object Source, EventArgs e)
{
    if (Text1.Value == "")
    {
        Span1.InnerHtml = "Please enter a file name";
        return;
    }

    if (File1.PostedFile != null)
    {
        try
        {
            File1.PostedFile.SaveAs("d:\\files\\"+Text1.Value);
            Span1.InnerHtml = "File uploaded successfully to
            <b>c:\\temp\\"+Text1.Value+"</b> on the web server";
        }
        catch (Exception exc)
        {
            Span1.InnerHtml = "Error saving file
            <b>c:\\temp\\"+Text1.Value+"</b><br>" + exc.ToString();
        }
    }
}
}
}
}
```

```

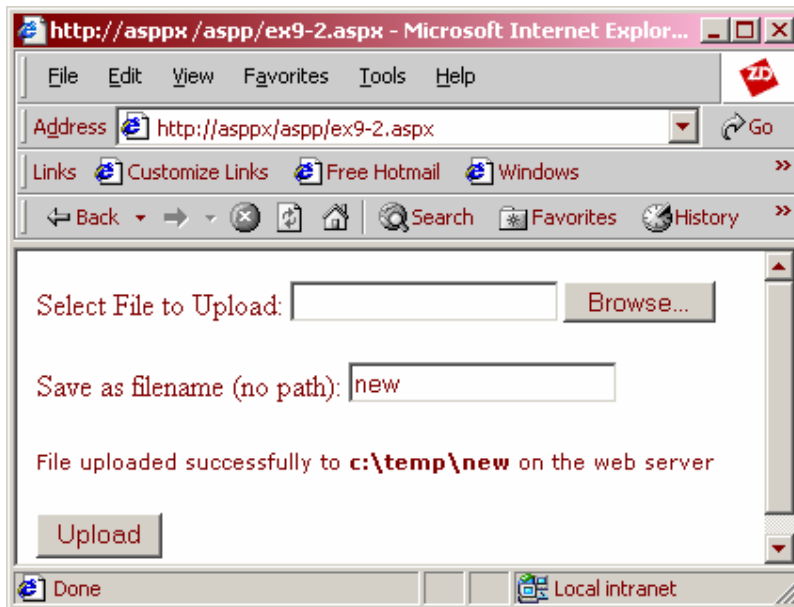
</script>

</head>
<body>

  <form enctype="multipart/form-data" runat="server">
    Select File to Upload: <input id="File1" type=file runat="server">
      <p> Save as filename (no path): <input id="Text1" type="text"
        runat="server">
      <p> <span id=Span1 style="font: 8pt verdana;" runat="server" />
      <p> <input type=button id="Button1" value="Upload"
        OnServerClick="Button1_Click" runat="server">
  </form>

</body>
</html>

```



The above program prompts the user to select a file to upload to the Web server. It requires a new filename and if it is not entered, an error message is displayed. The program also displays an error message if there is a problem in saving the file. When the file is uploaded successfully a message is displayed.

### HtmlInputImage Control

The HtmlInputImage control allows to program against the HTML `<input type=image>` element. This control is used in conjunction with the **HtmlInputText**, **HtmlTextArea**, and other controls to construct user input forms. Since this control is run on the server, it offers the same customizations for the button like HTML. This control offers greater support for earlier browsers than the HtmlButton control.

### Syntax

```
<input
```



```

    type=image
    runat="server"
    id="programmaticID"
    src="imagepath"
    align="imagealignment"
    alt="alttext"
    OnServerClick="OnServerClickhandler"
    width="borderwidthinpixels"
>

```

- **src** is the source path of the image file
- **align** is the alignment of the image in the browser
- **alt** is the alternate text that has to be displayed in case the image file is absent in the specified path



### Practice 2.4

The following program displays two images and responds to the mouse move and mouse click events.

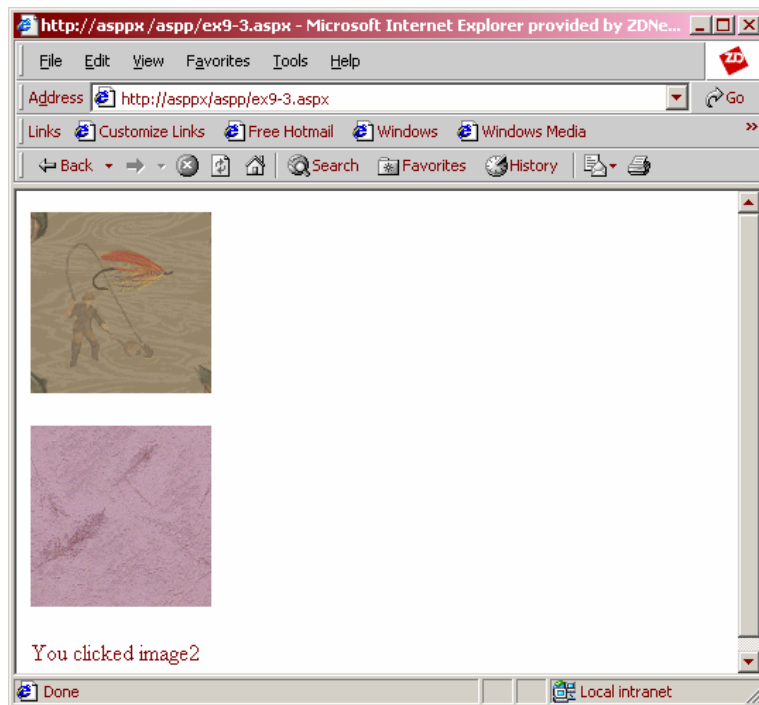
```

<head>
<script language="C#" runat="server">
    void Button1_Click(object Source, ImageClickEventArgs e)
    {
        Message.InnerHtml="You clicked image1";
    }
    void Button2_Click(object Source, ImageClickEventArgs e)
    {
        Message.InnerHtml="You clicked image2";
    }
</script>
</head>
<form runat="server">
    <p><input type=image id="InputImage1" src="d:/images/one.bmp"
        OnServerClick="Button1_Click" runat="server">
    <p><input type=image id="InputImage2" src="d:/images/one.bmp"
        onmouseover="this.src='d:/images/two.bmp';"
onmouseout="this.src='d:/images/one.bmp';" OnServerClick="Button2_Click"
runat="server">
    <p><span id="Message" runat="server"/>
</form>

```

In the above program, two images are loaded using the Html-input control. When the mouse cursor is moved over the second image, the image changes to another image using the **onmouseover**. When the mouse is moved out of the second image, the original image is loaded using **onmouseout**. When either of the images is clicked, a message is displayed indicating the image that is clicked. This is handled by the OnServerClick event.





### HtmlInputText Control

**Radiant**  
RAY OF HOPE

**ASP.NET**

- ⌚ Allows to run server code against the HTML  
<input type=text> and <input type=password>
- ⌚ Can be used to enter user names and passwords

This control can be used in conjunction with the **HtmlInputButton**, **HtmlInputImage**, or **HtmlButton** control to process user input on the server. Values can be assigned and retrieved to the properties **HtmlInputText** **MaxLength**, **Size**, and **Value**.

**Note :** This control does not require a closing tag.

### Syntax

```
<input
  type=text | password
  runat="server"
  id="programmaticID"
  maxlength="max#ofcharacters"
  size="widthoftextbox"
  value="defaulttextboxcontents"
```

&gt;

- **type** specifies whether the input characters are for an ordinary field or for a password field. If set to password, then an asterisk (\*) is displayed instead of the character which is typed to preserve secrecy
- **maxlength** attribute specifies the maximum length of the text that can be typed in the text box



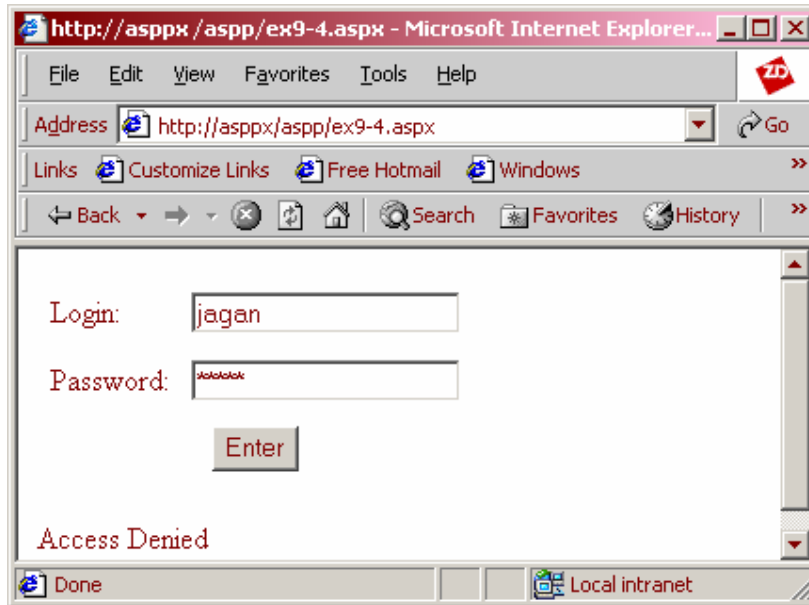
### Practice 2.5

The following program checks the authentication of the user.

```
<head>
<script language="C#" runat="server">
void SubmitBtn_Click(object Source, EventArgs e)
{
    if (Password.Value == "ASP.NET")
        Message.InnerHtml="Access Granted";
    else
        Message.InnerHtml="Access Denied";
}
</script>
</head>

<body>
    <form runat="server">
        <table cellpadding=5>
            <tr>
                <td>Login: </td>
                <td><input id="Name" runat="server"></td>
            </tr>
            <tr>
                <td>Password: </td>
                <td><input id="Password" type=password runat="server"></td>
            </tr>
            <tr>
                <td colspan=2 align="center"><input type=submit value="Enter"
                    OnServerClick="SubmitBtn_Click" runat="server"> </td>
            </tr>
        </table>
        <p><span id="Message" runat="server"/>
    </form>
</body>
```





In the above program, two text boxes are displayed to enter the user name and password. If the password entered is ASP.NET, then the message **Access Granted** is displayed, otherwise the message **Access Denied** is displayed.

### HtmlForm Control

The HtmlForm control allows to program against the HTML <form> element. All Web Form controls, whether HTML, Web, pagelet or custom, must be nested between well-formed opening and closing tags of the HtmlForm control in order to take advantage of the page framework's postback services.

**Note:** More than one HtmlForm control cannot be included on a single Web Forms page.

By default, the method attribute of the HtmlForm control is set to POST, and the action attribute is set to the URL of the source page. These attributes can be customized to suit the needs of an individual. However, this can break the built-in view state and postback services provided by the Web Forms Page Framework.

### Syntax

```
<form
  runat="server"
  id="programmaticID"
  method=POST | GET
  action="srcpageURL"
>
```

Other controls, input forms, and so on.  
</form>

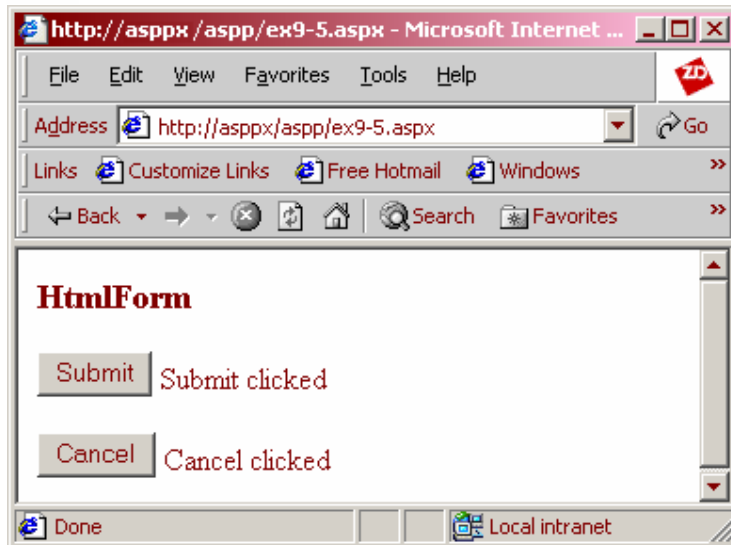
- **method** is the way in which the form is displayed
- **action** is the URL source



## Practice 2.6

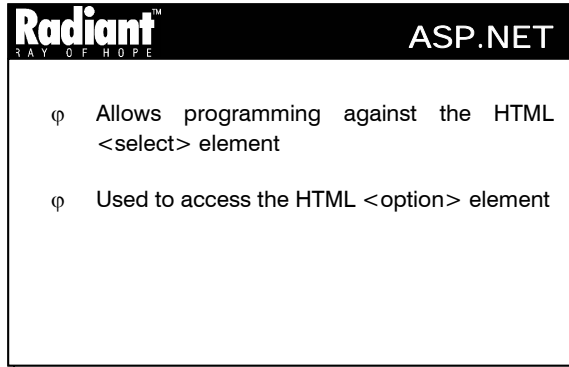
The following program illustrates the use of forms.

```
<head>
<script language="C#" runat="server">
void Button1_OnClick(object Source, EventArgs e)
{
    Span1.InnerHtml="Submit clicked";
}
void Button2_OnClick(object Source, EventArgs e)
{
    Span2.InnerHtml="Cancel clicked";
}
</script>
</head>
<body>
<div class="header"><h3>HtmlForm</h3></div>
<form id=ServerForm runat="server">
    <p><button id=Button1 runat="server" OnServerClick="Button1_OnClick">
Submit</button>
        <span id="Span1" runat="server" />
        <p><button id=Button2 runat="server" OnServerClick="Button2_OnClick">
Cancel</button>
        <span id=Span2 runat="server" />
</form>
</body>
```



The above program displays two buttons in a form that has the id **ServerForm**. The form has two buttons namely Submit and Cancel. When the buttons are clicked, the appropriate messages appear on the right side of the button.

### HtmlSelect Control



### Syntax

```
<select
  runat="server"
  id="programmaticID"
  OnServerChange="onserverchangehandler"
  DataSource="databindingsource"
  DataTextField="fieldtodatabindoptionttext"
  DataValueField="fieldtodatabindoptionvalue"
  MultipleItems="collectionofoptionelements"
  SelectedIndex="indexofcurrentlyselecteditem"
  Size="#ofvisibleitems"
  Value="currentitemvalue"
>
<option>value1</option>
<option>value2</option>
</select>
```

- **OnServerChange** is the handler for the select control
- **DataSource** is the source from which the items for the select list are to be taken
- **DataTextField** specifies the text that has to be displayed in the select list
- **DataValueField** specifies the value corresponding to the **DataTextField**
- **MultipleItems** option specifies whether multiple items can be selected
- **SelectedIndex** is the index value of the selected item



### Practice 2.7

The following program creates a selectable drop-down list and allows to choose an option.

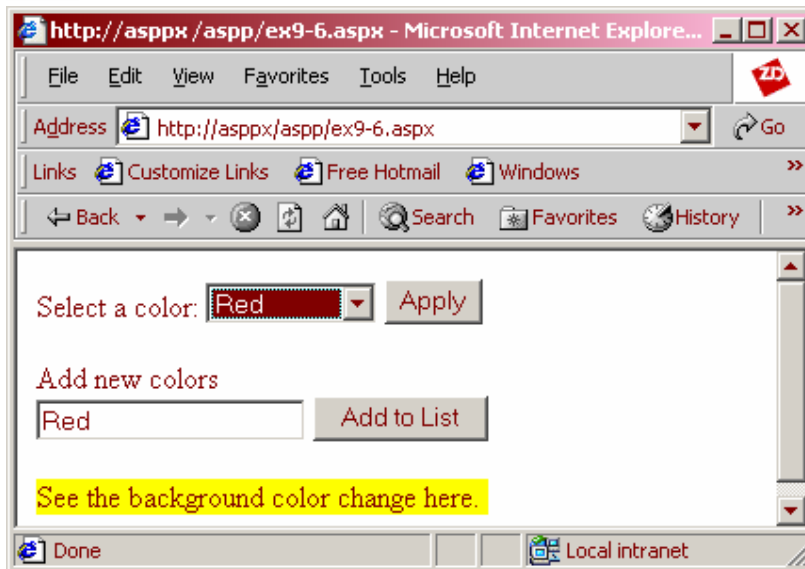
```
<head>
<script language="C#" runat="server">

void Apply_Click(object Source, EventArgs e)
{
    Message.Style["background-color"]=ColorSelect.Value;
```

```
}
void AddToList_Click(object Source, EventArgs e)
{
    ColorSelect.Items.Add(Text1.Value);
}

</script>
</head>

<body>
    <form runat="server">
        <p>Select a color:</p>
        <select id="ColorSelect" runat="server">
            <option>SkyBlue</option>
            <option>LightGreen</option>
            <option>Blue</option>
            <option>Yellow</option>
        </select>
        <input type="button" runat="server" Value="Apply"
            OnServerClick="Apply_Click">
        <p>Add new colors<br>
        <input type="text" id="Text1" runat="server">
        <input type="button" runat="server" Value="Add to List"
            OnServerClick="AddToList_Click">
        <p><span id="Message" runat="server"> See the background color change here.
        </span>
    </form>
</body>
```



The above program displays a listbox of colors. On selecting an option from the list and clicking the **Apply** button, the color of the text at the bottom of the page is changed. If a new color is to be added to the list, the name is typed in the text box and the **Add to List** button is clicked.

### HtmlTable Control

The **HtmlTable** control allows to program against the HTML <table> element. A table control can be dynamically bound to add rows and cells to the table using the methods provided by the **HtmlTableRowCollection** and **HtmlTableCellCollection**.

#### Syntax

```
<table runat="server"
    id="accessID"
    align=left | center | right
    bgcolor="bgcolor"
    border="borderwidthinpixels"
    bordercolor="bordercolor"
    cellpadding="spacingwithincellsinpixels"
    cellspacing="spacingbetweenincellsinpixels"
    height="tableheight"
    rows="collectionofrows"
    width="tablewidth"
>
</table>
```

### HtmlTableCell Control

The **HtmlTableCell** control allows programming against the HTML <td> and <th> elements. Cell can be added dynamically to an **HtmlTableRow** control, in response to control events or by binding an **HtmlTable** control to the entries in a data source.

#### Syntax

```
<td or th
    runat="server"
    id="programmaticID"
    align="alignmentofcontentincell"
    bgcolor="bgcolor"
    bordercolor="bordercolor"
    colspan="#ofcolscellspans"
    height="tableheight"
    nowrap="True | False"
    rowspan="#ofrowscellspans"
    valign="verticalalignmentofcellcontent"
    width="cellwidth"
>
CellContent
</td or /th>
```

### HtmlTableRow Control



Creates a table row control. The **HtmlTableRow** control allows you to program against the HTML <tr> element. Rows can be dynamically added to an **HtmlTable** control, whether in response to control events or through binding an **HtmlTable** control to the entries in a data source.

### Syntax

```
<tr runat="server"
    id="accessID"
    align="tablecontentalignment"
    bgcolor="tablebgcolor"
    bordercolor="bordercolor"
    height="tableheight"
    cells="collectionoftablecells"
    valign="verticalalignmentofrowcontent"
>
    <td>cellcontent</td>
</tr>
```



### Practice 2.8

The following program illustrates the use of table, tablecell and table row.

```
<head>
<script language="C#" runat="server">

void Page_Load(Object sender, EventArgs e)
{
    int row=0;
    int numRows=int.FromString(Select1.Value);
    int numcells=int.FromString(Select2.Value);
    for (int j=0; j<numrows; j++)
    {
        HtmlTableRow r=new HtmlTableRow();
        if (row%2== 1)
            r.BgColor="Gainsboro";
        row++;
        for (int i=0; i<numcells; i++)
        {
            HtmlTableCell c=new HtmlTableCell();
            c.Controls.Add(new LiteralControl(j.ToString() + ", " +
                i.ToString()));
            r.Cells.Add(c);
        }
        Table1.Rows.Add(r);
    }
}
</script>
</head>
```

```

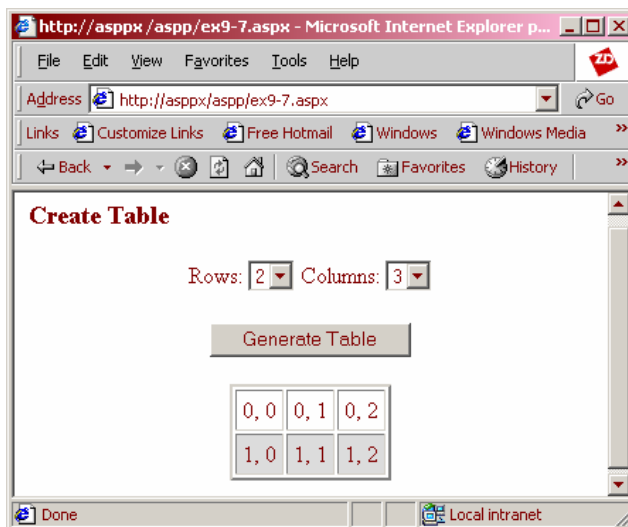
<body>
<div class="header"><h3>Create Table</h3></div>
<div align="center">
<form runat="server">
<table>
  <tr>
    <td>Rows:</td>
    <td>
      <select id="Select1" runat="server">
        <option Value="2">2</option>
        <option Value="3">3</option>
        <option Value="4">4</option>
      </select></td>

    <td>Columns:</td>
    <td>
      <select id="Select2" runat="server">
        <option Value="2">2</option>
        <option Value="3">3</option>
        <option Value="4">4</option>
      </select></td></tr>
  </table>
  <p><input type="submit" runat="server" value="Generate Table" > </form>
  <p><table id="Table1" runat="server" CellPadding=4 Border=2 /></p>


</div>
</body>
</html>

```

The above example uses the Add function to add cells and rows to a table dynamically. The number of rows and columns of the table are selected from the two drop down list boxes, one for the number of rows of the table and the other for the number of columns.



## HtmlTextArea Control



**Radiant™**  
RAY OF HOPE

**ASP.NET**

- ⊕ Allows programming against the HTML <textarea> elements
- ⊕ Can be used to gather feedback from the users

### Syntax

```
<textarea
  runat="server"
  id="programmaticID"
  cols="numberofcolsintextarea"
  name="namepassedtobrowser"
  rows="numberofrowsintextarea"
  OnServerChange="onserverchangehandler"
>
textareacontent
</textarea>
```

- **cols** indicates the width of the control
- **rows** indicates the height of the control
- **OnServerChange** is the handler for the control



### Practice 2.9

The following program displays a text area.

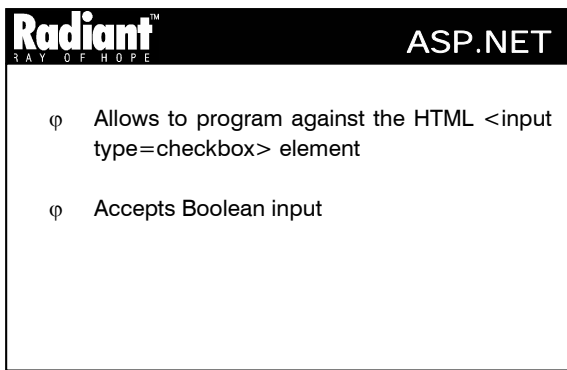
```
<head>
<script language="C#" runat="server">
void SubmitBtn_Click(Object sender, EventArgs e)
{
  Message.InnerHtml= TextAreal.Value;
}
</script>
</head>
<H3>Customize the message in your e-card</H3>
<body>
  <form runat="server">
    <p>Your message</p>
    <textarea id="TextAreal" runat="server" cols=40 rows=4 />
    <p><input type=submit value="Send Card" runat="server"
OnServerClick="SubmitBtn_Click">
    <p><span id="Message" runat="server" />
  </form>
</body>
```





In the above program, a text area is displayed along with a button. On clicking the button, after entering text in the text area, the typed text is displayed below the button.

### HtmlInputCheckBox Control



When checked, its **Checked** property returns true. **HtmlInputCheckBox** is typically used with other user-input controls, such as the **HtmlInputButton** control, to determine whether a check box is selected. This is done by evaluating the **Checked** property on this control.

### Syntax

```
<input type="checkbox"
  runat="server"
  id="accessID"
  checked
  name="radiobuttongroup"
  >
```

The **HtmlCheckBox** control does not have an event that notifies the server when a user has selected the control. Checkbox can be used with one of the button controls – such as the **HtmlInputButton**,

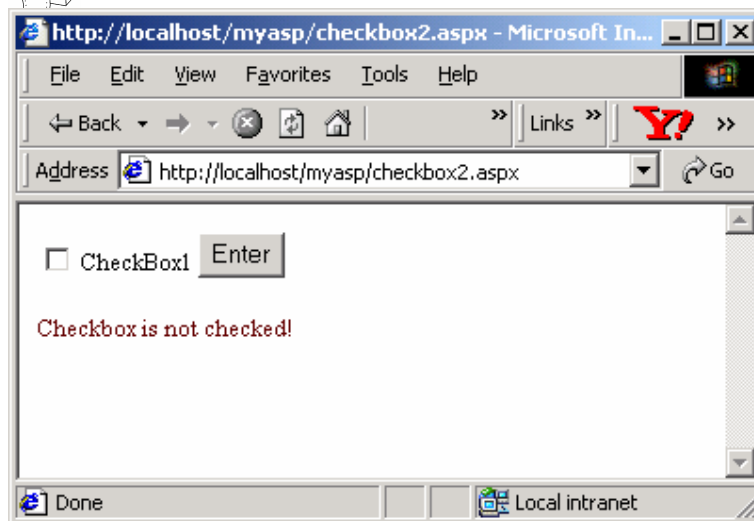


## Practice 2.10

HtmlInputImage, or HtmlButton – to cause a postback that will allow to program the **HtmlCheckBox** control itself.

The following example illustrates the use of HtmlInputCheckBox control. When a user clicks the input button included on the form, the button's click event handler evaluates the value of the **HtmlInputCheckBox** control's **Checked** property, then displays a message in a span control that depends on the value of the property.

```
<head>
<script language="C#" runat="server">
void Button1_Click(object Source, EventArgs e) {
    if (Check1.Checked== true) {
        Message.InnerHtml="Checkbox is checked!";
    }
    else {
        Message.InnerHtml="Checkbox is not checked!";
    }
}
</script>
</head>
<form runat="server">
    <input id="Check1" type="checkbox" runat="server"> CheckBox1
    <input type="button" id="Button1" value="Enter"
        OnServerClick="Button1_Click" runat="server">
<p><span id="Message" style="color:#600" runat="server" />
</form>
```



In the above program a check box and a button are used. In the OnServerClick event of the button, we check, whether the checkbox is checked or not. It also prints whether the checkbox is checked or unchecked.

### HtmlInputRadioButton

The **HtmlInputRadioButton** control creates a single radio button input field. By default, each individual radio button can be selected.

#### Syntax

```
<input type="radio"
      runat="server"
      id="controlID"
      checked
      name="radiobuttongroup"
>
```

This control does not require an opening and closing tag. The **HtmlRadioButton** control does not have an event that notifies the server when a user has selected one of the buttons.

A set of radio buttons can be grouped together in cases where only one button from the set of options needs to be selected. This is done by setting a common value for the **Name** attribute on each radio button in the group. For example, the following block of code creates a group of radio buttons in which only one radio button may be selected.

```
<input type=radio name="optgroup1">Red
<input type=radio name="optgroup1">White
<input type=radio name="optgroup1">Blue
```

However, the selected state must be tested on the individual radio buttons. **HtmlInputRadioButton** allows to program against the HTML `<input type=radio>` element.

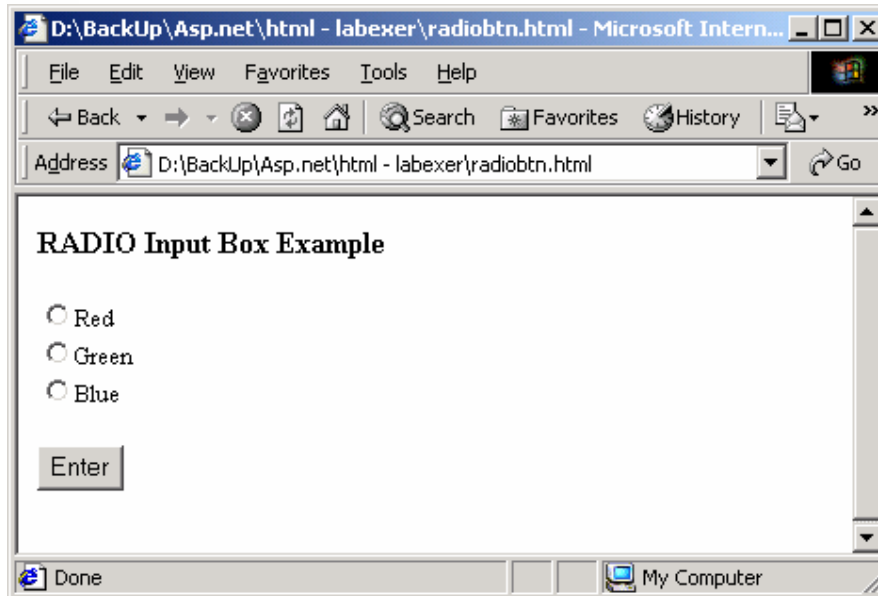


### Practice 2.11

The sample given below illustrates the use of `HtmlInputRadioButton` control.

```
<form runat="server">
  <h3>RADIO Input Box Example</h3>
  <input type="radio" id="Red" name="Mode" runat="server" />Red<br>
<input type="radio" id="Green" name="Mode" runat="server" />Green<br>
  <input type="radio" id="Blue" name="Mode" runat="server" />Blue
  <p><input type=button id="Button1" value="Enter"
      OnServerClick="Button1_Click" runat="server">
  <p><span id="Message" runat="server" />
</form>
<script language="C#" runat="server">
void Button1_Click(object Source, EventArgs e) {
    if (Red.Checked== true)
        Message.InnerHtml="Red is checked";
    else if (Green.Checked== true)
        Message.InnerHtml="Green is checked";
    else if (Blue.Checked==true)
        Message.InnerHtml="Blue is checked";
}
```

```
}
</script>
```



To respond to user selection of radio button controls, a set of radio button controls are declared. A button control causes postback when clicked by the user. A span control is used here to render the postback message. Event-handling code is included to process the information gathered from the radio button group.

### HtmlAnchor Control

**Radiant**  
RAY OF HOPE

**ASP.NET**

- φ Used to navigate from the client page to another page
- φ Allows to program against the HTML <a> element
- φ Used to dynamically modify the attributes and properties of the <a> element

The **HtmlAnchor** control is used to navigate from the client page to another page, or to another location within the same page.

This control can be generated by using a data source. Control events can be used dynamically to generate this control.

### Syntax

```

<a runat="server"
  id="accessID"
  href="linkurl"
  name="bookmarkname"
  OnServerClick="OnServerClickhandler"
  target="linkedcontentframeorwindow"
  title="titledisplayedbybrowser"
>
  linktext
</a>

```

Target values must begin with a letter, except when mentioning the following special values that begin with an underscore: `_blank`, `_self`, `_parent`, and `_top`. This control requires an opening and closing tag.

**Note:** Remember to embed the **HtmlAnchor** control inside the opening and closing tags of an **HtmlForm** control.

The hyperlinks can be generated on a Web Forms page by binding the **HtmlAnchor** control to a data source.



### Practice 2.12

The following example shows how to include an **HtmlAnchor** control and to bind the address of the links (URLs), as well as to link the text in order to display for the controls.

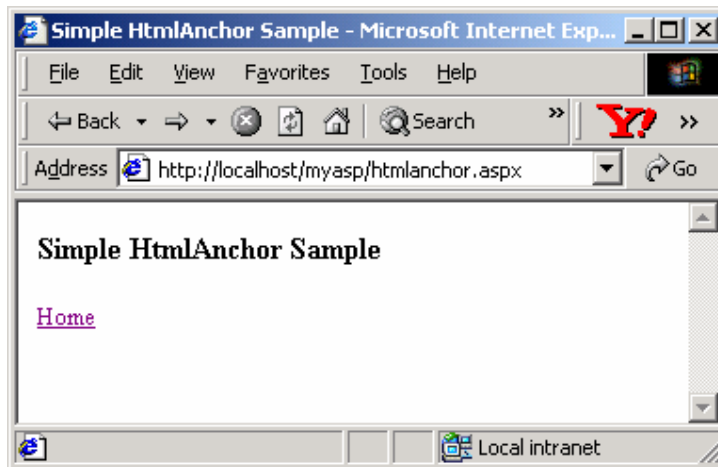
```

<html>
<head>
<title>Simple HtmlAnchor Sample</title>
<script language="C#" runat=server>
void Page_Load(Object sender, EventArgs e)
{
  anchor1.HRef="/";
}
</script>
</head>
<body>
  <h3>Simple HtmlAnchor Sample</h3>
  <form runat=server>
    <p><a id=anchor1 runat="server">Home</a>
  </form>
</body>
</html>

```







In this example the anchor control is used to load another page. The HRef refers to Index.html and when the anchor is clicked, the index.html page is loaded in the browser.

### Generating the HRef Attribute Dynamically

Declare an **HtmlAnchor** control in the BODY of an HTML document, as shown below.

```
<form runat="server">
  <p><a id=anchor1 runat="server">Home</a>
</form>
```

An event-handling code that assigns a URL to the HtmlControl HRef property has to be written. The following code generates a URL that will let the user navigate the site's home page when the **Page\_Load** event occurs.

```
<script language="C#" runat="server">
  void Page_Load(Object sender, EventArgs e) {
    anchor1.HRef="/";
  }
</script>
```

### HtmlGenericControl

The **HtmlGenericControl** provides a server control implementation for all other HTML server control tags that are not directly represented by a specific HTML server control. These include the `<span>`, `<body>`, `<div>`, and `<font>` elements, among others.

### Syntax

```
<span | body | div | font | others
  runat="server"
  id="accessID"
>
element content here
</span | body | div | font | others>
```

An **HtmlGenericControl** is created on the server in response to tags that include the `runat="server"` attribute in elements that do not map directly to a specific HTML control. The control maps the name of the tag of the particular element to be used as an HTML control to the page framework through the

TagName property. This control inherits the functionality from the HtmlContainerControl class, which allows to dynamically change the inner content of HTML control tags.

The following samples illustrate the use of **HtmlGenericControl**.

Assign an **id** attribute to the form which enables the programmatic access to the control.

```
<body id=Body runat=server>
```

Declare an **HtmlInputButton** control and assign it an **id** attribute to allow programmatic access to the control. Here, we declare an **OnServerClick** designate to instruct the framework about the function when the button is clicked as given below.

```
<input type="submit" runat="server" Value="Apply"
  OnServerClick="SubmitBtn_Click">
```

In the script declaration block at the head of the page, create event-handling code that manipulates the value entered by the user in the option to display the color in the background

```
void SubmitBtn_Click(object Source, EventArgs e) {
  Body.Attributes["bgcolor"]=ColorSelect.Value;
}
```

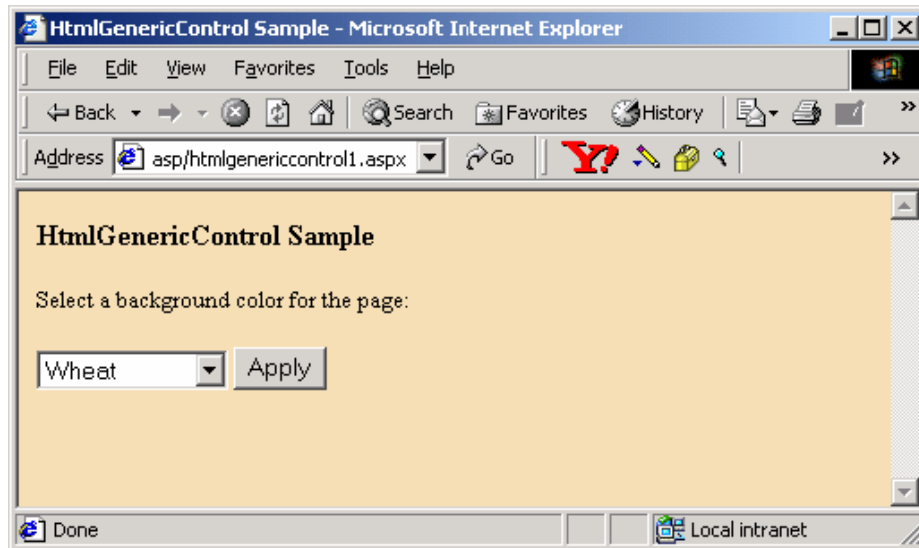


### Practice 2.13

The following example shows how an **HtmlGenericControl** is used to modify a background color of the page. The sample also demonstrates how to use the AttributeCollection class to programmatically access the attributes that can be declared on any HTML control, in this case, the HTML BODY element.

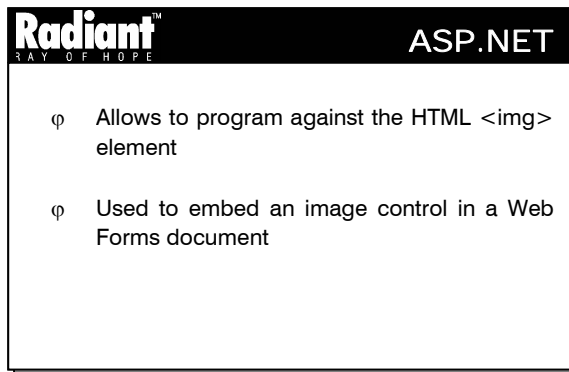
```
<html>
<head>
<script language="C#" runat="server">
void SubmitBtn_Click(object Source, EventArgs e) {
  Body.Attributes["bgcolor"]=ColorSelect.Value;
}
</script>
</head>
<body id=Body runat="server">
<div class="header"><h3>HtmlGenericControl Sample</h3></div>
<form runat="server">
  <p>Select a background color for the page: <p>
  <select id="ColorSelect" runat="server">
    <option>White</option>
    <option>Wheat</option>
    <option>Red</option>
    <option>Green</option>
  </select>
  <input type="submit" runat="server"
    Value="Apply" OnServerClick="SubmitBtn_Click">
</form>
</body>
</html>
```





The above program uses an `HtmlGenericControl` to display a list box and a button. When a color is selected and the `Apply` button is clicked, the background color of the form changes accordingly.

### HtmlImage Control



The `HtmlImage` control allows to program against the HTML `<img>` element. This control allows to dynamically set and retrieve image attributes that include the `src`, `width`, `height`, `border`, `alt`, and `align` attributes.

This control embeds an image control in a Web Forms document.

#### Syntax

```

```

## Working with HtmlImage

The **HtmlImage** server control renders an image file specified by its **Src** property. The **HtmlImage** control allows to program against the HTML `<img>` element, enabling developers to dynamically set and retrieve image attributes, including the image file's **src**, **width**, **height**, **border**, **alt**, and **align** attributes.

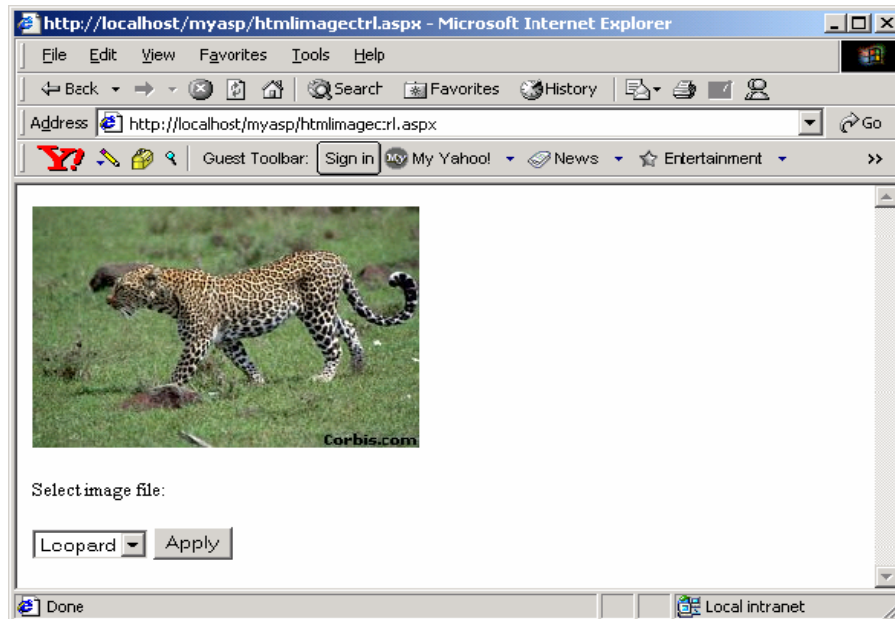


### Practice 2.14

The following example shows how to dynamically change a displayed image in a Web Forms page based on the choice of the user.

```
<html>
<head>
<script language="C#" runat="server">
void SubmitBtn_Click(object Sender, EventArgs e) {
    Image1.Src="d:/myprog/asp.net/Images/" + Select1.Value;
}
</script>
</head>
<body>
<form runat="server">

    <p>Select image file: </p>
    <select id="Select1" runat="server">
        <option Value="Bobcat.jpg">Bobcat</option>
        <option Value="Cheetah.jpg">Cheetah</option>
        <option Value="Leopard.jpg">Leopard</option>
        <option Value="Lynx.jpg">Lynx</option>
        <option Value="Tiger.jpg"> Tiger</option>
    </select>
    <input type="submit" runat="server" Value="Apply"
        OnServerClick="SubmitBtn_Click">
</form>
</body>
</html>
```



This example shows how to change a displayed image based on the choice of the user. The example uses the click event of the `HtmlInputButton` control to trigger a handler that specifies the path of the image file to be displayed, in this case, from the application's **images** directory. Notice that the `SubmitBtn_Click` event handler is simple, specifying the `d:/../images` directory as the source path for the images to be displayed. An `HtmlSelect` control provides the list of option for the user. The selected value in the `HtmlSelect` control in the Web Forms page determines which image to display from the absolute path specified.

## 2.5 Short Summary

- In ASP.NET, all the functions and subroutines must be placed within the `<script>` section
- Server controls are specifically designed to work with Web Forms
- The `HtmlInputButton` is used to display a command button
- `HtmlInputText` control is used to display a text box
- Tables can be dynamically bound to add rows and cells to the table using the methods provided by the `HtmlTableRowCollection` and `HtmlTableCellCollection`
- `HtmlAnchor` is used to navigate from the client page to another page, or to another location within the same page
- The `HtmlGenericControl` provides implementation for other HTML server control tags not directly represented by a specific HTML server control
- The `HtmlImage` server control renders an image file specified by the `SRC` property

## 2.6 Brain Storm

1. What are the languages supported by ASP.NET?
2. What is the significance of runat = "server"?
3. How can the text box be made to accept passwords?

❧

---

## ASP.NET Web Controls - I

---

### Objectives

In this lecture you will learn the following

- + Knowing about Web Controls
- + Learning Web Controls and its hierarchy

---

## Coverage Plan

---

<b>Lecture 3</b>
3.1 Snap Shot
3.2 Web Controls
3.3 Web Controls Hierarchy
3.4 Short Summary
3.5 Brain Storm

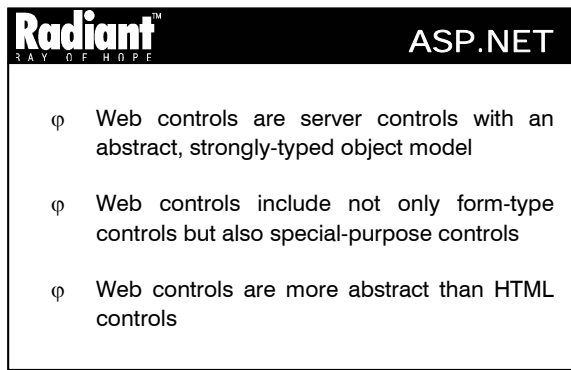


### 3.1 Snap Shot

This session is designed to enlighten the readers about web controls, its properties and events. It throws light on the common properties shared by all the web controls. This session enables the learners to know about the properties and events of each control with an example in detail.

Web controls are Server-Side controls, which can be used in the same way as HTML controls. The only difference is that they must have the **runat = "server"** attribute set. This attribute makes the control available for server-side programming. Each ASP.NET Server Control is capable of exposing an object model containing properties, methods and events. This object model can be utilized by the ASP.NET developers to modify and interact with the Web page.

### 3.2 Web Controls



Web controls are nothing but the server controls with an abstract, strongly-typed object model. Web controls include not only form-type controls such as buttons and text boxes, but also special-purpose controls such as a calendar. Web controls are more abstract than HTML controls. In web controls, the object model does not necessarily reflect the HTML syntax. There are some common properties shared by all the Web controls which are as follows.

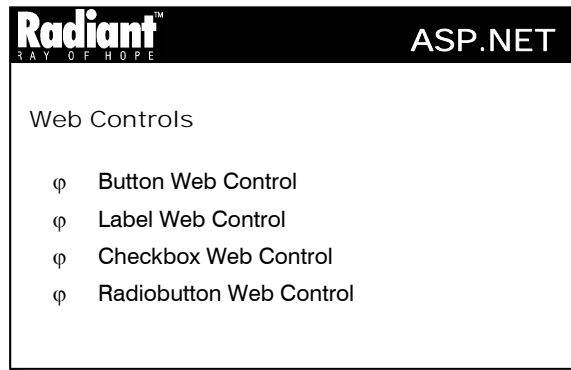
- **AccessKey** - This is the control's keyboard accelerator key. It specifies a single letter or number that the user should use while pressing ALT. For example, if the user wants to press ALT+K to access the control, "K" should be specified. Access keys are not supported on all browsers
- **Attributes** - Attributes denote the complete set for the control's persistent format. This property is only used while programming and cannot be set while declaring the control
- **BackColor** - This property indicates the color behind the text of the control
- **BorderWidth** - This indicates the size of the control's border in pixels
- **BorderStyle** - This indicates the border style of the control in pixels
- **CSS Class** - This class can be assigned to the control
- **CSS Style** - This is a collection of text attributes that will be rendered as a CSS style attribute on the outer tag of the control. This property is only used while programming and cannot be set while declaring the control
- **Enabled** - This property enables the control if set to true (the default) or disables when it is set to false

- **ForeColor** - ForeColor is the text color of the control. This property may not work in some of the earlier version browsers
- **TabIndex** - This property sets the order of the control when the user tabs between controls. If it is not set, the control's index is zero. Controls with the same tab index will be navigated in the order in which they are declared in the content
- **ToolTip** - ToolTip is indicated in the form of the text that appears when the mouse is placed on the control. ToolTip does not work in all browsers

### 3.3 Web Controls Hierarchy

All Web controls (except **Repeater**) derive directly or indirectly from the base class **System.Web.UI.WebControls.WebControl**. The controls shown on the left in the above figure map to HTML elements. The controls on the right provide data binding support. The controls in the middle are for validating form input. Additionally, in the center, are controls that provide rich functionality, such as the **Calendar** and the **AdRotator** controls.

It is possible to develop a custom Web control by extending an existing Web control, by combining existing Web controls, or by creating a control that derives from the base class **System.Web.UI.WebControls.WebControl**.



The following figure shows the Web Controls Hierarchy in the **System.Web.UI.WebControls** namespace.

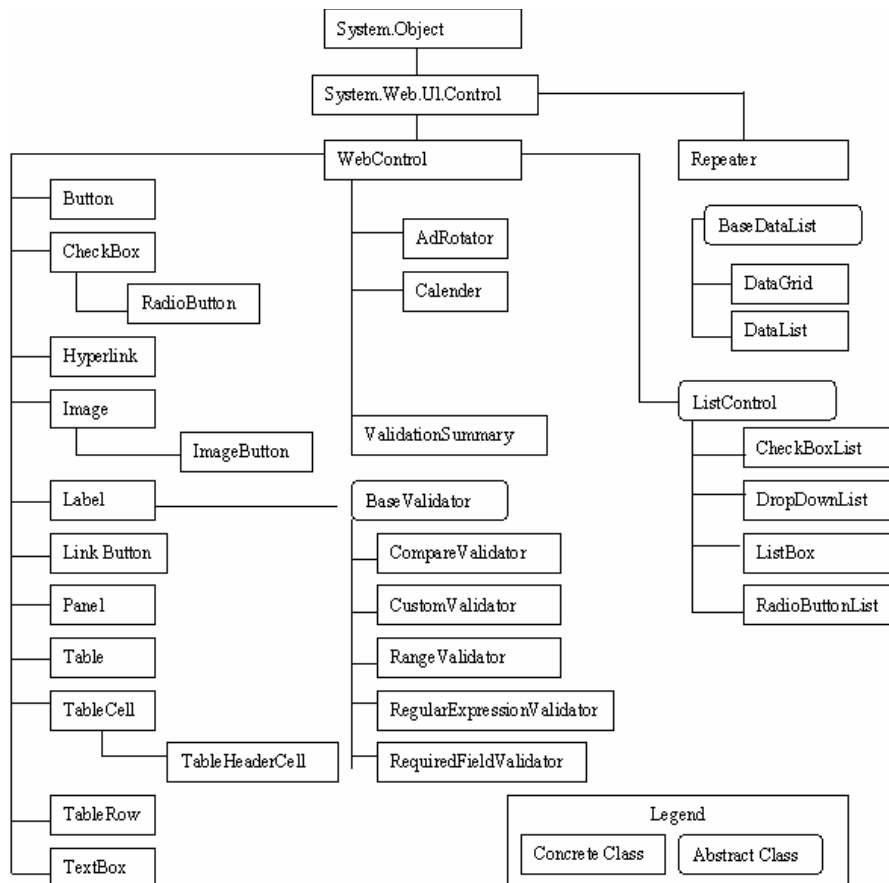


Figure 3.1

## Button

The **Button** Web control causes the Web Forms page to be posted back to the server.

## Syntax

```

<asp:Button runat="server"
  id="accessID"
  Text="label"
  Command="command"
  CommandArgument="commandArgument"
  OnClick="OnClickMethod"
/>

```

The properties of the button control are given below.

- **Command** - This property will be invoked when a button embedded in a container control is clicked
- **CommandArgument** - This is an optional command argument that is used in combination with the value of the Command property
- **Text** - This is the button caption

This control has only one event:

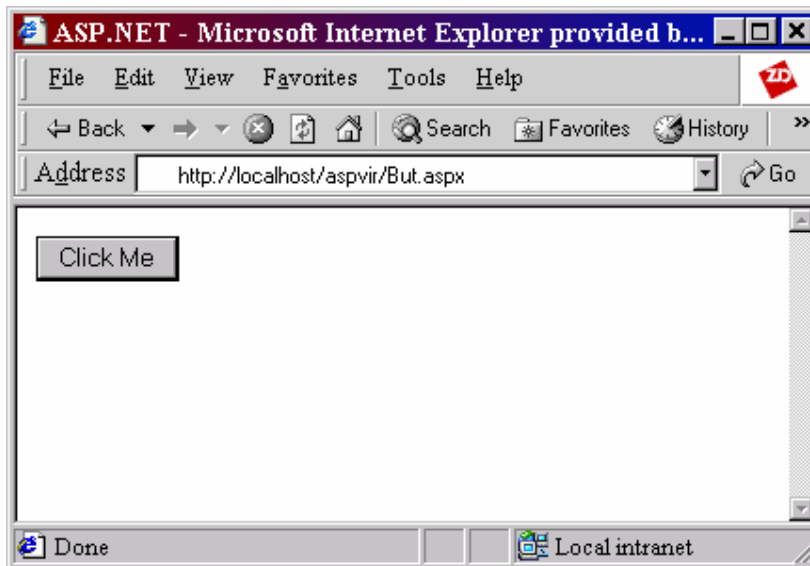
- **OnClick** - This event will be raised when the user clicks the button. This event always causes the page to be posted to the server



### Practice 3.1

The following example shows the code for displaying a button.

```
<HTML>
<TITLE> ASP.NET </TITLE>
<HEAD>
  <form runat="server">
    <p><asp:button id="Button1" text="Click Me" runat="server" />
  </form>
</HEAD>
</HTML>
```



In the above example a button control is created. The id of the button is Button1 and the text "click me" is the caption that appears on the button.

### Label

This control allows to display static text to be displayed on the page and to manipulate it programmatically.

### Syntax

```
<asp:Label runat="server"
  id="accessID"
  Text="label"
/>
```

The main property of the Label control is given below:

- **Text** - This the text to be displayed on the label

## Textbox

This control generates single- and multi-line text boxes.

### Syntax

```
<asp:TextBox runat="server"
  id=accessID
  AutoPostBack="True|False"
  Columns="characters"
  MaxLength="characters"
  Rows="rows"
  Text="text"
  TextMode="Single | Multiline | Password"
  Wrap="True|False"
  OnTextChanged="OnTextChangedMethod"
>
</asp:TextBox>
```

The properties of this control are:

- **AutoPostBack** - If this property is used in the control, it will automatically cause a postback to the server when it is true; otherwise false. The default is false
- **Columns** - The column width of the control is in characters. This property differs from the Width, which sets the absolute width of the control independent of the character spacing
- **MaxLength** - The maximum number of characters are allowed within the text box. This property has no effect unless the TextMode property is set to SingleLine or Password
- **Rows** - Rows indicates the number of rows within the text box. This property has no effect unless the TextMode property is set to MultiLine
- **Text** - Implies the text that the user has entered into the box
- **TextMode** - Indicates whether the text box is in single-line, multi-line, or password mode. Possible values are Single, MultiLine and Password
- **Wrap** - Indicates whether text should wrap around as users type text into a multi line control. This property has no effect unless the TextMode property is set to Multi line

This Control has only one event

- **OnTextChanged** - This event is raised on the server when the contents of the text box change. This event does not cause the Web Forms page to be posted to the server unless the AutoPostBack property is set to true.



---

### Practice 3.2

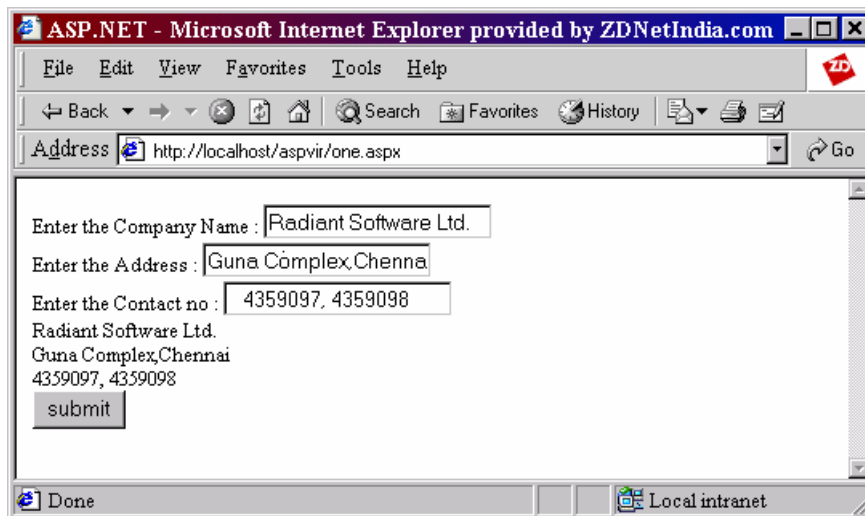
The following example illustrates the usage of label, Textbox controls.

```
<html>
<title> ASP.NET </title>
```

```

<head>
<script language=c# runat=server>
    void button_click(Object sender,EventArgs e)
    {
        label1.Text=text1.Text;
        label2.Text=text2.Text;
        label3.Text=text3.Text;
    }
</script>
<body>
<form runat=server>
    Enter the name : <asp:Textbox id=text1 runat=server/> <br>
    Enter the address : <asp:Textbox id=text2 runat=server/> <br>
    Enter the Contact no: <asp:Textbox id=text3 runat=server/> <br>
    <asp:Label id="label1" runat=server/> <br>
    <asp:Label id="label2" runat=server/> <br>
    <asp:Label id="label3" runat=server/> <br>
    <asp:button OnClick="button_click" text="submit" runat=server/>
</form>
</body>
</html>

```



In the above example, Textbox, label, button controls are used. It can be seen from the above example that the names entered in the textboxes are displayed in the label controls when the submit button is clicked.

### CheckBox

This control creates a check box on a Web Forms page, allowing users to set a true or false value for the item associated with the control.

### Syntax

```

<asp:CheckBox runat="server"
    id="accessID"

```

```

    AutoPostBack="True|False"
    Text="label"
    TextAlign="Right|Left"
    Checked="True|False"
    OnCheckedChanged="OnCheckedChangedMethod"
  />

```

The properties of the CheckBox are:

- **Checked** - This is true if the check box is checked, otherwise false. The default is false
- **TextAlign** - TextAlign is the position of the caption. The possible values are Right and Left. The default is Right
- **Text** - This is the check box caption

The CheckBox has the following event:

- **OnCheckedChanged** - This event is raised when the user clicks the checkbox. It does not cause the Web Forms page to be posted to the server unless the AutoPostBack property is set to true



### Practice 3.3

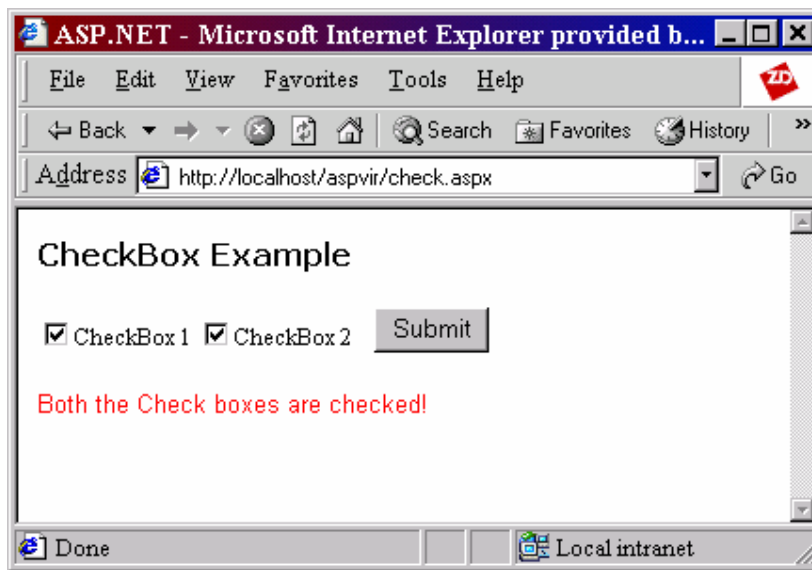
The following example illustrates the usage of the CheckBox Control.

```

<html>
<title> ASP.NET </title>
<head>
  <script language="C#" runat="server">
    void SubmitBtn_Click(Object Sender, EventArgs e) {
      if (Check1.Checked == true && Check2.Checked == true )
      {
        Labell.Text = "Both the Check boxes are checked!";
      }
      else if(Check1.Checked == false && Check2.Checked == true )
      {
        Labell.Text = "Check box 2 is checked!";
      }
      else if(Check1.Checked == true && Check2.Checked == false )
      {
        Labell.Text = "Check box 1 is checked!";
      }
      else
      {
        Labell.Text = "Both are not checked!";
      }
    }
  </script>
</head>
<body>
  <h3><font face="Verdana">CheckBox Example</font></h3>
  <form runat=server>
    <asp:CheckBox id=Check1 Text="CheckBox 1" runat="server" />

```

```
<asp:CheckBox id=Check2 Text="CheckBox 2" runat="server" />
 
 
<asp:button text="Submit" OnClick="SubmitBtn_Click" runat=server/>
<p>
<asp:Label id=Label1 font-name="arial" font-size="10pt" forecolor=red
runat="server" />
</form>
</body>
</html>
```



In the above example it can be seen that controls such as button, checkbox and labels are used. When the user checks the check box and clicks the "submit" button, a message is displayed in the label depending on the number of checks boxes that are checked.

## RadioButton

This control creates a single radio button on a Web Forms page.

### Syntax

```
<asp:RadioButton runat="server"
  id="accessID"
  AutoPostBack="True|False"
  Checked="True|False"
  GroupName="GroupName"
  Text="label"
  TextAlign="Right|Left"
  OnCheckedChanged="OnCheckedChangedMethod"
/>
```

The properties of the RadioButton are as follows:



- **AutoPostBack** - This returns true if client-side changes in the control automatically cause a postback to the server; otherwise false. The default is false
- **Checked** - This returns true if the radio button is checked, otherwise false. The default is false
- **GroupName** - The name of a group to which the radio button belongs. Radio buttons with the same group name are mutually exclusive
- **TextAlign** - This property indicates the position of the caption. The possible values are Right and Left. The default is Right
- **Text** - Text is the radio button caption

The RadioButton has the following event:

- **OnCheckedChanged** - This event will be raised when the user clicks the radio button and does not cause the Web Forms page to be posted to the server unless the AutoPostBack property is set to true

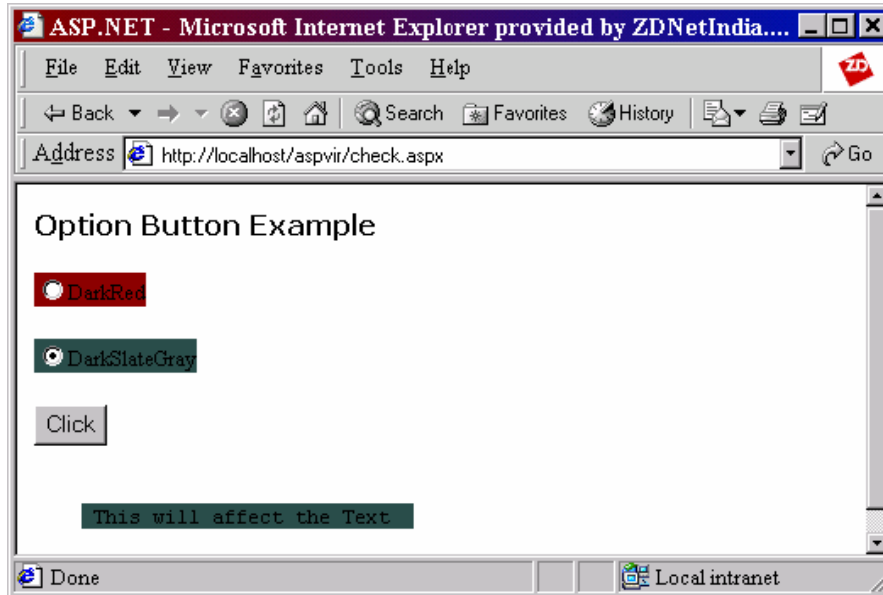


### Practice 3.4

The following example illustrates the usage of Radio Button.

```
<html>
<title> ASP.NET </title>
<head>
  <script language="C#" runat="server">
    void Button1_Click(Object sender, EventArgs e)
    {
      if (Radio1.GroupName=="Dark")
      {
        if (Radio1.Checked==true)
        {
          Span1.Style["background-color"] = Radio1.Text;
        }
        else if (Radio2.Checked==true)
        {
          Span1.Style["background-color"] = Radio2.Text;
        }
      }
    }
  </script>
</head>
<body>
  <h3><font face="Verdana">Option Button Example</font></h3>
  <form runat=server>
    <asp:RadioButton id="Radio1" GroupName="Dark"
      Text="DarkRed" BackColor="DarkRed" runat="server" />
    <p>
    <asp:RadioButton id="Radio2" GroupName="Dark"
      Text="DarkSlateGray" BackColor="DarkSlateGray" runat="server" />
    <p>
    <asp:Button id="Button1" OnClick="Button1_Click"
      Text="Click" runat="server" />
  <pre>
    <span id=Span1 runat=server > This will affect the Text </span>
  </pre>
  </form>
```

```
</body>
</html>
```



This program uses two radio buttons and a button control. When the user selects one of the Options and clicks the “Click” button, the corresponding color will be the background color of the text area.

### 3.4 Short Summary

- Web controls are mkservers controls with an abstract, strongly-typed object model.
- Web controls are more abstract than HTML controls, in that their object model does not necessarily reflect in the HTML syntax.
- All Web controls (except Repeater) derive directly or indirectly from the base class System.Web.UI.WebControls.WebControl.
- The Button Web control causes the Web Forms page to be posted back to the server.
- Label Web control allows to display static text on the page and manipulate it programmatically.
- Check box control creates a check box on a Web Forms page, allowing users to set a true or false value for the item associated with the control.
- Radio button control creates a single radio button on a Web Forms page.

### 3.5 Brain Storm

1. What are Web controls?
2. What are the various Web controls available in ASP.NET?



---

## ASP.NET Web Controls - II

---

### Objectives

In this lecture you will learn the following

- + Knowing about Image Controls & List Controls
- + Knowing about Link Button and Pannel
- + Knowing about AdRotator & Calendar

## Coverage Plan

---

### Lecture 4

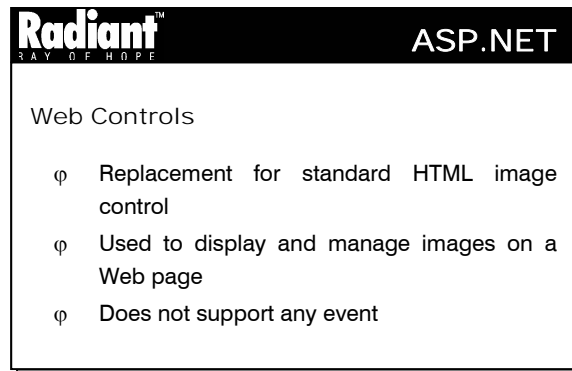
- 4.1 Snap Shot
- 4.2 ASP.NET Image Controls
- 4.3 ASP.NET List Controls
- 4.4 Link Button
- 4.5 Panel
- 4.6 AdRotator
- 4.7 Calendar
- 4.8 Short Summary
- 4.9 Brain Storm

## 4.1 Snap Shot

This session is aimed at making the learner familiar with advanced Web controls namely, ASP.NET Image controls, ASP.NET List controls, LinkButton, Panel, AdRotator and Calendar. This session also teaches how to apply styles to Web controls.

The ASP.NET list controls enable the users to choose one or more items from a list of items. The Panel is a container for other controls. The AdRotator control is used to display randomly selected advertisement banners on Web pages. The Calendar control displays a one-month calendar on the Web page, which can be used to view and select dates.

## 4.2 ASP.NET Image Controls



The **Image** control is a replacement for the standard HTML image control. It is used to display and manage images on a Web page. The syntax for using the Image control is:

```
<asp:Image id=imageId runat="server" ImageUrl=pathOfTheImage  
AlternateText=text ImageAlign=alignment />
```

where *alignment* takes one of the values NotSet, AbsBottom, AbsMiddle, BaseLine, Bottom, Left, Middle, Right, TextTop and Top.

The properties of the Image control are listed below:

- **ImageUrl**- The ImageUrl property specifies the URL of the image to be loaded into the image control
- **AlternateText** - **The AlternateText property specifies the text to be displayed if the image referenced by the ImageUrl property is not available**
- **ImageAlign** - The ImageAlign property specifies the alignment of the image with the text surrounding it.

The Image control does not support any event

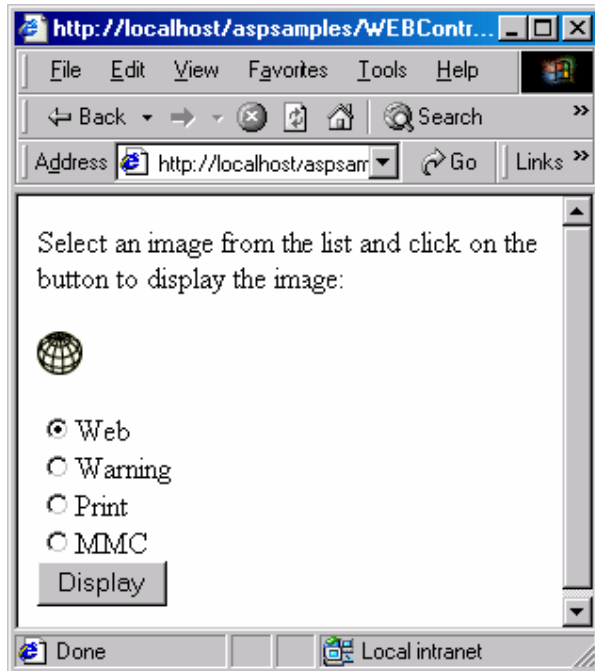


### Practice 4.1

The following example displays one of the four images, based on the user's selection.

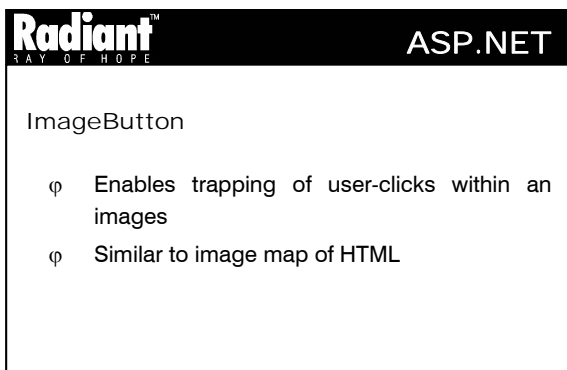
```
<html>
<head>
  <script language="C#" runat="server">
    void Btn_Click(Object sender, EventArgs e) {
      string s = "";
      if (Radiol.Checked)
        s = "web.gif";
      else if (Radio2.Checked)
        s = "warning.gif";
      else if (Radio3.Checked)
        s = "print.gif";
      else
        s = "mmc.gif";
      Img.ImageUrl="G:/Inetpub/wwwroot/" + s;
    }
  </script>
</head>
<body>
  <form runat=server>
    Select an image from the list and click on the button to display the image:
    <br><br>
    <asp:Image ID="Img" ImageUrl="G:/Inetpub/wwwroot/web.gif"
      AlternateText="Sample Images" runat="server"/>
    <br><br>
    <asp:RadioButton id=Radiol Text="Web" Checked="True" GroupName="Group1"
      runat="server" /><br>
    <asp:RadioButton id=Radio2 Text="Warning" GroupName="Group1"runat="server"/> <br>
    <asp:RadioButton id=Radio3 Text="Print" GroupName="Group1" runat="server" />
    <br>
    <asp:RadioButton id=Radio4 Text="MMC" GroupName="Group1" runat="server"/>
    <br>
    <asp:button text="Display" OnClick="Btn_Click" runat=server/>
  </form>
</body>
</html>
```





The example uses four radio buttons to enable the user to select his/her option. Since the default option is “Web”, the corresponding image is displayed. When the user selects one of the four options and clicks the button “Display”, the “OnClick” event of the button invokes the method “Btn\_Click”. This method checks which one among the four radio buttons is selected and based on this, it displays the corresponding image.

### ImageButton



The **ImageButton** control enables trapping of user-clicks within an image. This is similar to the image map of HTML. The syntax for using the ImageButton is:

```
<asp:ImageButton id=imageId runat="server" ImageUrl=pathOfTheImage
Command=cmdToInvoke CommandArgument=cmdargument OnClick=method />
```

The properties of the ImageButton are:

- **ImageUrl** - The ImageUrl property specifies the URL of the image to be displayed on the Web page

- **Command** - The Command property denotes the command to be executed when the button contained in a container control is clicked
- **CommandArgument** - The CommandArgument property is used along with the Command property if a value has to be supplied to the command invoked by that property

This control has only one event namely,

```
OnClick(Object sender, ImageClickEventArgs e)
```

that is raised when the user clicks the button and causes the page to be posted to the server. The second argument to the event specifies the position of the click.

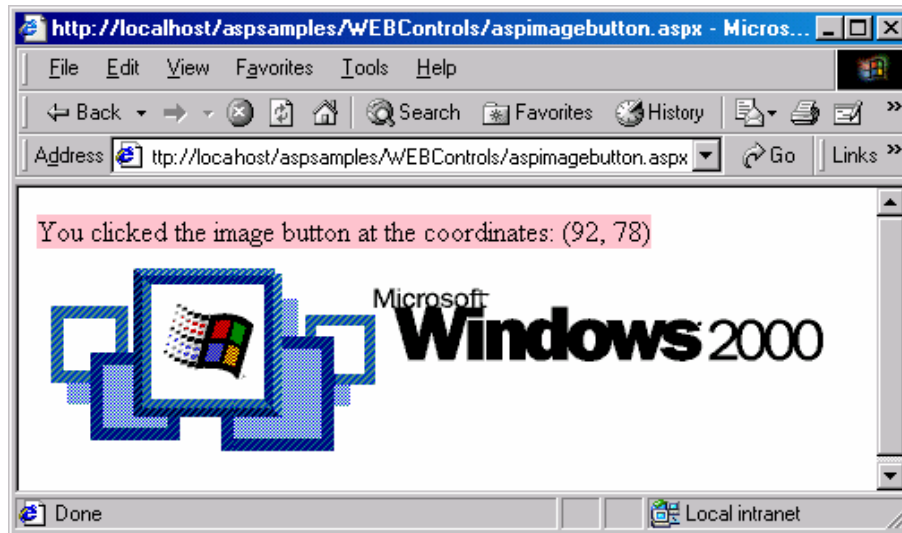


## Practice 4.2

The following example displays an ImageButton and when the user clicks on the button, it displays the coordinates of the image at which the user has clicked.

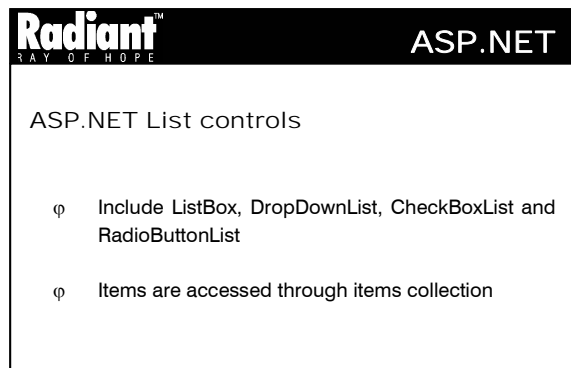
```
<html>
<head>
  <script language="C#" runat="server">
    void ImgBtn_Click(object Source, ImageClickEventArgs e)
    {
      Label1.Text="You clicked the image button at the coordinates:
(" + e.X.ToString() + ", " + e.Y.ToString() + ")";
    }
  </script>
</head>
<body>
  <form runat="server">
    <asp:label id="Label1" BackColor="Pink" runat="server"/>
    <asp:ImageButton id="imgbtn" runat="server"
      AlternateText="Windows 2000 Logo"
      ImageAlign="left"
      ImageUrl="g:/Inetpub/wwwroot/win2000.gif"
      OnClick="ImgBtn_Click"/>
  </form>
</body>
</html>
```





The example contains an **ImageButton** and a **Label**. When the user clicks on any part of the **ImageButton**, the **OnClick** event of the **ImageButton** is invoked. This event calls the method “**ImgBtn\_Click**” that displays the message “You clicked the image button at the coordinates:” followed by the x and y coordinates of the point at which the user has clicked on the **ImageButton**.

### 4.3 ASP.NET List Controls



The ASP.NET List controls include **ListBox**, **DropDownList**, **CheckBoxList** and **RadioButtonList**. The items for the list controls are accessed through the **Items** collection. The **Items** collections can be used to add items, remove items, etc., from any control that uses the collection. Processes like adding items, determining the selection, setting the selection and responding to the changes are common to all the list controls. So, these processes will be explained after introducing these controls.

#### Listbox

The **ListBox** control is used to allow the user to select one or more items from a specified list. It displays more than one item at the same time. The number of controls that should be visible can be set for the control. Each item of the list has the properties **Text**, **Value** and **Selected**. The **Text** property denotes the text to be displayed in the list and the **Value** property denotes the value associated with it. **Selected** takes a boolean value that denotes whether the item is selected or not. The following is the syntax for using the **ListBox**:

```

<asp:ListBox id=listboxid runat="server" DataSource=expression
DataTextField=source DataValueField= source AutoPostBack= value
Rows=numberOfRows SelectionMode=mode OnSelectedIndexChanged=method />
    <asp:Listitem value="value" selected=value> Text </asp:Listitem>
</asp:ListBox>

```

where **AutoPostBack** and **selected** can take one of the values "true" or "false".

The properties of the **ListBox** are as follows:

- **SelectionMode** - This property can either be "Single" or "Multiple" depending on whether the user wants to select a single item or multiple items. If it is set to "Single" then the **SelectedItem** property returns the selected item. If **SelectionMode** is set to "Multiple", then the **SelectedItems** property is used to retrieve the selected items
- **DataSource** - This property denotes the object that supports **ICollection** Interface
- **DataTextField** - The **DataTextField** property denotes the source of the **Text** property of the items
- **DataValueField** - The **DataValueField** property denotes the source of the **Value** property of the items
- **Rows** - The **Rows** property determines the number of rows to be visible in the list

The event **OnSelectedIndexChanged(Object sender, EventArgs e)** is raised on the server when the selection of the **ListBox** is changed.

### DropDownList

The **DropDownList** control enables users to select from a single-selection drop-down list. It is similar to the **ListBox** control, but it shows only the selected item. The syntax, properties and event for the **DropDownList** are similar to those of the **ListBox** except that **DropDownList** does not support multiple selection.

### CheckBoxList

The **CheckBoxList** control contains a group of checkbox controls. The syntax for using the **CheckBoxList** control is:

```

<asp:CheckBoxList id=CheckBoxId runat="server" AutoPostBack=value
CellPadding= space DataSource= expression DataTextField=source
DataValueField=source RepeatColumns=noOfCols RepeatDirection=direction
RepeatLayout=layout TextAlign=alignment OnSelectedIndexChanged=method/>
    <asp:ListItem text=caption value=value selected=value />
</asp:CheckBoxList>

```

In the above syntax, **AutoPostBack**, **DataSource**, **DataTextField**, **DataValueField**, **OnSelectedIndexChanged** and the properties of the list items are similar to those of the list box. The other properties of the **CheckBoxList** are:

- **CellPadding** - The **CellPadding** property determines the spacing between the controls

- **RepeatColumns** – The RepeatColumns property determines the number of columns in which the controls will be displayed
- **RepeatDirection** – The RepeatDirection property determines the direction in which the controls are to be displayed and can be either “Vertical” or “Horizontal”
- **RepeatLayout** – The RepeatLayout property determines whether the control renders in-line (when it takes the value “Flow”) or in a table (when it takes the value “Table”)
- **TextAlign** – TextAlign determines the position of the text with respect to the control and it can take one of the values “Right” or “Left”. The default alignment is Right

### RadioButtonList

The **RadioButtonList** control provides a radio button group that can be dynamically generated through databinding. The radio buttons in the group are mutually exclusive, i.e., only one of them can be selected at a time.

```
<asp:RadioButtonList id=buttonId runat="server" AutoPostBack=value
CellPadding=space DataSource=expression DataTextField=source
DataValueField=source RepeatColumns=noOfCols RepeatDirection=direction
RepeatLayout=layout TextAlign=align OnSelectedIndexChanged=method />
    <asp:ListItem text=caption value=value selected=value />
</asp:RadioButtonList>
```

All the properties mentioned in the syntax are similar to those of the **CheckBoxList** control.

### Adding items to the ASP.NET List Control

Following are the steps to add an item programmatically to a list control:

- Create a new object of **ListItem** and set its **Text** and **Value** properties
- Add the new object to the Items collection through its **Add** method

For instance, to add a new item to a **ListBox** named **list1**, the following lines of code are written:

```
Listitem it = new ListItem();
it.Text = "New item";
it.Value = "10";
list1.Items.Add(it);
```

### Determining the selection in ASP.NET List Control

If the control allows single-selection, then the value of **SelectedIndex** will denote the index of the item that is selected. If nothing is selected then the value of SelectedIndex is -1. The contents of the selected item are retrieved using the SelectedItem property that returns an object of **ListItem**. The **Text** and **Value** properties of the **ListItem** are used to get the details about the item.

If the control allows multi-selection, then the **Selected** property of each item in the collection is checked to see if it is true. If it is true then its contents can be accessed through its **Text** and **Value** properties.

The following part of a code checks the items of a list box named **list1** to see if they are selected and displays the text of the selected items.

```
for (i = 0; i < list1.Items.count; i++)
    if(list1.Items[i].Selected == true)
        document.write(list1.items[i].Text + " ");
```

### Setting the selection in ASP.NET List Control

To set a single selection, set the **SelectedIndex** property of the control to the index of the item to be selected. If the **SelectedIndex** property is set to -1 then no item is selected from the list. To set multiple selections, the **Selected** property of those items is set to **true**. It must be noted that multiple items can be selected only if the control's **SelectionMode** is set to **Multiple**.

The following line of code selects the second item in a list box named **list1**:

```
list1.SelectedIndex = 1;
```



#### Practice 4.3

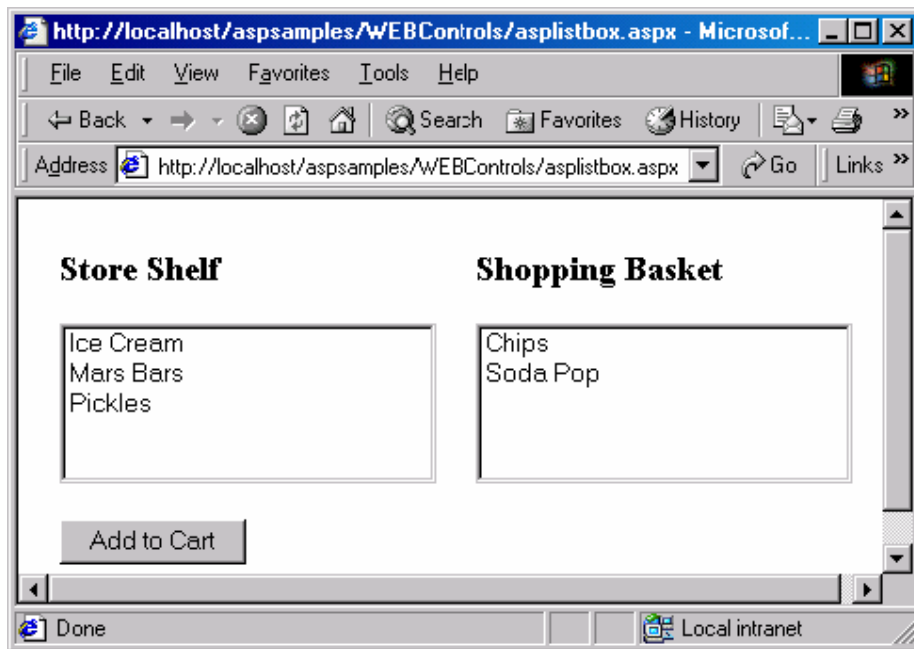
The following example moves items that are selected by the user from the "Store Shelf" ListBox to the "Shopping Basket" ListBox.

```
<html>
<head>
<Script language="c#" runat="Server">
    void addCart(Object sender,EventArgs e)
    {
        if (Store.SelectedIndex > -1)
        {
            shoppingBasket.Items.Add( new ListItem(Store.SelectedItem.Value ) );
            Store.Items.Remove( Store.SelectedItem.Value );
        }
    }
</Script>
</head>

<body>
<form runat="server">
    <table cellpadding="10">
        <tr>
            <td valign="top">
                <h3>Store Shelf</h3>
                <asp:listbox id="Store" width="200" runat="server">
                    <asp:listitem>Ice Cream</asp:listitem>
                    <asp:listitem>Mars Bars</asp:listitem>
                    <asp:listitem>Chips</asp:listitem>
                    <asp:listitem>Soda Pop</asp:listitem>
                    <asp:listitem>Pickles</asp:listitem>
                </asp:listbox>
                <br>
                <asp:button id="add2Cart" text="Add to Cart"  OnClick="addCart"
                    runat="server" />
            </td>

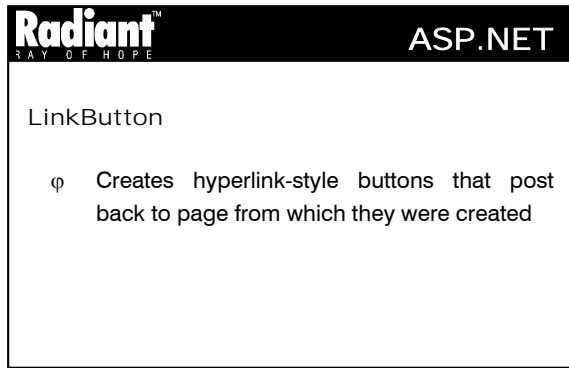
            <td valign="top">
                <h3>Shopping Basket</h3>
                <asp:listbox id="shoppingBasket" width="200" runat="server" />
            </td>
        </tr>
    </table>
```

```
</form>  
  
</body>  
</html>
```



The above example contains two **ListBoxes** “Store Shelf” and “Shopping Cart” and a **Button** “Add to Cart”. When the user selects an item from the first **Listbox** and clicks on the **Button**, the **OnClick** event of the **Button** gets fired. This event calls the method “addCart”. This method checks if any item is selected from the first **Listbox** by checking if its **SelectedIndex** is greater than -1. If any item has been selected in the **Listbox**, then the corresponding item is added to the second **Listbox** and then removed from the first.

## 4.4 LinkButton



The LinkButton control creates hyperlink-style buttons that post back to the page from which they were created. The syntax for using a link button is:

```
<asp:LinkButton id=buttonId runat="server" Text=caption Command=cmd
CommandArgument=argument OnClick=method />
```

The following are the properties of the LinkButton:

- **Text** - This property denotes the text that will be displayed in the link. It can also be specified outside the tag
- **Command** - The Command property denotes the command to be executed when the button contained in a container control is clicked
- **CommandArgument** - This property is used along with the Command property if a value has to be supplied to the command invoked by the Command property

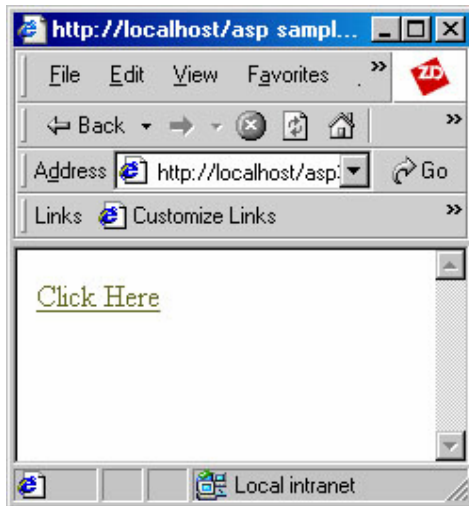
The `OnClick(Object sender, EventArgs e)` event is raised when the user clicks the button and causes the page to be posted to the server.



### Practice 4.4

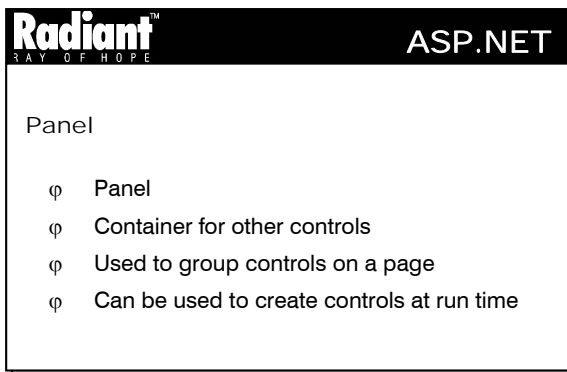
The following example uses LinkButton to enable the user to navigate to Microsoft's home page.

```
<html>
<head>
  <script language="C#" runat="server">
    void LinkBtn_Click(Object sender, EventArgs e) {
      Page.Navigate("http://www.microsoft.com");
    }
  </script>
</head>
<body>
  <form runat=server>
<asp:LinkButton Text="Click Here" onclick="LinkBtn_Click" runat="server"/>
  </form>
</body>
</html>
```



The example displays a **LinkButton** with the text “Click Here” displayed on the screen. When the user clicks on the **LinkButton**, the **OnClick** event of the **LinkButton** is invoked. This event calls the “LinkBtn\_Click” method that directs the user to the home page of Microsoft.

#### 4.5 Panel



The **Panel** is a container for other controls. It is used to group controls on a page. It enables the user to execute a command only on one group or certain groups of controls on a page. It can be used to create controls at run time.

```
<asp:Panel id=panelId runat="server" BackImageUrl=url HorizontalAlign=align
Wrap=value >
...
</asp:Panel>
```

The following are the properties of the Panel control:

- **BackColor** - The BackColor property specifies the color of the background of the panel
- **BackImageUrl** - The BackImageUrl property specifies the URL of the image to be displayed as a background of the panel
- **HorizontalAlign** - The HorizontalAlign property specifies the alignment of the panel with respect to the surrounding text. It can take one of the values "Center", "Justify", "Left", "NotSet" and "Right"
- **Wrap** - The Wrap property can be set to either "True" or "False". If it is set to "True" then the content can be wrapped. The default value is "True"



### Practice 4.5

The following example sets the background for a panel and adds a textbox to it, when the user clicks the corresponding buttons.

```
<html>
<head>
  <script language="c#" runat="server">
    void Btn1_Click(Object sender, EventArgs e)
    {
        Panell.BackColor = "G:/Inetpub/wwwroot/win2000.gif";
    }
    void Btn2_Click(Object sender, EventArgs e)
    {
        TextBox t = new TextBox();
        t.ID = "Text1";
        t.Text="Hai";
        t.Width=50;
        Panell.Controls.Add(t);
    }
  </script>
</head>
<body>
  <form runat=server>
<asp:Panel id="Panell" Height=120 Width=450 BackColor="Gainsboro"
  Wrap="True" HorizontalAlign="Right" runat="server"/>
  <br>
<asp:Button id="Btn1" OnClick="Btn1_Click" Text="Set the label back
ground" runat="server"/>
  <asp:Button id="Btn2" OnClick="Btn2_Click" Text="Create a
  TextBox" runat="server"/>
  </form>
</body>
</html>
```

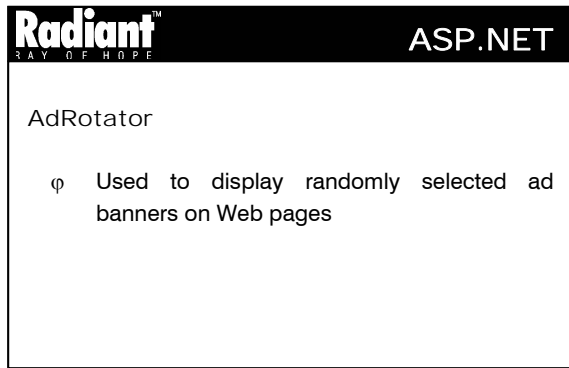






The above example consists of a **Panel** and two **Buttons**. When the first Button, namely “Set the label background” is clicked, its **OnClick** event is invoked that calls the method “Btn1\_Click”. This method sets the **BackColor** property of the **Panel** to display the “Windows 2000 logo”. When the second Button, namely “Create a Text Box” is clicked, its **OnClick** event is invoked that calls the method “Btn2\_Click”. This method creates a new **TextBox** and places it in the right top of the **Panel**.

#### 4.6 AdRotator



The **AdRotator** control is used to display randomly selected advertisement banners on Web pages.

```
<asp:AdRotator id=rotatorId runat="server" AdvertisementFile=file
KeywordFilter=filter Target=targetName OnAdCreated=method />
</asp:AdRotator>
```

The properties of the AdRotator are:

- **AdvertisementFile** - The AdvertisementFile property (a required property) specifies the path to the XML file containing the information about the advertisement
- **KeywordFilter** - This property indicates the category filter to pass to the source of advertisements
- **Target** - This property specifies the name of the browser in which the advertisement will be displayed

The only event of the AdRotator control is **OnAdCreated(Object sender, AdCreatedEventArgs e)**. This event is raised on the server once per round trip, after the control is created and before the page is rendered. When an AdvertisementFile is specified, this event is raised after an advertisement is selected from the XML file.

The source of the advertisement is specified in an XML file using the following syntax:

```
<Advertisements>
  <Ad>
    <ImageUrl> url </ImageUrl>
    <TargetUrl> url </TargetUrl>
    <AlternateText> tooltip </AlternateText>
    <Keyword>filter </Keyword>
    <Impressions> relativeWeight </Impressions>
  </Ad>
</Advertisements>
```

The properties of the element are:

- **ImageUrl** - The ImageUrl property contains the URL of the image file
- **TargetUrl** - The TargetUrl property contains the URL of the page that should be displayed when the user clicks on the ad
- **AlternateText** - The AlternateText property contains the text that will be displayed if the image is not loaded
- **Keyword** - The Keyword property specifies the category for the advertisement
- **Impressions** - The Impressions property denotes the weight of the advertisement with respect to the other ads in the file. It determines how often the advertisement will be displayed. The higher the value of Impressions, the more the importance given to the advertisement



### Practice 4.6

The following example uses AdRotator to display two advertisements. When the user clicks on an advertisement, he/she is directed to the corresponding Web page.

**The XML file:**

```
<Advertisements>
  <Ad>
    <ImageUrl>nat1.bmp</ImageUrl>
    <NavigateUrl>http://www.kodaihotels.com</NavigateUrl>
    <AlternateText> Click here to book a hotel at Kodaikkanal</AlternateText>
    <Keyword>websites </Keyword>
    <Impressions>10</Impressions>
```

```

</Ad>
<Ad>
  <ImageUrl>nat2.bmp</ImageUrl>
  <NavigateUrl>http://www.udagamandal.com</NavigateUrl>
  <AlternateText> Click here to book a hotel at Ooty </AlternateText>
  <Keyword>websites </Keyword>
  <Impressions>10</Impressions>
</Ad>

</Advertisements>

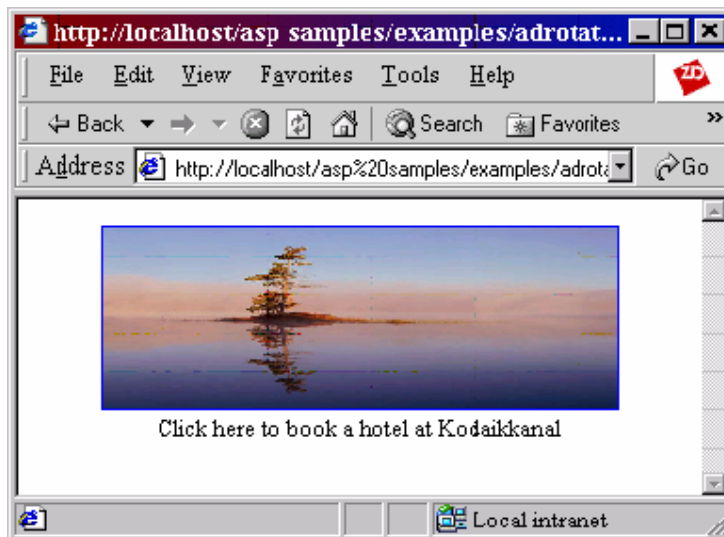
```

**The ASPX file:**

```

<html>
<head>
  <script language="c#" runat="server">
    void Ready(Object sender, AdCreatedEventArgs e) {
      labell1.Text= e.AlternateText ;
    }
  </script>
</head>
<body>
  <Center>
    <asp:AdRotator Width = "80%" Height=100 id=a1
      AdvertisementFile="adrot.xml" BorderWidth=1
      OnAdCreated = "Ready" runat=server />
    <br>
    <asp:label id="labell1" runat="server"/>
  </Center>
</body>
</html>

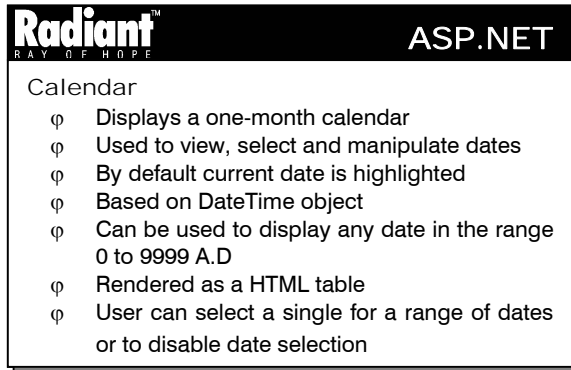
```



The example consists of an **AdRotator** control that is centered in the form using the **<Center>** tag. The **Width** of the **AdRotator** is set to be 80% of the width of the page. The **AdvertisementFile** property

specifies the XML file that has to be referred by the ASPX file in order to display the advertisements. The XML file specifies the image to be used, its alternate text, etc.

## 4.7 Calendar



The **Calendar** control displays a one-month calendar on the Web page. It can be used to view and select dates. By default the current date is highlighted. The Calendar control can be used to display details of each day, like the daily schedule, appointments, etc. It is based on the DateTime object of the .NET framework and can be used to display any date in the range between 0 to 9999 A.D. It is actually rendered as a HTML table on the Web page. Therefore, its properties are similar to that of the table.

The user can change the current date, navigate to another month or year, and select a single date, week or month. The user can also select a range of dates using this control. By default, the control allows the user to select a single date. The settings can also be changed to disable the selection of the date. If day selection is enabled, each date acts as a LinkButton and raises an event when clicked. If the week or month selection is enabled, a column of links is added to the left side of the Calendar so that the user can select an entire week or month. The appearance of the Calendar control like its color, font, borders etc., can be modified using the style objects.

The properties of the Calendar object are listed in Table 4.1

Property	Description
CellPadding	Determines the space between cells in pixels.
CellSpacing	Determines the space between the border of a cell and the text contained in the cell.
DayNameFormat	Determines the format of the name of the days. The value can be one of "Full", "Short", "FirstLetter" or "FirstTwoLetters". The default is "Short".
FirstDayOfWeek	Determines the day that will be displayed in the first column. The week will be set to start from this day. The default value depends on the local settings of the server.
NextMonthText	Determines the text that will be displayed in the hyperlink for the next month. This requires the <b>ShowNextPrevMonth</b> property to be set to true.

NextPrevFormat	Specifies the format in which the next month and previous month links are displayed. The default value is "Custom".
PrevMonthText	Determines the text that will be displayed in the hyperlink for the previous month. This requires the <b>ShowNextPrevMonth</b> property to be set to true.
SelectedDate	Specifies the date that will be highlighted. The default value is the value of the <b>TodayDate</b> property.
SelectedDates	A collection of the dates highlighted in the Calendar. It is a read-only property.
SelectionMode	Specifies whether the user can select a day, week or month. It can take one of the values "None", "Date", "DateWeek" or "DateWeekMonth". The default value is "Date". Setting this property to "None" disables date selection.
SelectMonthText	Represents the text to be displayed for month selection in the selector column if <b>SelectionMode</b> is set to "DateWeekMonth".
SelectWeekText	Represents the text to be displayed for week selection in the selector column if <b>SelectionMode</b> is set to "DateWeek" or "DateWeekMonth".
ShowDayHeader	Determines whether or not the names of the days of the week are displayed.
ShowGridLines	Specifies whether or not the border is displayed around each day.
ShowNextPrevMonth	Specifies whether or not next and previous month hyperlinks are displayed.
ShowTitle	Specifies whether or not the title is displayed.
TitleFormat	Specifies the format of the month in the title bar. The default value is "Month".
TodayDate	Determines the current date.
VisibleDate	Specifies the month to be displayed. The date can be any day within the month.

**Table 4.1**

The events of the Calendar control are raised on the server. They are:

- **OnDayRender(Object sender, DayRenderEventArgs e)** - Raised when the cell for each day is created
- **OnVisibleMonthChanged(Object sender, MonthChangedEventArgs e)** - Raised when the hyperlink for the next month or the previous month is clicked
- **OnSelectionChanged(Object sender, EventArgs e)** - Raised when the SelectionMode is changed. This event causes the page to be posted to the server



### Practice 4.7

Following is an example using the Calendar control.

```
<form runat = "server">
<asp:Calendar id=Calendar2 runat="server"

BackColor="lightgreen"
BorderColor="red"
BorderWidth=10
BorderStyle="inset"

CellPadding=2
CellSpacing=5
DayNameFormat="firstletter"
DayHeaderStyle-Font-Bold="True"
DayHeaderStyle-Font-Italic="true"
DayHeaderStyle-Font-Oblique="true"
DayStyle-BackColor="white"

FirstDayOfWeek="sunday"
Font-Name="Verdana;Arial" Font-Size="12px"
ForeColor="red"

Height="280px"

NextMonthText=">"
NextPrevFormat="fullmonth"
NextPrevStyle-ForeColor="white"
NextPrevStyle-Font-Size="10px"
OtherMonthDayStyle-ForeColor="green"
SelectionMode="DayWeekMonth"
SelectedDayStyle-BackColor="#ffcc66"
SelectedDayStyle-Font-Bold="True"
SelectorStyle-BackColor="#99ccff"
SelectorStyle-ForeColor="Olive"
SelectorStyle-Font-Size="9px"
SelectWeekText = "Week"
SelectMonthText = "Month"
TodayDayStyle-Font-Bold="True"
TodayDayStyle-ForeColor="blue"
TitleFormat="Monthyear"
TitleStyle-BackColor="blue"
TitleStyle-ForeColor="white"
TitleStyle-Font-Bold="True"

Width="500px"
/>
</form>
```



May 2001							
Month	S	M	T	W	T	F	S
Week	29	30	1	2	3	4	5
Week	6	7	8	9	10	11	12
Week	13	14	15	16	17	18	19
Week	20	21	22	23	24	25	26
Week	27	28	29	30	31	1	2
Week	3	4	5	6	7	8	9

An important point to be noted in the above example is that the Calendar control is enclosed within the Form tag. This enables the navigation between months. If the control is not embedded in the form, it will not be submitted to the server and thus will not support the navigation between months. The example illustrates how the control can be customized by setting many of its properties.

#### 4.8 Short Summary

- The Image control is a replacement for the standard HTML image control. It is used to display and manage images on a Web page
- The ImageButton control enables trapping of user-clicks within an image
- The ASP.NET List controls include ListBox, DropDownList, CheckBoxList and RadioButtonList. The items for the list controls are accessed through the Items collection
- The LinkButton control creates hyperlink-style buttons that post back to the page from which they were created
- The Panel is a container for other controls. It is used to group controls on a page and perform some action only to one group or certain groups of controls on a page
- The AdRotator control is used to display randomly selected advertisement banners on Web pages
- The Calendar control displays a one-month calendar on the Web page. It can be used to view and select dates
- The Web controls provide additional support for styles by adding properties for commonly used style settings

#### 4.9 Brain Storm

1. Differentiate between Image and ImageButton controls.
2. Differentiate between ListBox and DropDownList controls.
3. What is the use of the CheckBoxList control?
4. How is an advertisement file specified in an AdRotator?
5. Which property of the AdRotator is used to specify the importance of an advertisement with respect to other advertisements?
6. Which property of the Calendar control is used to set the Text for the hyperlink for the next month?
7. Can styles be applied to ASP Web Controls?





---

## Validation Controls

---

### Objectives

In this lecture you will learn the following

- + Knowing about the Validation in ASP.NET
- + What is meant by Validation Summary Controls?

---

## Coverage Plan

---

### Lecture 5

- 5.1 Snap Shot
- 5.2 Validation ASP.NET
- 5.3 Required Filed Validation
- 5.4 Compare Validator
- 5.5 Range Validator
- 5.6 Regular Expression Validation
- 5.7 Custom Validator
- 5.8 Validation Summary Controls
- 5.9 Short Summary
- 5.10 Brain Storm

## 5.1 Snap Shot

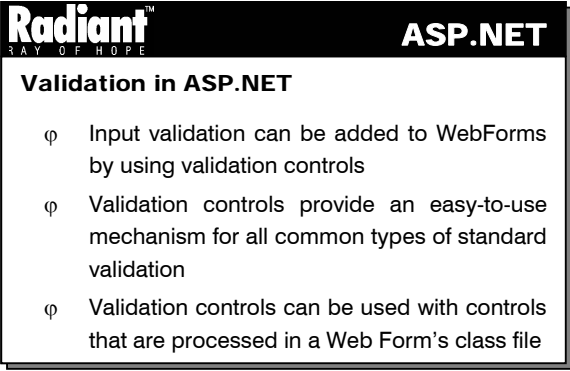
This session focusses on how input validation can be added to WebForms by means of validation controls. It also discusses the various validation controls available in ASP.NET. An important aspect of creating Web Forms pages for user is to enable Validation of user-input. The Web Forms framework provides a set of validation controls that provide an easy-to-use but powerful way to check for errors, and if necessary, display messages to the user. This session introduces to the validation controls in broader prospective.

Let us consider a typical web-based form wherein users have to provide their names and ages. A visitor to the page types his name properly, but say types in “twenty-four” for his age instead of the number “24”. When the user presses the Submit button on the form, his name and age (“twenty-four”) are sent to the Web server, where there is a program waiting for his input. The program expects to get proper data from the user to add it to the database. But instead, it gets a string of alphabetic characters for the age. So the Web server creates an HTML page that asks the user to enter a valid age and sends this page to the user’s browser. The user might complete the field correctly the second time and send the data back to the server.

This whole process may take just half a second, or several seconds, depending on several factors such as the speed of the user’s connection, the speed of the Net, the Web server’s load, and so on. These trips to the server for simple validations are a waste of resources.

Moving these simple validation tasks from the server to the client is a simple task and a big advantage of client-side scripting. If the above Web page had a simple validation script, then the user would be immediately notified of his data entry error. More importantly, the page would not have been sent to the server until it was validated and complete.

## 5.2 Validation in ASP.NET



The image shows a presentation slide with a black header. On the left, the word "Radiant" is written in a stylized white font with "RAY OF HOPE" underneath it. On the right, "ASP.NET" is written in white. The main content area has a white background with a black border. It contains the title "Validation in ASP.NET" and a bulleted list of three points:

- ☐ Input validation can be added to WebForms by using validation controls
- ☐ Validation controls provide an easy-to-use mechanism for all common types of standard validation
- ☐ Validation controls can be used with controls that are processed in a Web Form's class file

Validating user input can be a time-consuming and tedious process. ASP.NET provides developers with a set of pre-built validation controls that will help in speeding up and simplifying the task of validation. This session provides a brief overview of using the Validation controls.

Input validation can be added to WebForms by using validation controls. Validation controls provide an easy-to-use mechanism for all common types of standard validations – (eg. testing for valid dates or values within a range.) Moreover, these controls can be used to provide custom-written validations. In addition, validation controls allow to completely customize the display of error information to the user.

Validation controls can be used with any control that is processed in a Web Form’s class file, including both HTML and ASP.NET server controls.

## Using Validation Controls

User input validation can be enabled by adding validation controls to the form. There are controls for different types of validation, such as range-checking or pattern-matching.

Each validation control references an input control (a server control) elsewhere on the page. When the user's input is being processed (for example, when the form is submitted), the page framework passes the user's entry to the appropriate validation control or controls. The validation controls test the user's input and set a property to indicate whether the entry has passed the test or not. After all validation controls have been called, a property on the page is set; if any of the controls show that a validation check has failed, the entire page is set to invalid.

The state of the page and that of the individual controls can be tested by the code. For example, the state of the validation controls have to be tested before updating a data record with information entered by the user. If an invalid state is detected, the the update has to be bypassed. Typically, if any validation check fails, all the processing is skipped and the page is returned to the user. Validation controls detect errors and then produce an error message that appears on the page.

## Validating for Multiple Conditions

Generally speaking, each validation control performs only one test. (eg. to establish that a field is required and limited to accepting dates within a specific range.) More than one validation control can be attached to an input field on a form. In that case, the tests performed by the controls are resolved using a logical AND. The data input by the user must pass all the tests in order to be considered as valid.

In some instances, several different entries might be valid. For example, if the code prompts for a phone number, the users may be allowed to enter a local number, a long-distance number, or an international number. This situation arises primarily where checking for specific patterns of numbers or characters. To perform this type of test, the pattern-matching validation control is used to specify the multiple valid patterns within the control.

## Displaying Error Information

Validation controls are normally not visible in the rendered form. However, if the control detects an error, it produces error message text. The error message can be displayed in a variety of ways, as listed in the following table.

Display option	Description
In place	Each validation control can individually display an error message or appear in place (usually next to the control where the error occurred).
Summary	Validation errors can be collected and displayed in one place, for example, at the top of the page. (This strategy is often used in combination with displaying the error message next to the input fields with errors.) If the user is working in a browser that supports DHTML, the summary can be displayed in a pop-up message box.
Custom	Create custom error display by capturing the error information and designing user-defined output.

**Note :** If in-place or summary display options are used, the error message text can be formatted using HTML.

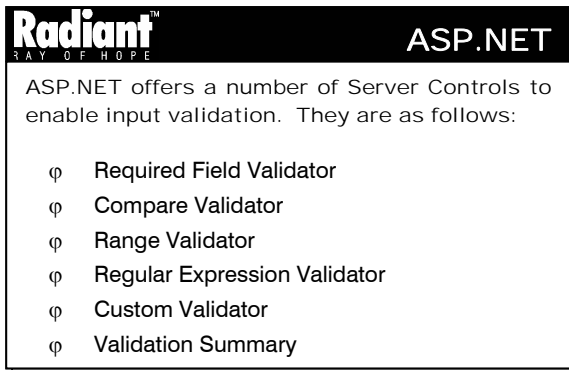
## Server-Side and Client-Side Validation

Validation controls perform input checking in server code. When the user submits a form to the server, the validation controls are invoked to review the user's input, control by control. If an error occurs in any of the input controls, the page itself is set to an invalid state. Hence, validity has to be tested before the code runs.

If the user is working with a browser that supports DHTML, the validation controls can also perform validation using client script. This can substantially improve response time in the page. Errors are detected immediately and error messages are displayed as soon as the user leaves the control containing the error. If client-side validation is available, the user has greater control over the layout of error messages and can display an error summary in a pop-up message box.

The page framework performs validation on the server even if the validation controls have already performed it on the client, so that the test for validity can be within the server-based event-handling methods. In addition, it helps prevent users from being able to bypass validation by impersonating as another user or a pre-approved transaction.

### Types of Validation



Each of these controls offers its own type of validation.

### 5.3 RequiredFieldValidator

The RequiredField Validator is used to ensure that the user does not leave a field blank. For fields where it does not matter as to what type of data the user enters, as long as they enter something, this is the validator to use. This is actually one of the most widely used validation controls. This validation ensures that the user does not skip an entry.

#### Automatic Client-side or Server-side Implementation

One of the dangers of using Client-side validation is that the users could create a page of their own that does not perform validation. When they submit this to the server, it could provide invalid values. To prevent this, the validation controls that are supplied with ASP.NET automatically perform validation of posted values on the server, even if the original validation is done on the client.

Validation can be managed by a Page directive. This overrides the automatic client-side validation process. The directive:

```
<%@ Page ClientTarget="DownLevel"%>
```

forces the validation controls to only do validation on the server, while the directive:

```
<%@ Page ClientTarget="UpLevel"%>
```

forces the controls to use Client-side validation as well as the Server-side validation of posted values.

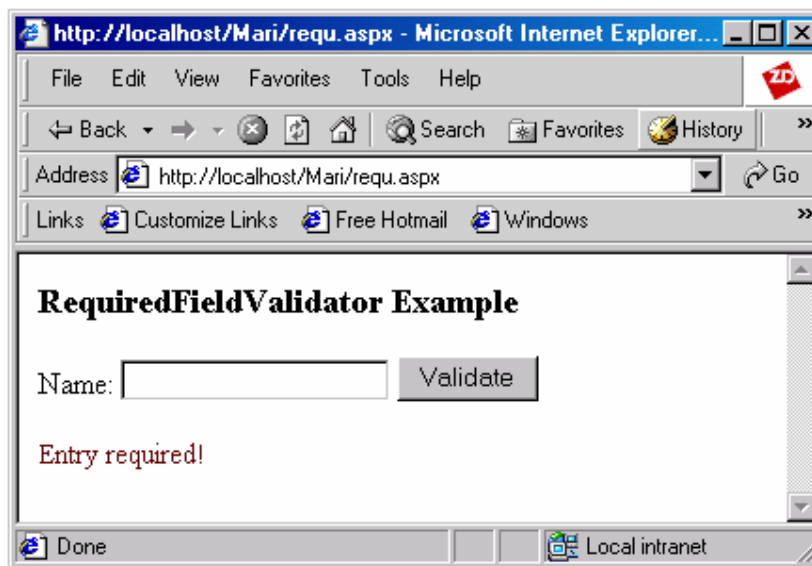
**Note :** "UpLevel" directive will force the client to attempt to use Client-side script for the validation - even if it doesn't support this technique.



### Practice 5.1

Here is an example using RequiredField Validator, a name is required from the user as input.

```
<html>
<body>
<h3>RequiredFieldValidator Example</h3>
<form runat=server>
  Name: <asp:textbox id=text1 runat="server" />
  <asp:button id="Button1" runat="server" text="Validate" />
  <p><asp:requiredfieldvalidator id="RequiredFieldValidator1"
    controltovalidate="Text1"
    font-size="11pt"
    forecolor="#600" runat="server">
    Entry required!
  </asp:requiredfieldvalidator>
</form>
</body>
</html>
```



The RequiredFieldValidator checks if the user has made any entry. It ensures that the user is not allowed to move further without entering the field. So the attribute control to Validate is set to text1. When the

control shifts from `text1`, the `RequiredFieldValidator` checks if there is any entry for the field and if there is no entry it prints the error message "Entry Required!".

## 5.4 CompareValidator

The Compare Validator is used to compare the data entered by a user with a constant or other field on the web form. The most common use of this Validation control is with respect to the password field. Typically Web Forms would ask the user to enter a password and to confirm it. These two fields must be identical for the form to be processed. The Compare Validator would be used in this case.

### Display Property

The display property indicates how the validation controls output will be rendered in the page. The value "static" tells the control to reserve sufficient space in the page for the text content so that the page layout does not change when this content is displayed.

The value "dynamic" indicates that the control will only take up space in the page when the error text string is displayed. In this case, the layout of the page might change. However, this is useful when there are more than one validation control is attached to a control on the form. The value "none" tells the control that no inline output will be displayed, even if the input value is invalid. Instead, the `ValidationSummary` control can be used to detect if there was an error, and display the contents of the `errorMessage` property elsewhere in the page.



### Practice 5.2

Given below is a typical example of using the compare validator control. Two text controls are used to get the password from the user and if there is a difference in the password it is reported by the comparison control.

```
<HTML>
  <SCRIPT LANGUAGE="c#" RUNAT="server">
    void submit_Me(Object sender,EventArgs E)
    {
      if (Page.IsValid==true)
      {
        Result.Text="Successful Login.";
      }
    }
  </SCRIPT>

  <BODY bgcolor="wheat">
  <FORM RUNAT="server">
    <h4>Example for CompareValidator Control</h4>
    <br>
    <asp:label id=Result runat="server"/>    Enter your name :
    <INPUT TYPE="text" id="UserName" Runat="server" />
      <asp:RequiredFieldValidator id="regUserName"
        runat="server"
        ControlToValidate="UserName"
        errormessage="Please Enter User Name"
        display="static">
    </asp:RequiredFieldValidator>
```

```

<br><br>
Enter New Password :
<INPUT TYPE="text" id="Password" Runat="server" />

<asp:RequiredFieldValidator id="reqPassword"
    runat="server"
    ControlToValidate="Password"
    errormessage="Please Enter Password"
    display="static">
<--
</asp:RequiredFieldValidator>

<br><br>
Retype Password :
<INPUT TYPE="text" id="vPassword" Runat="server" />
<asp:RequiredFieldValidator id="reqvPassword"
    runat="server"
    ControlToValidate="vPassword"
    errormessage="Please Verify Password"
display="static">
<--
</asp:RequiredFieldValidator>

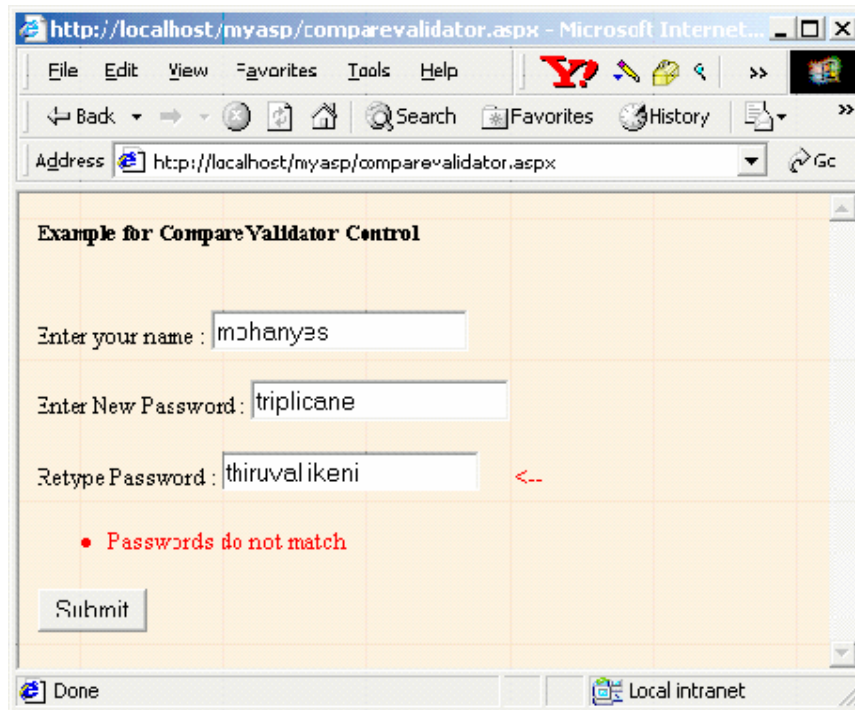
<asp:CompareValidator id="reqMatch"
    runat="server"
    ControlToValidate="Password"
    errormessage= "Passwords do not match"
    ControlToCompare = "vPassword"
    type = "string"
    operator = "Equal"
display="static">
<--
</asp:CompareValidator>

<asp:ValidationSummary id="sumErrors"
    runat="server"
    showSummary = true
displayMode = "BulletList" />
<asp:button text="Submit" RUNAT="server" onClick="submit_Me" />
</FORM>
</BODY>
</HTML>

```







In the above program, three text boxes are created for Name, New Password and Retype password. The New password is for entering the new password and the retype password is used to check if the password has been typed without typographical errors. Normally, the password will be masked and it will not be visible. Here the password has not been masked to enable us to view the text.

The RequiredFieldValidator is used to check if the username, password and vpassword have any entry. If the user leaves them blank it shows the user that some entry is required for the field. The CompareValidator which checks if both the password and vpassword entries are identical. If they are not identical, the error message "Passwords do not match is displayed" otherwise "Successful Login" is displayed.

## 5.5 RangeValidator

The Range Validator checks if the data that a user enters falls within a given range of values. The minimum and maximum values can be set to constant values or they can be set to other fields on the web form.

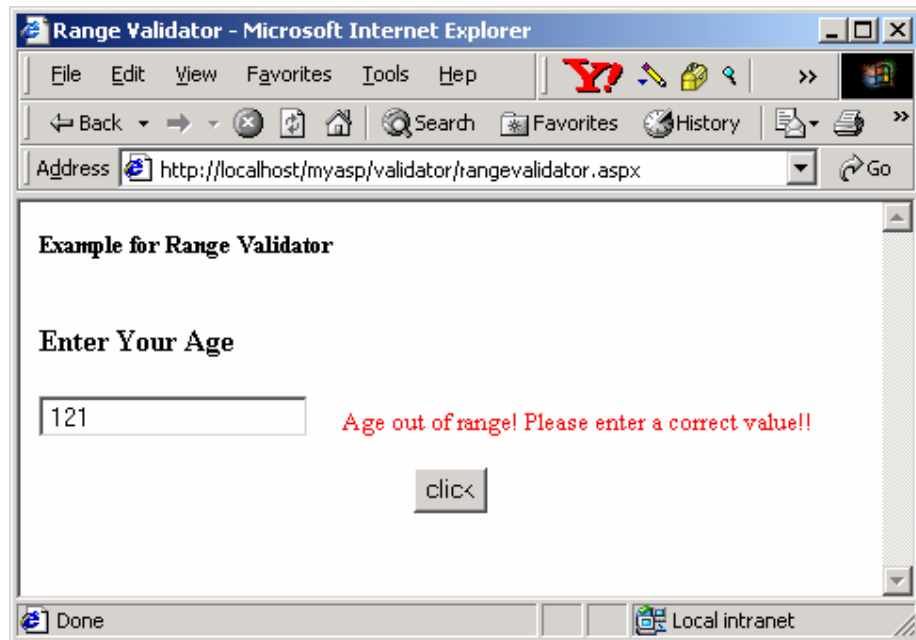


### Practice 5.3

Now let us see an example using RangeValidator which checks whether the given input is within the specified range.

```
<html>
<script language="c#" runat="server">
void btnClick(object sender,EventArgs e)
{
    if (Page.IsValid==true)
```





## 5.6 RegularExpressionValidator

The RegularExpression Validator evaluates the data that a user enters against a regular expression. This allows for complex formatting restrictions on the field data. A popular example of this kind is the email address. A RegularExpression Validator can check if there is at least one character, followed by the '@' symbol, followed by at least one character, followed by a period, which is followed by either 'com', 'org', 'net' or any other valid email extension. This enables the developer to force the user to enter a valid email address.

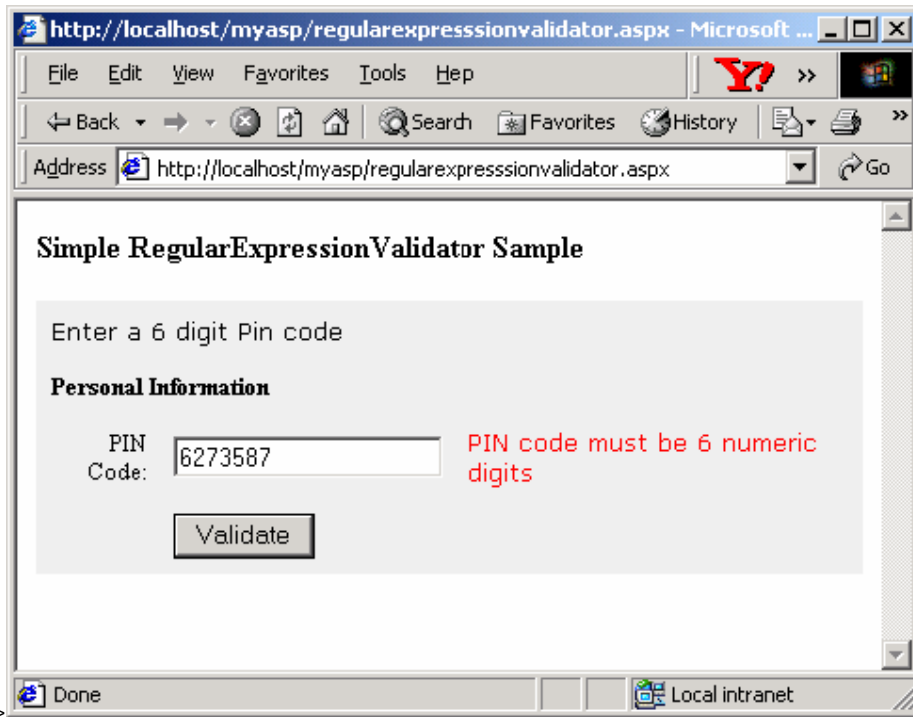


### Practice 5.4

Here is an example with RegularExpression validator which validates if a specific number of digits have been entered.

```
<html>
  <head>
    <script language="C#" runat=server>
      void ValidateBtn_Click(Object Src, EventArgs E)
      {
        if (Page.IsValid)
        {
          lblMessage.Text="Page is Valid!";
        }
      }
    </script>
  </head>
  <body>
    <div class="header"><h3>Simple RegularExpressionValidator Sample</h3></div>
```

```
<form runat="server">
<center>
<table bgcolor="#e0e0e0" cellpadding=6>
<tr valign="top">
    <td colspan=3> <asp:label id="lblMessage" text="Enter a 6 digit Pin code"
font-name="Verdana" font-size="10pt" runat="server" />
    </td>
</tr>
<tr>
    <td colspan=3><b>Personal Information</b></td></tr>
<tr>
    <td align=right>
        PIN Code:</font>
    </td>
    <td>
        <asp:textbox id="TextBox1" runat=server /></td>
    <td> <asp:regularexpressionvalidator id="RegularExpressionValidator1"
        runat="server"
        controlovalidate="TextBox1"
        validationexpression="^\d{6}$"
        display="Static"
        font-name="verdana"
        font-size="10pt">
            PIN code must be 6 numeric digits
        </asp:regularexpressionvalidator>
    </td>
</tr>
<tr>
    <td></td>
    <td>
        <asp:button text="Validate" OnClick="ValidateBtn_Click" runat=server />
    </td>
    <td></td></tr>
</table>
</center>
</form>
</body>
```



</html>

In the above example, the text box control `textbox1` is used to get the PIN Code from the user. `RegularExpressionValidator` is used to check the validity of the PIN Code. So the attribute control to validate is set to `textbox1`. The color of the error message is set to red so that the user can easily notice the error message. (When the control shifts from the textbox, the `RegularExpressionValidator` checks for the validity of the PIN Code and if there is an error prints the error message even before the "Validate" button is clicked. It checks if all the entries are numeric and that there are six significant digits. When the "validate" button is clicked, it checks for the validation of the page and if it is valid prints "Page is Valid!".

## 5.7 CustomValidator

The Custom Validator allows the developer to create user-defined criteria for validation. The Validation can be defined at two functions which they determine. The developer can place any logic he or she wants in these functions to determine if the data is acceptable. A common use of this Validator is to check for a username against the usernames in a database, to see if the person is a registered user. Credit Card number validation is also a very common use of the Custom Validator.



### Practice 5.5

Given below is a typical example of using the custom validator control. A text control is used to get an user input and if the input exceeds 9 digits the custom validator prints an error message.

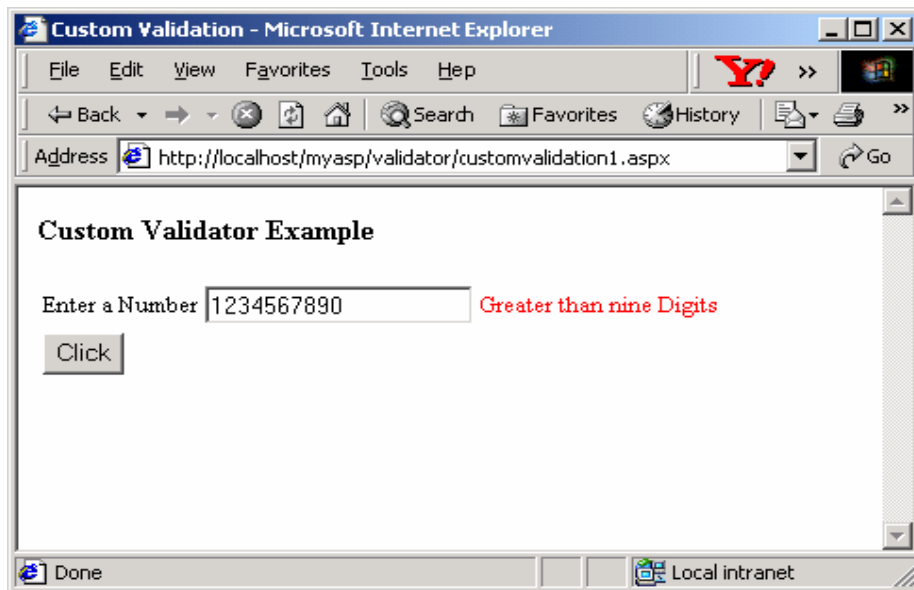
```
<html>
<title>Custom Validation</title>
<script language="c#" runat="server">
    bool ServerValidate(object sender,String value)
    {
        if (value.Length <= 9)
            return true;
        else
```

```

        return false;
    }
</script>
<body>
<h3>Custom Validator Example</h3>
<form runat=server>
<table>
<tr>
<td> <asp:label id="label1" text="Enter a Number" runat="server" /> </td>
<td> <asp:textbox id="text1" runat="server" /> </td>

<td> <asp:CustomValidator id="CustomValidator1" runat="server"
        ControlToValidate="text1"
        OnServerValidate="ServerValidate"
        Display="static">
        Greater than nine Digits
    </asp:customvalidator>
</td>
</tr>
<tr>
<td><asp:button id="bt1" text="Click" runat="server" /> </td>
<td> <asp:label id="label2" text=" " runat="server" /> </td>
<td> <asp:label id="label3" text=" " runat="server" /> </td>
</tr>
</table>
</form>
</body>
</html>

```

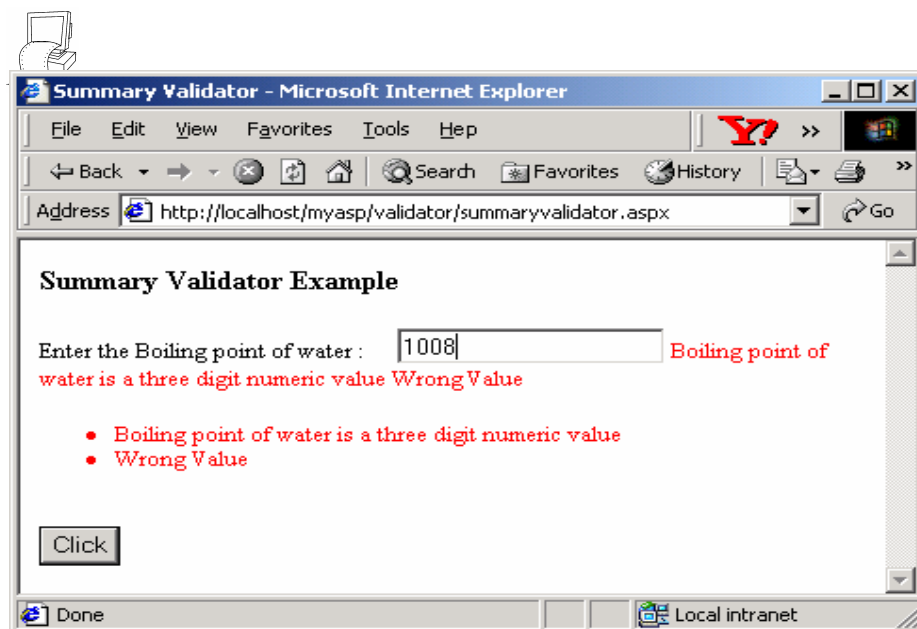




```

</asp:regularexpressionvalidator>
<asp:comparevalidator id="compboilpt"
  controltovalidate="text1"
  Type="Integer"
  ValueToCompare=100
  Operator="Equal"
  display="static"
  ErrorMessage="Wrong Value"
  runat="server">
</asp:comparevalidator>
<asp:requiredfieldvalidator
  id="RequiredFieldValidator1"
  controltovalidate="Text1"
  runat="server"
  ErrorMessage = "Should Enter a Value">
</asp:requiredfieldvalidator>
<asp:ValidationSummary
  id="sumErrors"
  runat="server"
  showSummary = true
  displayMode = "BulletList">
</asp:ValidationSummary >
<br>
<asp:button text="Click" OnClick="ValidateBtn_Click" runat=server />
</form>
</body>
</html>

```





In the above example, the text box control text1 is used to get the boiling point of water. RegularExpressionValidator, CompareValidator, RequiredFieldValidator are used to check the validity of the figure entered. So the attribute controltovalidate is set to text1 for all the validators. When the control shifts from text1, the RegularExpressionValidator checks for the validity of the number entered for the boiling point of water and if there is an error, prints the error message even before the “click” button is clicked. The RegularExpressionValidator checks for the number of digits that are entered, if it is less than or greater than 3 an error message is displayed. The ComparisonValidator checks whether the entry is exactly equal to 100. The RequiredFieldValidator checks if there is an entry in the text box. The ValidationSummary control lists all the errors as bulleted list.

### 5.9 Short Summary

- Validator controls can be used to validate input on any Web Form
- More than one control can be used on a given input field
- Client-side validation may be used in addition to server-validation to improve form usability
- The CustomValidator control lets the user define custom validation criteria

### 5.10 Brain Storm

1. What is client-side validation?
2. What are the validation controls available in ASP.NET?
3. Which control can be used to validate against a data type?
4. How is pattern-matching validation done through RegularExpression Validator?
5. What is the advantage of ValidationSummary control?
6. What is the use of TabIndex?
7. How are user inputs validated?
8. What are the different ways in which error messages can be displayed?
9. What are the different types of Validators available in ASP.NET?
10. What is the purpose of the ValidationSummary control?

## Lecture 6

---

# HttpRequest

---

### Objectives

In this lecture you will learn the following

- + About HTML Forms
- + Learning HttpRequest Properties
- + Learning about Cookies
- + How the Query String is used?

---

## Coverage Plan

---

### Lecture 6

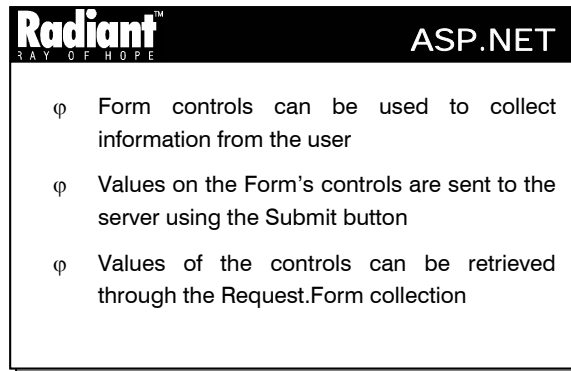
- 6.1 Snap Shot
- 6.2 HTML Form
- 6.3 HhttpRequest Properties
- 6.4 Cookies
- 6.5 Short Summary
- 6.6 Brain Storm

## 6.1 Snap Shot

This session aims at introducing ways to retrieve data from the HTML form and then focusses on the various properties and methods of the HttpRequest class.

The Request object is responsible for retrieving information from the Web browser. The Request object is filled with various types of collection, properties, and methods that provide many ways to retrieve information from the user. The HttpRequest class is used to gain access to the HTTP request data elements that are supplied by a client.

## 6.2 HTML Form



To interact with the user, HTML recognizes a few special tags that insert controls on a Form, besides the universal hyperlinks. A control can be used to collect information from the user for registration purpose, take orders over the Internet, or let the user specify selection criteria for record retrieval from database. Before placing any controls on a page, a Form must be created, with the Form tag. All controls must appear within a pair of Form tags:

```
<FORM NAME = "myForm">
  Controls go here
</FORM>
```

The NAME attribute is optional, but it is advisable to have Form names. The FORM tag also accepts two more attributes namely **METHOD** and **ACTION** that determine how the data will be submitted to the server and processed. The **METHOD** attribute can have the value **POST** or **GET**. The **ACTION** attribute specifies the script that will process the data on the server.

The values entered by the user on the Form's controls are sent to the server using the Submit button. Every Form has a **Submit** button that extracts the **ACTION** attribute from the <FORM> tag. The values of the controls create a new URL and send it to the server.

The values of the various controls can be retrieved through the Request. Form collection. The Form collection has one member for each control on the Form and individual control's value can be accessed by name. For example, if the Form contains a control named "UserValue", its value can be accessed with the expression **Request.Form ("UserValue")**.

### Retrieving Form Data

Whenever a user submits an HTML form, all the form fields and their values are placed in the Form collection of the Request object. This session explains how to retrieve the values of the form fields.

**Note :** The Form collection contains the values of form elements when an HTML form is submitted with the POST method. When an HTML Form is submitted with the GET method, the values of the form elements are placed in the **QueryString** collection.



## Practice 6.1

The following code accepts details from the user and submits it to the .aspx file.

### html file

```
<html>
<head><title>Simple HTML Form</title></head>
<body>
<form method="post" action="result.aspx">
  <b>Enter name: </b>
    <input name="username" type="text" size=30>

  <p>
  <b>Enter Comments: </b>
  <br><textarea name= "comments" cols=40 rows=5></textarea>
  <p>

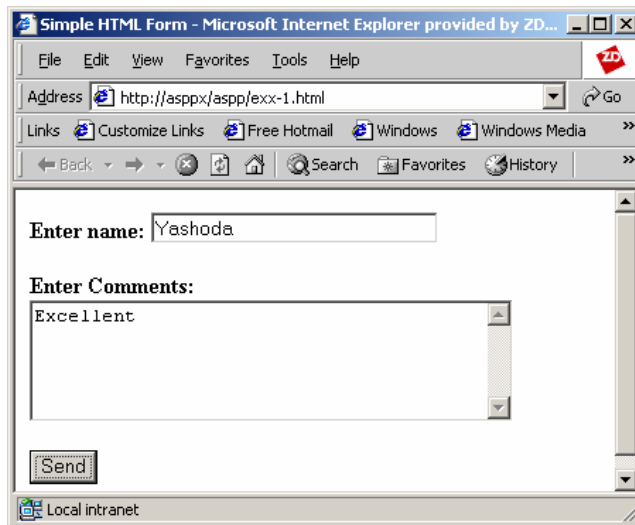
  <input type="submit" value="Send">
</form>
</body>
</html>
```

### result.aspx

```
<%@Page Language="C#"%>
<%
  string username="";
  string comments="";
  username = Request.Form ["username"];
  comments= Request.Form ["comments"];
%>

<head><title>Result</title></head>
<body>
  User Name : <%=username%>
  <p>
  Comments : <%=comments%>
</body>
```





In the above program the user is prompted to enter a user name and comments. When the user clicks the button labeled **Send**, the contents of the form are submitted to result.aspx. The **Form** collection can be accessed using the **Request** object inside the result.aspx file is then passed to the browser.

When the Send button is clicked after entering the name (Yashoda) and comments (Excellent), the result.aspx file is called which displays the output as shown below:

```
User Name : Yashoda
Comments : Excellent
```

In certain situations it is necessary to display a second form where the user can approve the information entered into the first form. In such a case the user is provided with another opportunity to change the information before the information is actually entered into a database. The following program (confirm.aspx) redisplay the information posted in a previous HTML form with a new form.



### Practice 6.2

The following program displays a confirmation form after the submitted details are processed.

#### html file

```
<html>
<head><title>Simple HTML Form</title></head>
<body>
<form method="post" action="confirm.aspx">
  <b>Enter name: </b>
    <input name="username" type="text" size=30>

  <p>
  <b>Enter Comments: </b>
  <br><textarea name= "comments" cols=40 rows=5></textarea>
  <p>

  <input type="submit" value="Send">
</form>
</body>
</html>
```

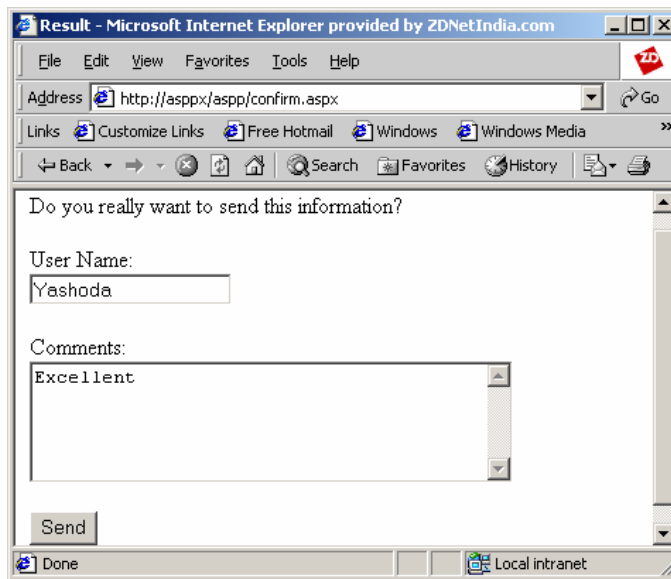
#### confirm.aspx

```
<%@Page Language="C#"%>
<%
    string username="";
    string comments="";
    username = Request.Form ["username"];
    comments = Request.Form ["comments"];
%>

<head><title>Result</title></head>
<body>

<form method= "post" action= "result.aspx">
    <p>Do you really want to send this information?</p>
    <p>User Name:
    <br><input name= "username"
        type= "text" value= "<%= (username) %>">

    <p>Comments:
    <br><textarea name= "comments" cols=40 rows=5>
    <%= (comments) %></textarea>
    <p>
    <input type="submit" value="Send">
</form>
</body>
```



When the Send button is clicked after entering the details, the **confirm.aspx** file is invoked and the confirmation form is displayed. The user can make further changes and submit the form by clicking the Send button.

## 6.3 HttpRequest Properties

**Radiant™** **ASP.NET**  
RAY OF HOPE

Some of the properties of the HttpRequest class are

- φ Application
- φ Browser
- φ ClientCertificate
- φ ContentType
- φ FilePath
- φ RawUrl
- φ Url
- φ Cookies

### Application Path

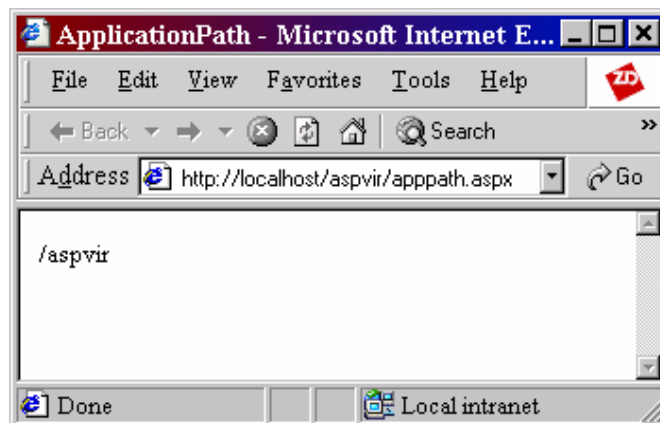
This property gets the virtual path to the currently executing server application.



#### Practice 6.3

The following example illustrates the usage of the ApplicationPath property.

```
<head>
<title> ApplicationPath </title>
<script language="C#" runat="server">
void Page_Load(Object Source, EventArgs e)
{
    String apath;
    apath=Request.ApplicationPath;
    Info.InnerHtml=apath;
}
</script>
</head>
<form runat="server">
    <span id="Info" runat="server"/>
</form>
```





In the above example, it can be seen that the property **ApplicationPath** is used to get the virtual path. As in the case of the above example the virtual directory is **aspvir**.

### Browser

The Browser property gets the virtual path of the currently executing server application.

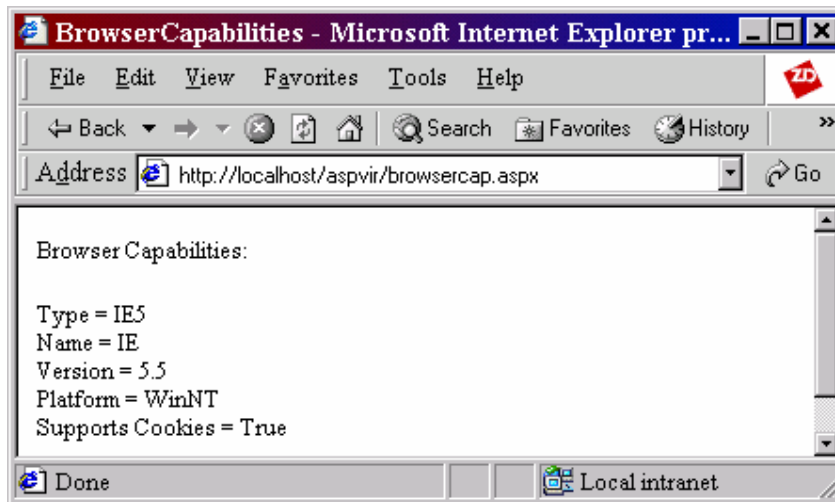


#### Practice 6.4

The following example illustrates the usage of the Browser property.

```
<head>
<script language="C#" runat="server">
void Page_Load(Object Source, EventArgs e)
{
    HttpBrowserCapabilities bcap = Request.Browser;

    Response.Write("<p>Browser Capabilities:</p>");
    Response.Write("Type = " + bcap.Type + "<br>");
    Response.Write("Name = " + bcap.Browser + "<br>");
    Response.Write("Version = " + bcap.Version + "<br>");
    Response.Write("Platform = " + bcap.Platform + "<br>");
    Response.Write("Supports Cookies = " + bcap.Cookies + "<br>");
}
</script>
/head>
```



This Browser property is used here to get the browser details namely, the browser type, name, version and the platform on which the browser is running and whether the browser supports cookies.

### ClientCertificate

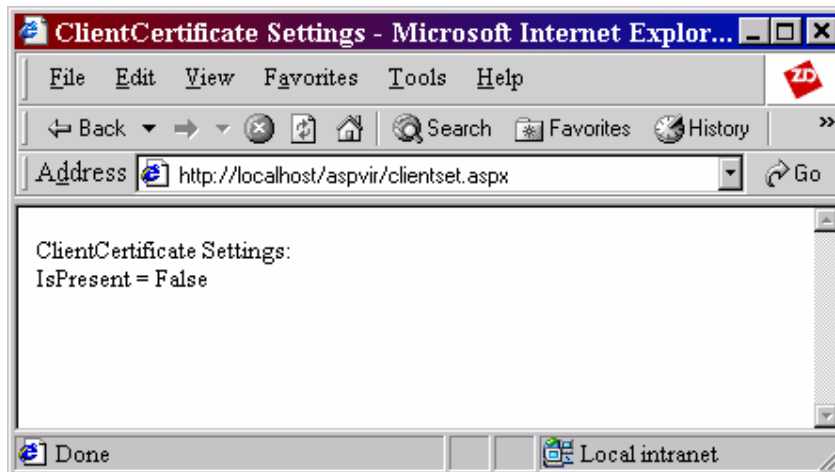
This property gets information on the current request's client security certificate.



### Practice 6.5

The following example illustrates the usage of the ClientCertificate property.

```
<head>
<script language="C#" runat="server">
void Page_Load(Object Source, EventArgs e)
{
    HttpClientCertificate cert = Request.ClientCertificate;
    Response.Write("ClientCertificate Settings:<br>");
    Response.Write("IsPresent = " + cert.IsPresent + "<br>");
}
</script>
</head>
<form runat="server">
</form>
</head>
```



As it can be seen from the above example, the information about the client certificate settings details can be found using the ClientCertificate property.

### ContentType

This property indicates the Multipurpose Internet Mail Extensions (**MIME**) content type of the incoming request. This property is read-only.

```
contype = Request.ContentType;
Response.Write (contype);
```

### FilePath

This property indicates the virtual path of the current request. It is read-only.

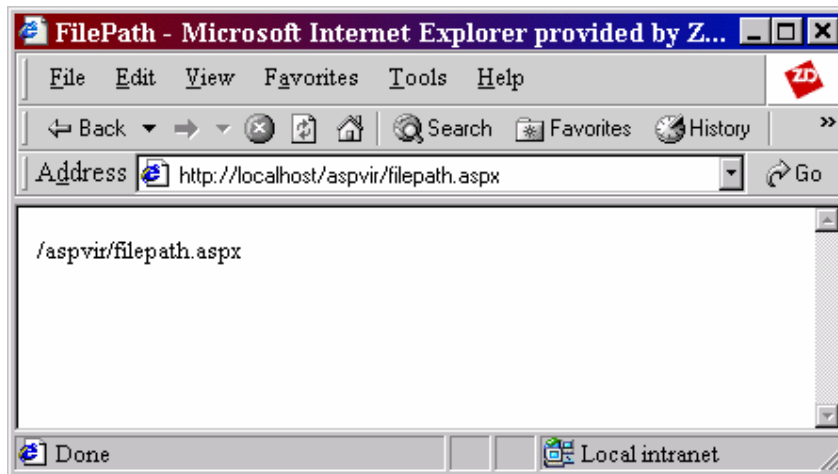


## Practice 6.6

The following example illustrates the usage of the Filepath property.

```
<head>
<title> FilePath</title>

<script language="C#" runat="server">
void Page_Load(Object Source, EventArgs e)
{
    String fiPath;
    fiPath = Request.FilePath;
    Message.InnerHtml=fiPath;
}
</script>
</head>
<form runat="server">
    <span id="Message" runat="server"/>
</form>
```



The above example shows that aspvir is the virtual path.

### RawUrl

This property indicates the raw URL of the current request.



## Practice 6.7

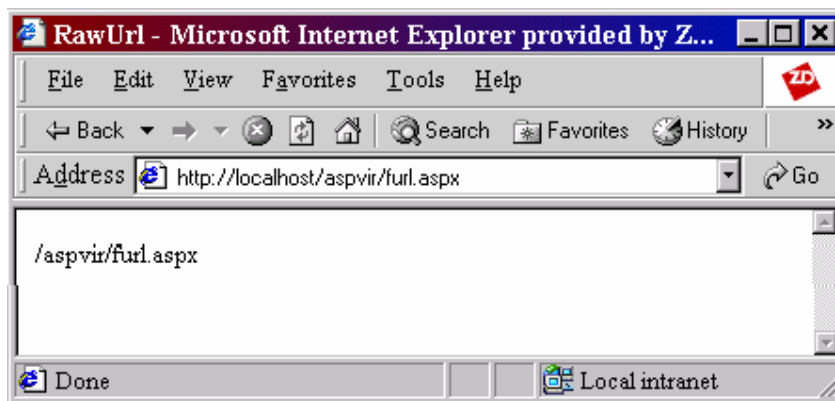
The following program illustrates the usage of the RawUrl property.

```
<head>
<title> RawUrl </title>
<script language="C#" runat="server">
void Page_Load(Object Source, EventArgs e)
```

```

{
    String Rurl;
        Rurl = Request.RawUrl;
    Info.InnerHtml=Rurl;
}
</script>
</head>
<form runat="server">
    <span id="Info" runat="server"/>
</form>

```



The Raw URL of the above request is `aspvir/furl.aspx` as shown in the above output. It is obvious that **aspvir** is the virtual directory and **furl** is the file name.

### Url

This property gets information regarding the URL of current request.



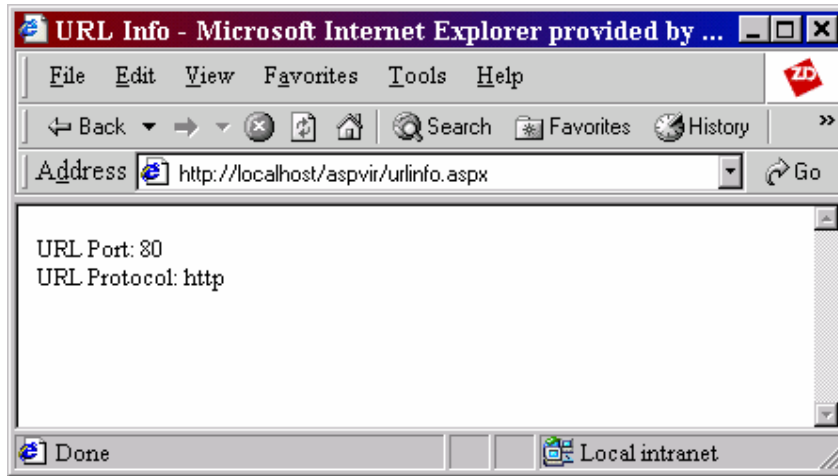
### Practice 6.8

The following example gives information about the URL of the current request.

```

<head>
<title> URL Info </title>
<script language="C#" runat="server">
void Page_Load(Object Source, EventArgs e)
{
    HttpUrl objUrl = Request.Url;
        Response.Write("URL Port: " + objUrl.Port + "<br>");
        Response.Write("URL Protocol: " + objUrl.Protocol + "<br>");
}
</script>
</head>
<form runat="server">
</form>

```



The above example shows that by using the Url property, the port and protocol information can be obtained.

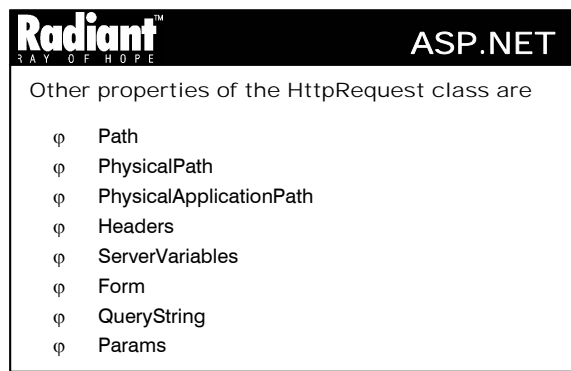
## 6.4 Cookies

Cookies are small pieces of information, which can be sent to a browser by a server program and stored by the web browser. The web browser will then pass the cookie back to the server every time it makes a request from that server. This facility is particularly useful for allowing authentication.

For example, when a user logs into a password restricted system, a cookie containing that users login/password details can be set with those details, so that the user does not have to re-type their password for every new page they wish to download.

It is important to know that the client's web browser could be configured to refuse cookies and that some browsers manage cookies incorrectly. When a cookie was sent, the server can retrieve it in a successive client's connection. This strategy makes it possible to track the history of client's connections.

The cookie property gets a collection of the client's cookie variables. Cookies can be seen elaborately in the following session.



### Path

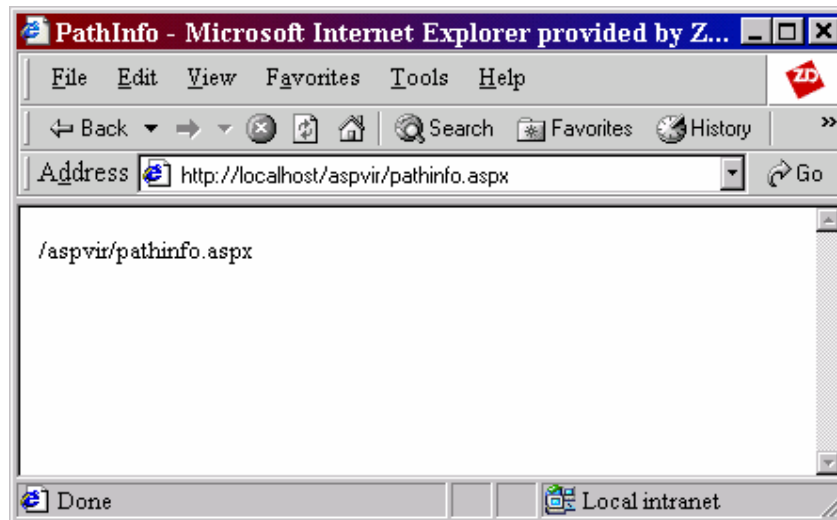
This property indicates the virtual path of the current request and it is read-only.



### Practice 6.9

The following example illustrates the usage of the Path property.

```
<head>
<title> PathInfo </title>
<script language="C#" runat="server">
void Page_Load(Object Source, EventArgs e)
{
    String vpath;
    vpath = Request.Path;
    Info.InnerHtml=vpath;
}
</script>
</head>
<form runat="server">
    <span id="Info" runat="server"/>
</form>
```



As shown from the above example, the Path property is used to get the virtual path of the current request.

### ServerVariables

This property gets a collection of Web server variables.



### Practice 6.10

The example shows how the ServerVariables are used.

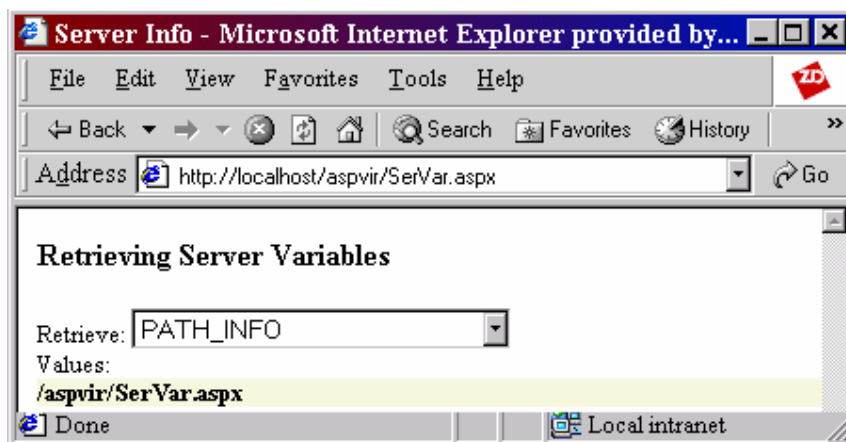
```
<script runat="server" language="c#">
```

```

<title> server Info </title>
void Page_Load(Object Sender, EventArgs E){
    NameValueCollection NVCSrvElements = Request.ServerVariables;
    if (! IsPostBack){
        Names.DataSource=NVCSrvElements;
        Names.DataBind();
    }else{
        ValuesHead.Text = "Values: ";
        if (NVCSrvElements.Get(Names.SelectedItem.Text).ToString() == null) {
            Values.Text = "null";
        } else {
            Values.Text = NVCSrvElements.Get(
                Names.SelectedItem.Text).ToString() ;
        }
    }
}
}
</script>

<html>
<body>
<form runat="server">
    <h3>Retrieving Server Variables</h3>
    Retrieve: <asp:DropDownList id="Names" runat="server" AutoPostBack="true"/>
    <br>
    <asp:Label id="ValuesHead" runat="server" />
    <br>
    <b>
    <asp:Label id="Values" runat="server" BackColor="Beige" Width="500" />
    </b>
</form>
</body>
</html>

```



As it can be seen from the output the various server variables information can be obtained by selecting the required property in the list box.

### Form

This property gets a collection of the Form variables.



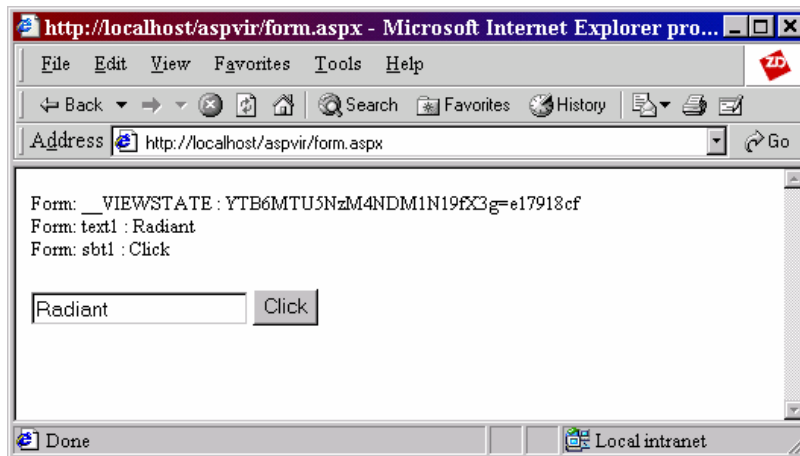
### Practice 6.11

The example illustrates the use of Form property to retrieve Form variables.

```
<html>
<title> Form Info </title>
<head>
<script language="C#" runat="server">
void Page_Load(Object Source, EventArgs e)
{
    if(IsPostBack){
        int ctrl1;
        NameValueCollection nvc;
        nvc=Request.Form; // Load Form variables into NameValueCollection
        variable.
        String[] arr1 = nvc.AllKeys; // Get names of all forms into a string
        array.
        for (ctrl1 = 0; ctrl1 < arr1.Length; ctrl1++) {
            Response.Write("Form: " + arr1[ctrl1] + " : " +
                nvc.Get(arr1[ctrl1])+ "<br>");
        }
    }
}
</script>
</head>
<body>
<form runat="server">
    <input type="text" id="text1" runat="server">
    <input type="submit" id="sbt1" value="Click" runat="server">
</form>
</body>
</html>
```







The above code first loads all the form variables into the NameValueCollection variable. Then using a for loop the contents of the collection are displayed in the browser.

### QueryString

This property gets the collection of QueryString variables.

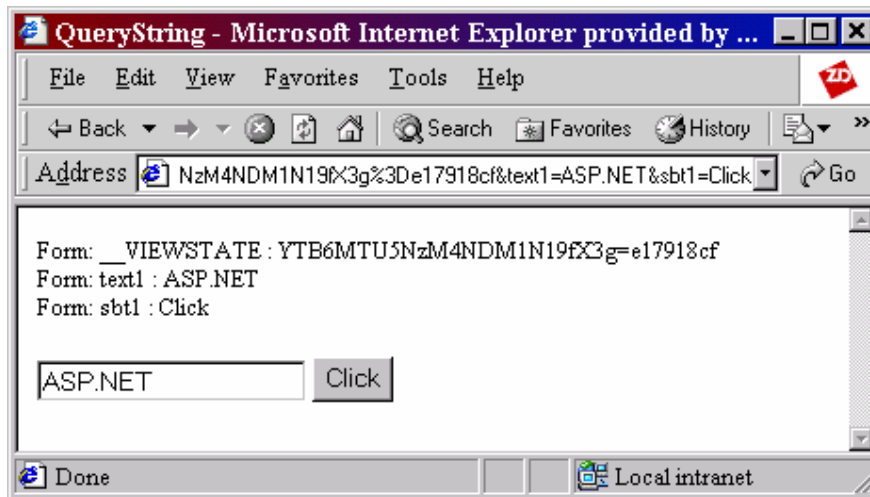


#### Practice 6.12

The example uses QueryString to retrieve form variables.

```
<html>
<title> QueryString </title>
<head>
<script language="C#" runat="server">
void Page_Load(Object Source, EventArgs e)
{
    if(IsPostBack){
        int ctrl;
        NameValueCollection nvc;
        nvc=Request.QueryString; // Load Form variables into
                                NameValueCollection variable.
String[] ary1 = nvc.AllKeys; // Get names of all forms into a string array.
        for (ctrl = 0; ctrl < ary1.Length; ctrl++) {
            Response.Write("Form: " + ary1[ctrl] + " : " +
                nvc.Get(ary1[ctrl])+ "<br>");
        }
    }
}
</script>
</head>
<body>
<form runat="server" method="Get">
    <input type="text" id="text1" runat="server">
    <input type="submit" id="sbt1" value="Click" runat="server">
</form>
</body>
```

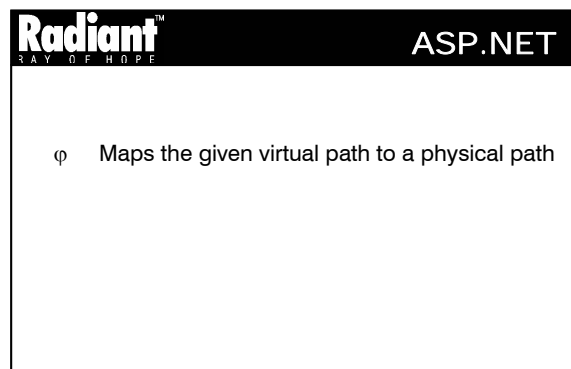




In the above code the variables of the form are retrieved using the Request.QueryString property. The values are then displayed in the browser.

## HttpRequest Method

### MapPath Method



#### Syntax

```
public string MapPath(
    string virtualPath
);
```

- **virtualPath** indicates the virtual path for the current request

## 6.5 Short Summary

- HttpRequest class is used to gain access to the HTTP request data elements supplied by a client

- The Browser property gets the virtual path of the currently executing server application
- ClientCertificate property gets information on the current request's client security certificate
- FilePath property indicates the virtual path of the current request. This property is read-only
- RawUrl property indicates the raw URL of the current request
- Form property gets a collection of Form variables
- QueryString property gets the collection of QueryString variables
- Params property gets a combined collection of QueryString + Form + ServerVariable + Cookies

### 6.6 Brain Storm

1. What is the use of HttpRequest?
2. What is the difference between the properties RawUrl and Url?
3. What are cookies?
4. What are the properties that are used to read the Form data?

❧

## Lecture 7

---

# HttpResponse

---

### **Objectives**

In this lecture you will learn the following

- + Knowing about the HttpResponse Properties
- + Knowing about the Redirecting

---

## Coverage Plan

---

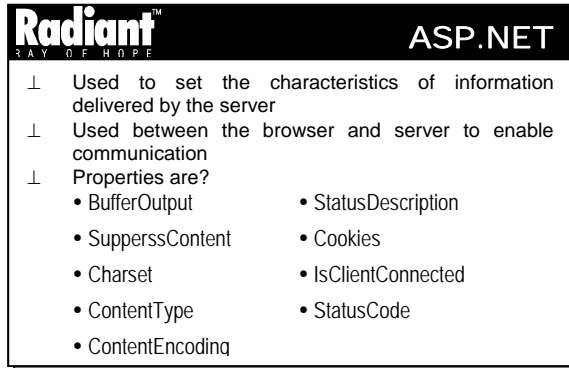
Lecture 7
7.1 Snap Shot
7.2 HttpResponseMessage Properties
7.3 HttpResponseMessage Methods
7.4 Short Summary
7.5 Brain Storm

## 7.1 Snap Shot

This session explains the important and commonly used properties and methods of the HttpResponse.

The Response object provides access to all the responses sent to the user. The Response object is responsible for sending the output from the server to the client. The methods and properties of this object control the way in which information is sent from the Web server to a Web browser.

## 7.2 HttpResponse Properties



HttpResponse has different properties that are used to set the various characteristics of the information delivered by the server. The characteristics are used between the browser and server to provide a communication mechanism between the information provider and the information consumer. The properties include:

### BufferOutput

The BufferOutput gets or sets a value indicating whether the HTTP output is buffered or not. This property takes the value true if the output to client is buffered and false otherwise. The default value is true.

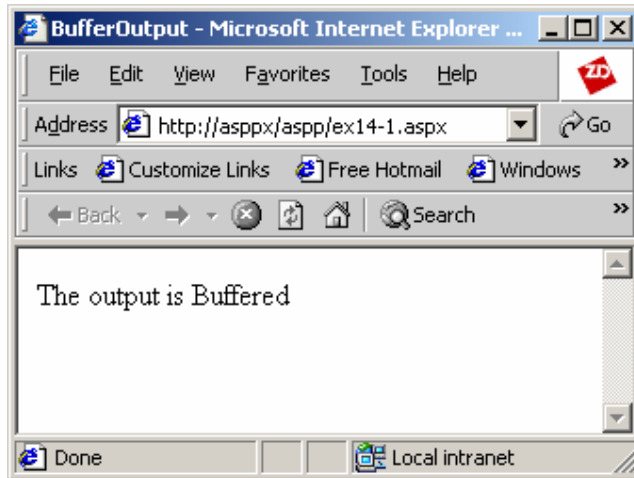


### Practice 7.1

The following program checks if the output is buffered.

```
<head>
<title>BufferOutput</title>
<script language="C#" runat="server">
void Page_Load(Object Source, EventArgs e)
{
    if (Response.BufferOutput == true)
        Message.InnerHtml="The output is Buffered";
}
</script>
</head>

<form runat="server">
    <span id="Message" runat="server"/>
</form>
```



The BufferOutput property in the above program checks if the output is buffered and displays a message in the browser.

### SuppressContent

The SuppressContent property gets or sets a value indicating that the HTTP content will not be sent to the client. The property returns true if the output is to be suppressed and false otherwise.

The following code removes the suppression of the content.

```
Response.SuppressContent = false;
```

### Charset

The Charset property gets or sets the HTTP charset of the output.

### Cookies

Cookies are used for session tracking, and user preferences. Cookies may also be used to determine how long the users view a Web page, content (e.g., advertising), the link, and other services. The cookies property gets the **HttpCookie** collection that is sent by the current request.

The following code takes the current cookie collection and fills a string array with the names of cookies.

```
HttpCookieCollection oReqCookies = Response.Cookies;  
String[] sReqCookies = oReqCookies.AllKeys;
```

### IsClientConnected

The IsClientConnected property gets a value that indicates whether the client is still connected to the server or not. The value of the property will be **true** if the client is currently connected and **false** otherwise.

### StatusCode

The `StatusCode` property gets or sets the HTTP status code of the output returned to the client. The return value is an integer code that represents the status of the HTTP output that is returned to the client. The default value is 200. Some `StatusCode` values and their corresponding meaning are given in Table 7.1

Code	Meaning
200	OK. The operation has been finished successfully
302	Redirection
401	Unauthorized access
403	Access forbidden
404	Requested resource is not found

Table 7.1

### 7.3 HttpServletResponse Methods

Radiant™

ASP.NET

RAY OF HOPE

- ⊥ Used to control information flow from the server to the browser
- ⊥ Used to control the HTTP header information, for server buffering and page redirection problems
- ⊥ Methods are:
  - AppendToLog
  - Write
  - Clear
  - End
  - Flush
  - Redirect

The methods are used to control the HTTP header information, non-textual content such as binary images, and Web server log files. In addition, the methods control the buffering mechanism happening in the server and page redirection.

#### AppendToLog

The `AppendToLog` method adds custom log information to the IIS log file.

#### Syntax

```
public void AppendToLog( string param );
```

where, `param` is a string that contains the text to be added to the log file.

The following code appends the text **Page delivered** to the IIS log file.

```
Response.AppendToLog("Page delivered");
```

#### Write

The `Write` method writes the values to an HTTP output content stream.

#### Syntax



```
public void Write( string s);
```

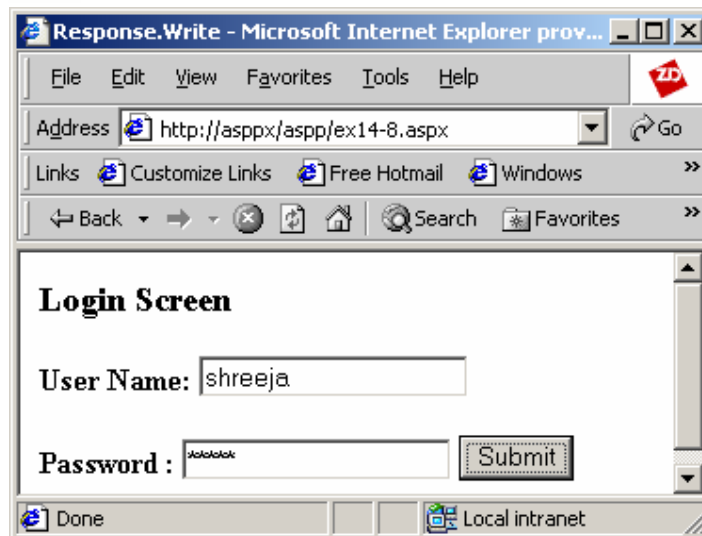
Here, s is the string that has to be written to the HTTP output.



## Practice 7.2

The following program adds two input textboxes and a submit button.

```
<head>
<title>Response.Write</title>
<script language="C#" runat="server">
void Page_Load(Object Source, EventArgs e)
{
    Response.Write("<H3>Login Screen</H3>");
    Response.Write("<b>User Name: </b>");
    Response.Write("<input type=text id=user /> "+"<br><br>");
    Response.Write("<b>Password : </b>");
    Response.Write("<input type=password id=user /> ");
    Response.Write("<input type=submit value=Submit />");
}
</script>
</head>
```



The above program uses the Response.Write method and displays two textboxes - to enter the user name and password. A Submit button is also displayed.

### Clear

The Clear method clears all headers and content output from the buffer stream.

### Syntax

```
public void Clear();
```

The following code clears the content output from the buffer stream.

```
Response.Clear();
```

### End

The End property sends all the currently buffered output to the client and then closes the socket connection.

### Syntax

```
public void End();
```

The following code closes the socket after sending the buffered output to the client.

```
Response.End();
```

### Flush

The Flush method sends the currently buffered output to the client. The method can be called multiple times during request processing.

### Syntax

```
public void Flush();
```

The following code sends the buffered output to the client.

```
Response.Flush();
```

### Redirect

The Redirect method redirects a client to a new URL.

### Syntax

```
public void Redirect( string url );
```

where, **url** is the new target location.

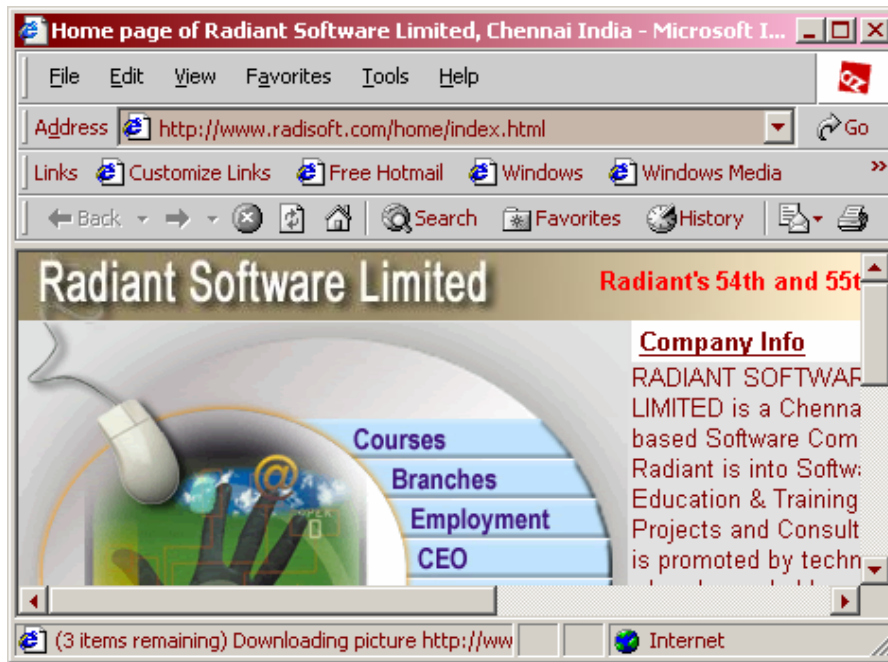


### Practice 7.3

The following program forces an unconditional redirection to another Web site.

```
<head>
<title>Response.Redirect</title>
<script language="C#" runat="server">
void Page_Load(Object Source, EventArgs e)
{
    Response.Redirect("http://www.radisoft.com");
}
</script>
</head>
```





As it can be seen from the above output, the program redirects the client to another Web Site called Radiant Software.

#### 7.4 Short Summary

- HttpResponseMessage provides access to the responses that are sent to the user
- The BufferOutput property gets or sets a value that indicates if the HTTP output is buffered
- The HTTP charset is set using the CharSet property
- The HTTP MIME type is set using the ContentType property
- Cookies are used for tracking the session. The cookies property is used to retrieve the HttpCookie collection that is sent by the current request
- The IsClientConnected property is used to check if the client is connected to the server
- The Write method writes values to an HTTP output content stream
- The End property closes the socket connection after sending the currently buffered output to the client

#### 7.5 Brain Storm

1. What is the use of HttpResponseMessage?
2. What is the use of the IIS log file?
3. How are session information maintained?

END

## Lecture 8

---

# Application, Session State Management and Cookies

---

## Objectives

In this lecture you will learn the following

- + Knowing about HttpApplication
- + Learning about Session
- + How to create a Cookie?
- + Learning about ServerObject

---

## Coverage Plan

---

<b>Lecture 8</b>
------------------

8.1 Snap Shot
---------------

8.2 HttpApplication
---------------------

8.3 Session
-------------

8.4 Cookies
-------------

8.5 Server Object
-------------------

8.6 Short Summary
-------------------

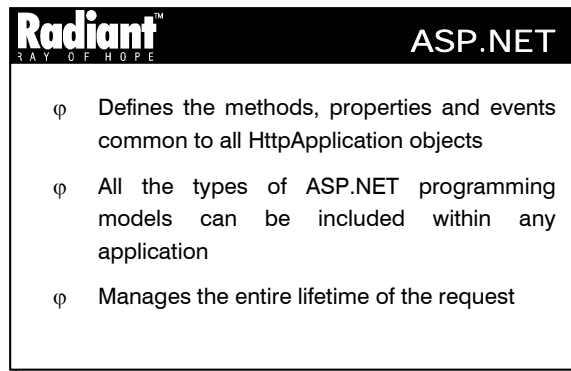
8.7 Brain Storm
-----------------

## 8.1 Snap Shot

This session deals with `HttpApplication`, `global.asax` file, and the methods to create an `Application` variable in `Global.asax` and accessing it in different applications. The session also discusses the properties and methods of an `ApplicationState`. Topics like `Session`, `SessionState` and cookies are also dealt with.

Active Server Pages can be thought in terms of traditional programming. A single Active Server Page is similar to a procedure or subroutine. When related Active Server Pages are grouped together they form an application. One of the greatest challenges faced in constructing a full-featured web application is keeping track of user-specific information when a user navigates the site. Sessions are used to store visitor's preferences and to keep track of the habits and interests of the visitors. This information can be used for advertising purposes, to improve the design of the Web site.

## 8.2 HttpApplication



The `HttpApplication` class defines the methods, properties and events common to all `HttpApplication` objects within the ASP.NET Framework.

An ASP.NET application is defined as a collection of files, pages, handlers, modules and executable code that can be invoked in the scope of a given virtual directory and its subdirectories on a Web Application Server. All the types of ASP.NET programming models can be included within any application (Web Forms, and Web Services). The only condition is that they must coexist in a single virtual directory structure.

Every ASP.NET application on the Web Server is executed within a unique .NET runtime Application Domain. ASP.NET maintains a collection of **`HttpApplication`** instances of a Web application's lifetime and automatically assigns an instance to process the incoming HTTP request. An **`HttpApplication`** instance manages the entire lifetime of the request, and is re-used only after completing the request.

### Creating an Application

An ASP.NET application is created by placing a simple `.aspx` page in the virtual directory and executing it in the browser. Then an appropriate code can be added to use the `Application` object, for example to store objects with application scope. Various event handlers can be defined by creating a `Global.asax` file (discussed later).

### Life Time of an Application

An ASP.NET application is created when a request is made for the first time to the server. On receiving the first request, a collection of **`HttpApplication`** instances are created, and the `Application_Start` event is

fired. An `HttpApplication` instance processes the request until the last instance is present. When there are no more instances the `Application_End` event is fired.

### Managing Applications

The **Application** object is a built-in object in ASP.NET. The `HttpApplication` is used to control and manage all the items that are available to all the users of an Active Server application. The **Application** items can be variables that are necessary for the application, or they can be instantiated objects providing special server-side functionality.

An application variable contains data that is used in all the web pages and by the users of an application. Application variables can contain different types of data, including arrays and objects. The common uses of application variables are:

- To display transient information on every Web page. For example, Tip of the day or a daily news update on every Web page
- To record the number of times a banner advertisement on the Web site has been clicked
- To store data that is retrieved from a database, which can then be displayed easily on multiple pages without frequently accessing the database
- To count the number of visitors to a particular Web site

An application variable helps in communicating with the users of the Web site. For example, application variables can be used to create multi-user games or multi-user chat rooms.

### Accessing Application Variable

To create a new application variable, the name of the new variable is passed to the **HttpApplication**.

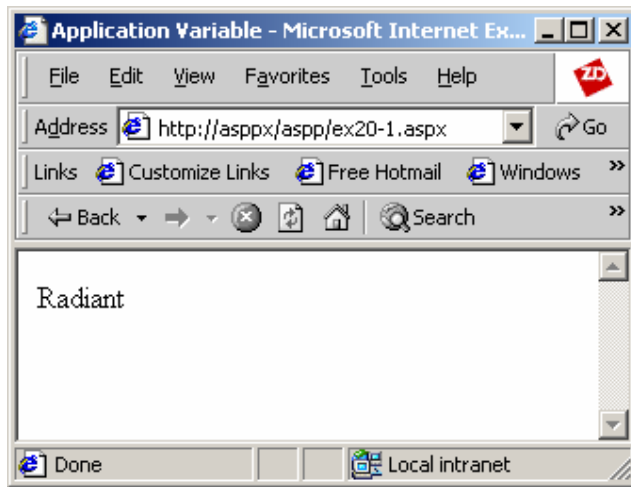


#### Practice 8.1

The following program illustrates the use of the application variable.

```
<HTML>
<HEAD><TITLE>Application Variable</TITLE></HEAD>
<BODY>
<script language="C#" runat="server">
void Page_Load(Object Source, EventArgs e)
{
Application ["var"] = "Radiant";
Response.Write (Application["var"].ToString());
}
</script>
</BODY>
</HTML>
```





The above example creates a new application variable named **var** and assigns the value **Radiant**. Then, the value of the variable **var** is displayed in the browser. After assigning a value to an application variable, it can be displayed on all the pages in an application.

### Global.asax

The Global.asax file (ASP.NET application file) is a file that contains the syntax for coding the server-side application object. The Global.asax file is stored in the root directory of an ASP.NET application. The file handles the application events like `Application_Start`, `Application_End`, `Session_Start`, `Session_End`, etc. The Global.asax is parsed and compiled by ASP.NET into a .NET Framework class when a request is made for the first time within its application namespace. The Global.asax file itself is configured so that direct URL request is automatically rejected. i.e. users are not allowed to view or download the code.

The ASP.NET Global.asax file is fully compatible with the ASP Global.asa file. A Global.asax file can be created either in a WYSIWIG HTML designer, Notepad, or as a compiled class that is deployed in the application's `\bin` directory as an assembly.

**Note:** The Global.asax file is not mandatory; if not defined, the ASP.NET framework assumes that there is no definition for application or session event handlers.

### Application Events in Global.asax

The Global.asax provides more than 15 events. The following table describes the different events in the Global.asax.

Event name	Description
<code>Application_Start</code>	Fired when the application is first started. This event is useful for applying application wide setting that are initialized only once.
<code>Session_Start</code>	Fired when a session is first started.
<code>Application_End</code>	Fired when the application ends (times out or reset). The event is fired only once and is used to clean the application code or global references created in the <code>Application_Start</code> event.
<code>Session_End</code>	Fired when a session ends (times out or reset). Fired only once.




Application_Error	Fired when an unhandled error occurs in the application. Errors are handled using error-trapping code (like Try/Catch blocks).
Application_BeginRequest	Fired whenever a new request is received.
Application_Authenticate	Signals that the request is ready to be authenticated. The event is used by the security module.
Application_Authorize	Signals that the request is ready to be authorized. The event is used by the security module. Used when code has to be run before the user is authorized for a resource.
Application_ResolveRequestCache	Used to short-circuit the processing of requests that are cached. The event is used by the output cache module.
Application_AcquireRequestState	Signals that per-request state (session or user) must be obtained.
Application_PreRequestHandlerExecute	Signals that the request handler is about to execute. The request handler will mostly be an ASP.NET page or a Web service.
Application_PostRequestHandlerExecute	Signals the first event available to the user after the handler has completed its work.
Application_ReleaseRequestState	Called when the request state must be stored, since the application is finished execution.
Application_UpdateRequestCache	Signals that the code processing is complete and the file is ready to be added to the ASP.NET cache.
Application_EndRequest	Signals the last event called when the application ends.
Application_PreRequestHeadersSent	Provides an opportunity to add, remove, or update headers and the response body.

Table 8.1

Events in the Global.asax can have different event method prototypes. The following is an example of how the BeginRequest event signature can vary:

```
public void Application_OnBeginRequest()
public void Application_BeginRequest(Object sender, EventArgs e)
```

### ApplicationState



**ASP.NET**

- φ Global information is shared across applications using the HTTPApplicationState class
- φ An instance can be manipulated through the Application property of the HttpContext object
- φ Object instantiated at the application-level scope can handle multiple concurrent access

Earlier versions of ASP had a single instance of the Application object for each application running in the machine. The values and object references are valid as long as the application is running. However, the

Application object is destroyed when changes are made to the Global.asax file or if the last client Session object is destroyed.

Global information can be shared across applications using the **HTTPApplicationState** class that exposes a key-value dictionary of objects that can be used to store both .NET Framework object instances and scalar values across multiple web requests from multiple clients.

An instance of the **HTTPApplicationState** class is created when the client requests a URL resource from within a particular ASP.NET Application virtual directory namespace for the first time. This is the case for each application that is stored in the computer. This instance can be manipulated through the Application property of the **HttpContext** object that is provided to all **IHTTPModules** and **IHTTPHandlers** during a given Web request.

The important thing to be noted is that objects instantiated at the application-level scope can handle multiple concurrent access. The limitation while using an HttpApplication is that it is not maintained across a **Web-farm** or in a **Web-garden**. A Web-farm is where more than a single server handles the requests for the same application and a Web-garden is where the same application runs in multiple processes within a single, multi-processor machine.

Application State supports provided by ASP.NET are:

- An easy to use state facility that is compatible with ASP that works with all .NET languages and is consistent with other .NET Framework APIs.
- The application state dictionary is available to all request handlers invoked within an application whereas in ASP it has restricted access only to pages.
- A simple synchronization mechanism that enables coordination of concurrent access to global variables stored in application-state easier.
- Application state values are accessible only from code running within the context of the originating application and not from other applications running on the system.

### Using Application State

Application state variables are also global variables for a given ASP.NET application. The impact of creating global variables must be considered before creating them. The points to be considered when using application state variables are:

- The memory occupied by application state variables will not be released until the value is either removed or replaced. For instance, storing 10Mb recordsets that is never used in application state permanently is not the best use of system resources
- The concurrency and synchronization implications of storing – and later correctly accessing – a global variable within a multi-threaded server environment. Multiple running threads within an application can access application state variables simultaneously. Care must be taken to ensure that either the application-scoped object is free-threaded and contains built-in synchronization support, or an application-scoped object is not free-threaded and explicit synchronization methods are coded to avoid deadlocks. Hence, explicit use the “Lock” and “Unlock” methods provided on the **HTTPApplicationState** class must be used to avoid problems
- Since locks that protect global resource are themselves global, the code running on multiple threads that accesses global resources will ultimately battle on these locks. This causes the OS to block the worker threads until the lock becomes available. In a server environment with high load this can

cause severe “thread thrashing” on the system. In multi-processor systems it can lead to processor under-utilization (since all the threads are waiting for a shared lock) and significant drops in overall scalability

- Global data stored in application-state is not stable, since it will be lost when the host containing it is destroyed. To avoid such failures, the state has to be stored either in a database or on some other durable store
- Since the application-state is not shared across either a Web-farm or a Web-garden, variables stored in application-state in these scenarios are global only to the particular process that the application is running

**Note:** For performance reasons all the built-in .NET collections do not contain built-in synchronization support.

### Properties

The properties of the application state are as follows:

#### All

The All property returns all application state objects as an array of objects.

#### AllKeys

The AllKeys property retrieves all application state object names in a collection.

#### Contents

The Contents property returns “this”. This property is included for compatibility with ASP.

#### Count

The count property gets the number of item objects in the application state collection.

#### Item

The item property is used to add, remove, or update an application state object. This property is the indexer for the **HttpApplicationState** class.

- **Item Property (String)** - Enables a user to add/remove/update a single application state object. This property is the indexer for the **HttpApplicationState** class.
- **Item Property (Int32)** - Enables user to retrieve an application state object by index. This property is the indexer for the **HttpApplicationState** class.

#### StaticObject

The StaticObject property exposes all objects declared through the <object runat=server></object> tag within the ASP.NET application file.

#### Collection

A **collection** represents a logical storage unit that helps to manage information.

#### Keys

The Keys collection represents a collection of the **System.String** keys of a collection. This class is used exclusively to implement the **NameObjectCollectionBase.Keys** property, which allows different methods

for accessing each key in the `NameObjectCollectionBase.Keys` property by using an index, an array or a `Get` method.

### Methods

The methods of the application state are discussed below:

#### Clear

The `Clear` method removes all objects from the application state collection.

#### Remove

The `Remove` method removes an object from the application state collection by specifying a name.

#### Syntax

```
public void Remove(string name);
```

- name is the object name

#### RemoveAll

The `RemoveAll` method removes all objects from the application state collection.

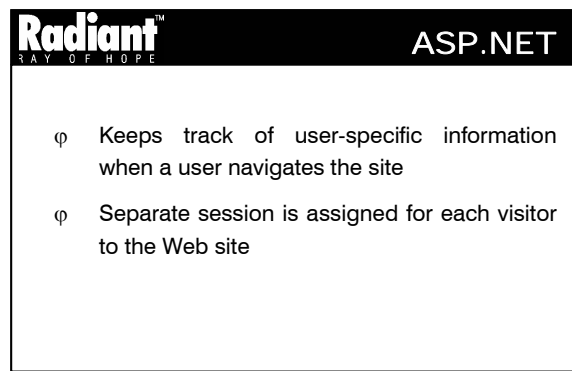
#### Lock

The `Lock` method locks access to all application state variables and hence facilitates access synchronization.

#### UnLock

The `Unock` method unlocks access to all application state variables. This also facilitates access synchronization.

## 8.3 Session



A session keeps track of user-specific information when a user navigates the site without asking the users identity for every request from the server. The different information to be maintained are a user's identification and security. In certain advanced applications, customization to the web site must also be maintained. A separate session is assigned for each visitor to the Web site.

### Usage of Session

A Session can store the user's preferences like the preferred background color of the Web page, whether to use frames etc. Sessions can also be used to create a virtual shopping basket. Whenever a user selects an item to buy, the item is automatically added to a shopping basket. When the user is ready to quit, he/she can purchase the items in the shopping basket at once. All the item information in the shopping basket can be stored in a session. Sessions can also be used to keep track of the habits and interests of the visitors. This information can be used for advertising purposes, to improve the design of the Web site, or to satisfy the Webmasters' curiosity.

### Life Time of an Session

A session starts when a user requests a page from a Web site. A session is closed when the user leaves the Web site.

### Managing Sessions

**Session variables** are similar to **application variables** and can be used in multiple ASP.NET pages. However, unlike an application variable, separate copies of a **session variable** are created for each visitor to the Web site.

In order to store data that will persist throughout a user session, the data has to be stored in a collection of the **Session object**. This is illustrated in the following example:



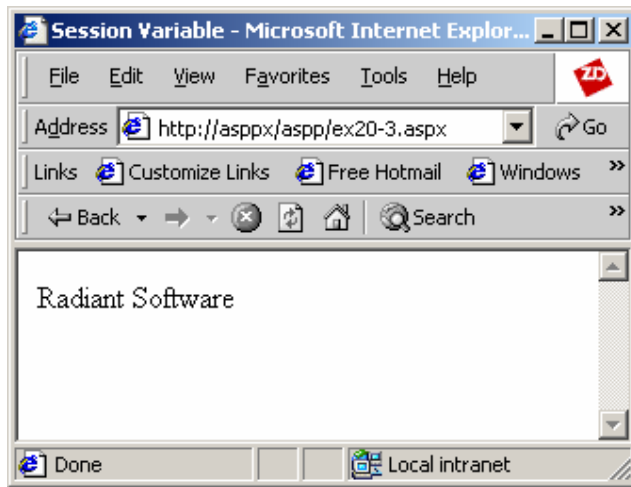
---

### Practice 8.2

The following program uses a session variable.

```
<HTML>
<HEAD><TITLE>Session Variable</TITLE></HEAD>
<BODY>
<script language="C#" runat="server">
void Page_Load(Object Source, EventArgs e)
{
    Session ["svar"] = "Radiant Software";
    Response.Write (Session["svar"].ToString());
}
</script>
</BODY>
</HTML>
```





The first line in this script assigns the text **Radiant Software** to a session variable named **svar**. The next line in the script displays the text on the screen.

If the same user requests another page the same text is displayed again. For example, consider the following code:

```
<script language="C#" runat="server">
void Page_Load(Object Source, EventArgs e)
{
Response.Write (Session["svar"].ToString());
}
</script>
```

In the above code the session variable was not assigned a value in this code. The session variable **svar** has retained the value it was assigned on the previous page. This cannot be accomplished using a normal script variable. The lifetime of a normal variable extends only throughout a single page. A session variable, on the other hand, persists until the user leaves the Web site.

**Note:** Session variables exist only in relation to a particular user. The values assigned to a session variable in one user session does not affect the values of the session variables in another user session; i.e. the data stored in session variables are not shared among different users.

### Session events in Global.asax

The session handles two events:

- **Session\_Start** - Triggered when a session begins
- **Session\_End** - Triggered when a session ends

To create a script that executes whenever a new session begins, the script has to be added to the **Session\_Start** section of the Global.asax file, as in the example given below:

```
<script language="C#" runat="server">
void Session_Start()
{
Session["UserName"] = "Unknown";
Session["UserPassword"] = "Unknown";
}
</script>
```

```
</script>
```

The script assigns the value “Unknown” to two session variables named **Username** and **UserPassword**. The **Session\_Start** script can be used for other purposes like redirecting the visitors to a new page.

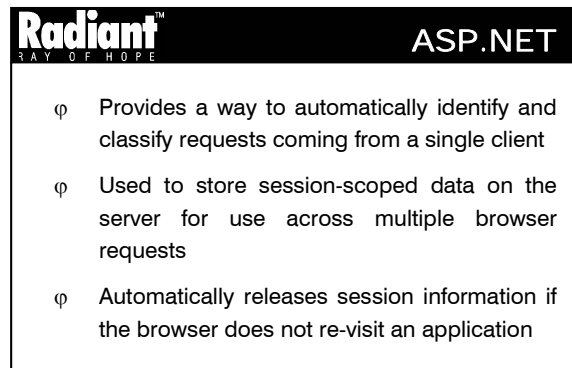
The following code shows the **Session\_End** event:

```
<script language="C#" runat="server">
void Session_End()
{
    Response.AppendToLog(Session.SessionID + "ending");
}
</script>
```

The **Session\_End** script stores data when the user leaves a session. This information can be used to determine the pages that are more often used to enter and exit the Web site.

When an instance of an object is created with session scope, a new instance of the object is created for each user. In general, the objects created with session scope demand more memory than objects created with application scope. However, there will be less access contention over the object because only one user can access each instance of the object.

### SessionState



HTTP is a stateless protocol, which means that it provides no method to automatically recognize that a series of requests are all from the same client. It does not even determine whether a single browser instance is still actively viewing a page or site. As a result, building web applications that need to maintain some cross-request state information (shopping carts, data scrolling, etc) can be extremely challenging without additional infrastructure help.

Session State provides an easy, robust and scalable way to automatically identify and classify requests coming from a single browser client into a logical application “session” on the server. It is also used to store session-scoped data on the server for use across multiple browser requests and to raise the appropriate session lifetime management events (**Session\_Start**, **Session\_End**, etc.) that can be handled in application code. A Session state will automatically release session information if the browser does not re-visit an application after a specified timeout period.

Session supports provided by ASP.NET are:

- Consistent with other .NET Frameworks APIs
- Reliable session state facility that can survive IIS crashes and worker process restarts and ensures that session data is not lost when these failures occur
- Can be used in both Web-farm and Web-garden scenarios. Enable administrators to allocate more processors/ machines to a web application, and hence increase scalability
- Works with browsers that do not support the use of HTTP cookies for privacy or legal reasons

### Properties

The properties of the session state are given below:

#### Timeout

The timeout property specifies the period in minutes a session should be valid. The default value is 20 minutes.

#### Cookieless

The Cookieless property indicates whether a cookie should be used as an identity key or not. The default setting is **false**. When the property is set to **true**, ASP.NET automatically encodes cookie data in the URL and passes it along with the request.

#### Count

The count attribute determines the number of items in the Cookies collection.

### Collections

The various collections of the session state are as follows:

#### Contents

The Contents collection exposes all variable items that have been added to the session-state collection directly through code.

```
Session["Message"] = "Bar";
```

#### StaticObject

The StaticObject collection exposes all variable items that have been added to the Session-state collection through the `<object runat="server">` `</object>` tags with the "Session" scope within the global.asax file.

```
<OBJECT RUNAT="SERVER" SCOPE="SESSION" ID="MyInfo"
  PROGID="Scripting.Dictionary">
</OBJECT>
```

The **StaticObjects** collection cannot have objects that are added to this collection in any other place within a .NET Application. The .NET Page Compiler automatically inserts member references to all objects stored within the **StaticObjects** collection at page compilation time. This enables direct access to **Session** objects at the time of page request without having to go through the **Application** collection.

```
<%@Page Language="C#" %>
<%
Response.Write(MyInfo.ToString());
%>
```

#### Method

The following are the methods of the Session object.



## Abandon

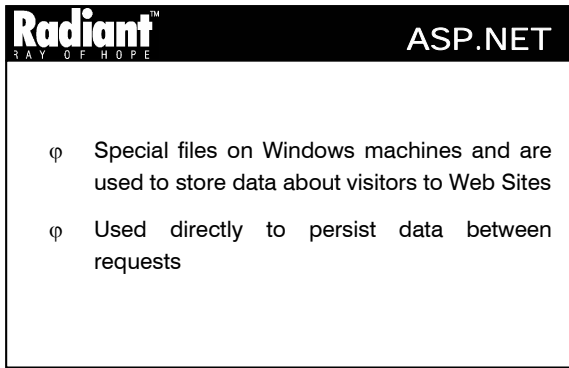
The Abandon method terminates a user's session and releases the memory allocated to maintain user information. If the user leaves the web site, the information is maintained until the session times out. If the user returns before the session times out, the web server does not create another new Session object; it uses the existing one. This is illustrated below:

```
Response.Write (Session.SessionID);
Session.Abandon();
```

In the above code, the session ID of the user is displayed on the screen and then the **Session.Abandon** method is called. When the user requests a new page, the information in the **Session** object is lost and a new ID is assigned to the session. The server treats the user as a new user after the **Abandon** method is called.

The **Session.Abandon** method has certain special scope characteristics. When the **Session.Abandon** method is called, the Web server continues to process the remaining ASP code on the page.

## 8.4 Cookies



used to store data about visitors to Web sites. A Web server can insert information into these cookie files. Storing cookies on the client is one of the methods used by the ASP.NET's session state in order to associate requests with sessions. Cookies can also be used directly to persist data between requests. However, the data is then stored on the client and sent to the server with every request. The size of the cookie is limited to 4096 bytes by the browser.

**Note :** Cookies work on the majority of browsers, but fail completely when used with browsers that do not support them, and hence the sessions will also fail.

### How to create a Cookie

A cookie is created using HttpCookie class. The Page\_Load method in the following code creates a new cookie, initializes it and stores the cookies on the client.

```
protected void Page_Load(Object sender, EventArgs E)
{
    HttpCookie cookie = new HttpCookie("p1");
    cookie.Values.Add("ForeColor", "brown");
    Response.AppendCookie(cookie);
}
```

The HttpCookie class provides a type-safe way to access multiple HTTP cookies. If expiry date is not specified, then the cookie will expire when the user leaves the Web site. The preceding script is used to create a cookie. The other properties of a cookie are:

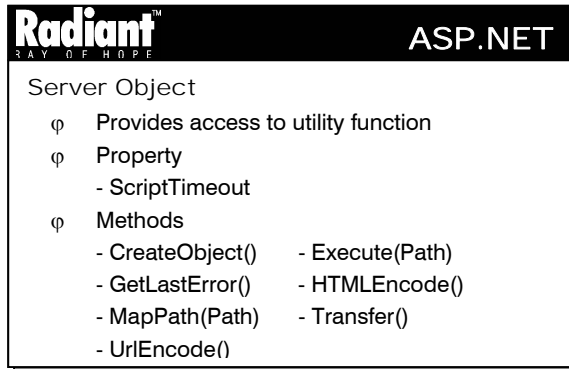
```
<%
```

```

cookie.Expires = DateTime.MaxValue;
cookie.Domain = ".forindia.com";
cookie.Path = "/examples";
cookie.Secure = "True".ToBoolean();
%>

```

## 8.5 Server Object



The Server object provides access to the utility functions of the server.

### Property of Server object

#### ScriptTimeout

The **ScriptTimeout** property specifies the amount of runtime in seconds for a script before it terminates. The default value is 90 seconds.

The following code causes a script to timeout if the script takes more than 150 seconds to complete.

```
<% Server.ScriptTimeout = 150 %>
```

The value of the **ScriptTimeout** property can be retrieved as well.

```
<% timeout = Server.ScriptTimeout %>
```

### Methods of Server object

#### CreateObject( )

The **CreateObject** method is probably the most widely used and the most important method available through the Built-in Asp.Net Objects. It allows to instantiate the components of a script, or in different terms, create an instance of other objects. As a direct consequence, it is possible to use and access any collections, events, methods, and properties associated with these objects.

There is one mandatory argument **ObjectID** that specifies the type of object to be created.

#### Execute(Path)

The **Execute** method allows to call another ASPX page from inside an ASPX page. When the called ASPX page completes its tasks, control is then returned to the calling ASPX page. The overall effect is very

similar to a function or subroutine call. Any text or output from the called ASPX page will be displayed on the calling ASPX page. The **Execute** method is a more useful alternative to using server-side includes.

In contrast, the **Transfer** method allows to transfer from one ASPX page to another without returning to the calling ASPX page.

There is one mandatory argument **Path** that is a string specifying either the absolute or relative path of the ASPX page being called. The file name must be included in the path. The entire **Path** must be enclosed inside a pair of quotes. The **Path** argument cannot include a query string, however, any query string that was available to the calling ASPX page will be available to the called ASPX page.

The following is an example that uses the above method:

```
// CallingAsp.aspx
<HTML>
<BODY>
Hello <%Server.Execute("CalledAsp.aspx")%> - Welcome to Radiant
</BODY>
</HTML>

// CalledAsp.aspx
<%
Response.Write "Anitha"
%>
```

### **GetLastError( )**

The **GetLastError** method returns the last recorded Exception.

### **HTMLEncode( )**

The **HTMLEncode** method applies HTML encoding syntax to a specified string of ASCII characters. For example, this allows to display a HTML tag on a web page and not have it treated as an actual tag.

There is one mandatory argument **String** that is the string to be encoded.

Following is an example of the above method:

```
<% Response.Write Server.HTMLEncode("The tag for a table is: <Table>") %>
```

On executing the above line, the following will be returned to the server:

```
The tag for a table is: &lt;Table&gt;
```

On processing the above line will be displayed in the browser as follows:

```
The tag for a table is: <Table>
```

### **MapPath(Path)**

The **MapPath** method maps a relative or virtual path to a physical path. This method does not check for the validity or the existence of the physical path. If the path starts with a forward or backward slash, the method returns the path as if the path is a full virtual path. If the path does not start with a slash, then the method returns the path relative to the directory of the ASPX file being processed.

There is one mandatory argument path that is the path to be mapped.



### Practice 8.3

There is one mandatory argument **Path** that is the path to be mapped.

The following example illustrates the `MapPath` method.

```
<HTML>
<HEAD>
</HEAD>
<BODY>
The path of this file is <% Response.Write Server.MapPath("test.aspx")
%>
The path of file1 is <% Response.Write Server.MapPath("test\test.aspx")
%>
The path of file2 is <% Response.Write Server.MapPath("/")
%>
</BODY>
</HTML>
```

```
The path of this file is D:\Inetpub\wwwroot\test.aspx
The path of file1 is D:\Inetpub\wwwroot\test\test.aspx
The path of file2 is D:\Inetpub\wwwroot
```

In the above code, the first **MapPath** method considers **test.aspx** as a page under the root directory. The second method considers **test.aspx** as a page residing in the **test** sub-directory of the root directory. The third method returns the path of the root directory.

### Transfer()

The `Transfer` method allows to transfer from inside one ASPX page to another ASPX page. All of the state information that has been created for the first (calling) ASPX page will be transferred to the second (called) ASPX page. This transferred information includes all objects and variables that have been given a value in an Application or Session scope, and all items in the Request collections. For example, the second ASPX page will have the same SessionID as the first ASPX page.

When the second (called) ASPX page completes its tasks, control does not return to the first (calling) ASPX page.

In contrast, the `Execute` method that allows to call another ASPX page, and when the called page has completed its tasks, control returns to the calling ASPX page.

It has one mandatory argument namely `Path`, that is a string specifying either the absolute or relative path of the ASPX page being called. The file name must be included in the path. The entire `Path` must be enclosed inside a pair of quotes.

### URLEncode()

The **URLEncode** method takes a string and converts it into a URL-encoded format. For example, it is possible to use **URLEncode** to ensure that hyperlinks in Active Server Pages are in the correct format.

This contains a mandatory argument **String** that is the string to be encoded.

The following line of code illustrates the usage of the **URLEncode** method:

```
<% Response.Write Server.URLEncode("http://www.issi.net") %>
```

When the above line is executed in a page, the output is similar to the following:

```
http%3A%2F%2Fwww%2Eissi%2Eenet
```

## 8.6 Short Summary

- An ASP.NET application is a collection of files, pages, handlers, modules and executable code that can be invoked in the scope of a given virtual directory and its subdirectories on a Web Application Server
- An ASP.NET application is created when a request is made for the first time to the server
- An application variable contains data that is used in all the web pages and by all the users of an application
- The global.asax file contains syntax for coding the server-side application object
- A collection represents a logical storage unit that helps to manage information
- A session keeps track of user-specific information when a user navigates the site without asking the users identity for every request from the server
- A session begins when a user requests a page from a Web site. A session is closed when the user leaves the Web site
- Session variables like application variables can be used in multiple ASP.NET pages
- Session State provides an easy, robust and scalable way to automatically identify and classify requests coming from a single browser client into a logical application “session” on the server
- Cookies are special files on Windows machines that are used to store data about visitors to Web sites

## 8.7 Brain Storm

1. What are the uses of Application state?
2. What are the uses of Session state?
3. Explain the importance of global.asax file?
4. What are Cookies? What is their use?
5. How is synchronization of variables achieved?

❧❧❧

## Lecture 9

---

# ADO.NET - I

---

### Objectives

In this lecture you will learn the following

- + Knowing about DBMS
- + Knowing about SQL
- + Manipulating of database in ASP.NET
- + Advantages of ADO.NET over ADO

---

## Coverage Plan

---

### Lecture 9

- 9.1 Snap Shot
- 9.2 Database
- 9.3 DBMS
- 9.4 DBMS Standardization
- 9.5 Structured Query Language
- 9.6 Using SQL as a DataQuery Language
- 9.7 Manipulation of database in ASP.NET
- 9.8 Introduction to ADO.NET
- 9.9 Short Summary
- 9.10 Brain Storm

## 9.1 Snap Shot

This session is designed to introduce the user to the concepts of database and DBMS. It acquaints the user with various commands in SQL. Further, the user is introduced to ADO.NET and its advantages over ADO.

Every organization has a pool of resources that it must manage effectively to achieve its objectives. Although their rules differ, all resources human, financial and material share a common characteristic.

The organization that fails to treat data or information as resource and to manage it effectively will be handicapped in how it manages its manpower, material and financial resources. In order to satisfy the information requirements of management, the data should be stored in an organized form.

### Data verses Information

Data are facts concerning people, places, events or other objects or concepts. Data are often relatively useless to decision-makers until they have been processed or refined in some manner. Information is data that have been processed and refined and then given in the format that is convenient for decision making or other organizational activities. For example, report about student fee paid details is useful information for finance section.

Actually data are the facts stored in the record of a database. But the processed facts are presented in a form for usage is information.

## 9.2 Database

Radiant™

ASP.NET

Database

- ⊕ Shared collection of inter-related data designed to meet varied information needs
- ⊕ Acts as a media to store data in an organized way
- ⊕ Two properties
  - Integration
  - Sharing

A database is a shared collection of interrelated data designed to meet the varied information needs of an organization. For example, in a payroll application each person's record has Name, Age, Designation, Basic Pay etc as columns. So payroll database has a collection of all employee records, that are interrelated. From the database, various reports like payslip, persons with particular designation, service report etc can be obtained. The database acts as a media to store the data in an organized way so that it can be managed effectively. A database has two important properties integration and sharing.

Typical examples of **information stored** for some practical purpose are:

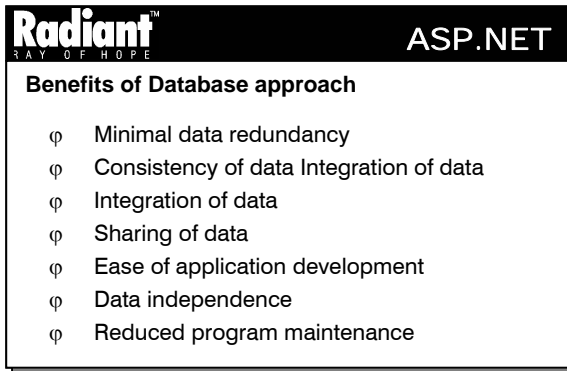
Information collected for the sake of making a statistical analysis, e.g. the national census. Operational and administrative information required for running an organization or a commercial concern will take the form of stock records, personnel records, customer records etc.

Because of the investment involved in setting up a database, the expectation must be that it will continue to be useful, over years rather than months. But the relationship with time varies from one type of



information to another. An organizational database may not change very drastically in size, but it will be subject to frequent updating (deletions, amendments, insertions) following relevant actions within the organization itself. Ensuring the accuracy, efficiency and security of this process is the main concern of many database designers and administrators.

### Benefits of the Database approach



The database approach offers a number of important advantages compared to the file system. These benefits include minimal data redundancy, consistency of data, integration of data, sharing of data, enforcement of standards, ease of application development, uniform security, privacy and integrity controls, data accessibility and responsiveness, data independence, and reduced program maintenance.

### Minimum data redundancy

With the database approach, data files that were separate are integrated into a single, logical structure. So each item is ideally recorded in only one place in the database. Hence, in a database system, the data redundancy is controlled.

### Consistency of data

By eliminating data redundancy, the consistency of data is greatly improved. If there is any change in the data, it can be incorporated in one place unlike the traditional file system where the change has to be incorporated manually in each and every place.

### Integration of data

In a database, data are organized into a single, logical structure, with logical relationships defined between associated data entities.

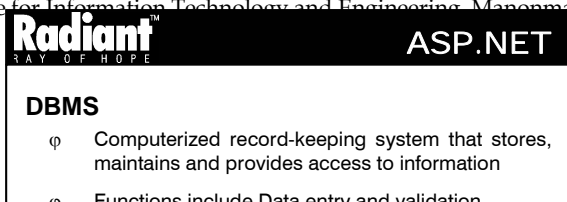
### Sharing of data

The database is intended to be shared by all authorized users in the organization.

### Database Concepts

The database stores information in tables that comprises of columns and rows. The columns separate data into fields. Each column contains a different kind of information. Each row constitutes a record. Let us consider an organization that needs to store the information about its employees. The information might include employee name, age, department, designation, salary, etc. Each of these is a column (or field). The details of each employee form a row (or record).

## 9.3 DBMS



---

A DBMS (Database Management System) is a computerized record-keeping system that stores, maintains and provides access to information. The function of the DBMS is to store, maintain and retrieve information as required by applications, programs or users. The functions of the DBMS are listed below:

### **Data entry and validation**

Validation may include:

- Type Checking
- Range Checking
- Consistency Checking

In an interactive data entry system, errors should be detected immediately and recovery and re-entry should be permitted. If the database is error bound then it will be the main cause to make the program error prone.

### **Updating**

Updating of data is very important otherwise it will be fruitless in future. Updating involves:

- Record Insertion
- Record Modification
- Record Deletion

Updating may take place interactively, or by submission of a file of transaction records; handling these may require a program of some kind to be written, either in a conventional programming language or in a language supplied by the DBMS for constructing command files.

### **Data retrieval on the basis of selection criteria**

For this purpose most systems provide a **Query Language** through which the characteristics of the required records may be specified. Query languages differ enormously in power and sophistication but a standard, which is becoming increasingly common, is based on the so-called RELATIONAL operations.

These allow:

- Selection of records on the basis of particular field values
- Selection of particular fields from records to be displayed
- Linking together records from two different files on the basis of matching field values

Arbitrary combinations of these operations on the files making up a database can answer a very large number of queries without requiring users to go into one record at a time processing.

### **Report definition**

Most systems provide facilities for describing how summary reports from the database are to be created and laid out on paper. These may include obtaining:

- COUNTS
- TOTALS
- AVERAGES
- MAXIMUM and MINIMUM values

On a table over a particular **Control Field** the above layouts have to be found out. Also specification of **Page** and **Line Layout, Headings, Page-Numbering**, and other narrative are required to make the report comprehensible.

### **Security**

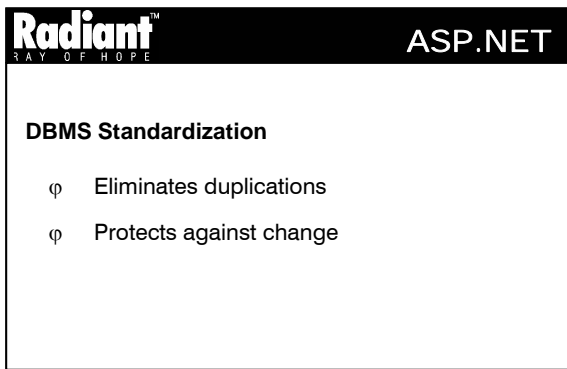
The security consists of several aspects:

- Ensuring that only those authorized can see and modify the data, generally by some extension of the password principle
- Ensuring the consistency of the database where many users are accessing and up-dating it simultaneously
- Ensuring the existence and INTEGRITY of the database after hardware or software failure. At the very least this involves making provision for back-up and re-loading

### **Importance of Databases and DBMS**

An organization uses a computer to store and process information because it expects speed, accuracy, efficiency, economy etc. beyond what could be achieved by using clerical methods. The objectives of using a DBMS must in essence be the same although the justifications may be more indirect.

## **9.4 DBMS Standardization**



Early computer applications were based on existing clerical methods and stored information was partitioned in much the same way as manual files. But the computer's processing speed gave a potential for relating data from different sources to produce valuable management information, provided some standardization could be imposed over departmental boundaries. The idea emerged of the integrated database as a central resource emerged. Here, data is captured as close as possible to its point of origin and transmitted to the database, then extracted by anyone within the organization who requires it. However, many provisos have become attached to this idea in practice, it still provides possibly the strongest motivation for the introduction of a DBMS in large organizations. The idea is that any piece of information is entered and stored just once, eliminating duplications of effort and the possibility of inconsistency between different departmental records. Data redundancy has to be removed as much as possible.

### **Advantages**

Organizational requirements change over time, and applications laboriously developed need to be periodically adjusted. A DBMS gives some protection against change by taking care of basic storage and retrieval functions in a standard way, leaving the application developer to concentrate on specific

organizational requirements. Changes in one of these areas need not affect elsewhere. In general, a DBMS is a substantial piece of software, the result of many man-years of effort.

The points discussed above are probably most relevant to larger organizations using a DBMS for its administrative functions - the environment in which the idea of databases first originated. In other words the convenience of a DBMS may be the primary consideration. The purchaser of a small business computer needs all the software to run it in a package form, written so that the minimum of expertise is required to use it. The same applies to departments (e.g. Research & Development) with special needs that cannot be satisfied by a large centralized system. When comparing database management systems it is obvious that some are designed in the expectation that professional staff will be available to run them, while others are aimed at the total novice. Actual monetary costs vary widely from, for instance, a large multi-user Oracle system to a small PC-based filing system.

## 9.5 Structured Query Language

Radiant™

ASP.NET

**SQL**

- ⊕ Basic language that allows users to access data in database management systems
- ⊕ Mainly classified into DDL and DML

**DDL**

- ⊕ Create Database Statement
- ⊕ Create Table Statement
- ⊕ Alter Table Statement
- ⊕ Drop Table Statement

Structured Query Language (SQL) is a basic language that allows users to access data in database management systems, such as Oracle, Sybase, Informix, Microsoft SQL Server, Access by allowing users to describe the data the user wishes to see. SQL also allows users to define the data in a database, and manipulate that data.

SQL is mainly classified into Data Definition Language (DDL) and Data Manipulation Language (DML).

### Using SQL as a Data Definition Language

Using SQL to define a database means creating a database, creating tables and adding them to a database, updating the design of existing tables, and removing tables from a database.

#### Create Database Statement

The **Create Database** Statement can be used to create a database:

```
CREATE DATABASE databaseName
```

where *databaseName* is the name of the database to be created. For example, the following statement creates a database named **Salesreps**

```
Create Database Salesreps
```

**Note:** The **Create Database** statement is not supported by all SQL implementations

#### Create Table Statement

The **Create Table** statement creates a table and adds it to the database:

```
CREATE TABLE tableName (columnDefinition,... , columnDefinition)
```

Each columnDefinition is of the form: ColumnName columnType

The **columnName** is unique to a table. The **columnType** identifies the type of data that will be stored in the table. Common data types are

- Char(n) - An n character text string
- Int - An integer value
- Float - A floating point value
- Bit - A boolean (1 or 0) value
- Datetime - A date value
- Money - A money value



### Practice 9.1

---

The following is an example of a **Create Table** statement which creates a table named **Customers** containing columns CustName, Company, Cust\_rep, Credit\_limit:

```
CREATE TABLE Customers (CustName char(30), Company char(50), Cust_rep integer, Credit_limit money)
```



The command(s) completed successfully.

The example creates a **Customers** table with the following columns:

- CustName - A 30-character-wide text field
- Company - A 20-character-wide text field
- Cust\_rep - A integer type
- Credit\_limit - A money data type to represent currency values

### ALTER TABLE Statement

The **Alter Table** statement is used to add a row to an existing table or to change the table definitions.

```
ALTER TABLE tableName ADD (columnDefinition... columnDefinition)
```

The values of the newly added columns are set to NULL. Columns are defined as described in the previous section.



### Practice 9.2

---

The following is an example of the ALTER TABLE statement

```
ALTER TABLE CUSTOMERS ADD CONTACT_NAME VARCHAR(20)
```



The command(s) completed successfully.

The preceding SQL statement of the ALTER TABLE adds a column named Contact\_Name to the Customers table

### DROP TABLE Statement

The DROP TABLE statement deletes a table from the database:

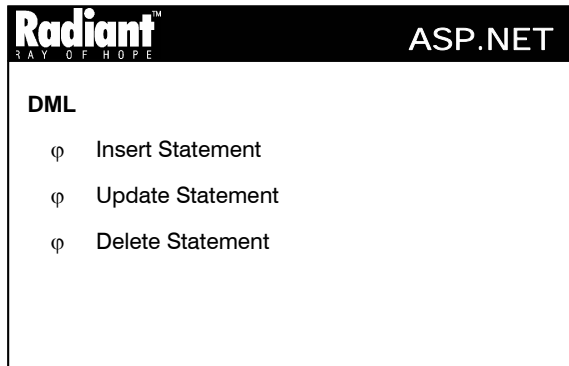
```
DROP TABLE tableName
```

The dropped table is permanently removed from the database. The following is an example of the DROP TABLE statement:

```
DROP TABLE CUSTOMERS
```

The preceding statement removes the CUSTOMERS table from the database. Tables once dropped cannot be retrieved.

### Using SQL as a Data Manipulation Language



One of the primary uses of SQL is to update the data contained in a database. There are SQL statements for inserting new rows into a database, deleting rows from a database, and updating existing rows.

### INSERT Statement

The INSERT statement inserts a row into a table:

```
INSERT INTO tableName VALUES (value 1, ..., value n )
```

In the preceding form of the INSERT statement, **value 1** through **value n** identify all the column values of a row. Character values should be enclosed within single quotes.



#### Practice 9.3

The following is an example of the preceding form of the INSERT statement:

```
INSERT INTO Customers VALUES ('Joy Anand', 'Radiant', 29, 500000)
```



(1 row(s) affected)

The preceding statement adds a row to the Customer table. All columns of this row are filled in.

An alternative form of the INSERT statement may be used to insert a partial row into a table. The following is an example of this alternative form of the INSERT statement:

```
INSERT INTO tableName (columnName 1,...,columnName m) VALUE      (value
1,...,value m)
```

The values of **columnName 1** through **columnName m** are set to **value 1** through **value m**. The value of the other columns of the row are set to NULL.



### Practice 9.4

The following is an example of the preceding form of the INSERT statement:

```
INSERT INTO Customers (CustName, Company)
VALUES ('Smith', 'Radiant')
```



(1 row(s) affected)

The preceding statement adds a row to the Customer table with CustName - Smith and Company - Radiant. The other columns of the table are set to null.

### DELETE Statement

The DELETE statement deletes a row or a set of rows from a table:

```
DELETE FROM tableName [WHERE condition]
```

All rows of the table that meet the condition of the WHERE clause are deleted from the table. The WHERE clause is covered in a subsequent section of this session.



If WHERE clause is omitted, all the rows of the table are deleted.



### Practice 9.5

The following is an example of the DELETE statement:

```
DELETE FROM Customers WHERE CustName = 'Smith'
```



(1 row(s) affected)

The preceding statement deletes the row which has CustName as Smith from the Customers table.

### UPDATE Statement

The UPDATE Statement is used to update an existing row of a table:

```
UPDATE tableName SET columnName1 = 'value1', ...,columnName = 'value'
[WHERE condition]
```

All the rows of the table that satisfy the condition of the WHERE clause are updated by setting the columns with the specified values. If the WHERE clause is omitted, all rows of the table are updated.



### Practice 9.6

The following statement is an example of the UPDATE statement:

```
UPDATE Customers SET Credit_Limit = 1200000 WHERE Company = 'Radiant'
```



(1 row(s) affected)

The preceding statement changes the Credit Limit of the company Radiant with the new Credit Limit of Rs12,00,000.

## 9.6 Using SQL as a DataQuery Language

**Radiant™** **ASP.NET**  
SAY OF HOPE

**Select Statement**

- ☐ Specifies a database query
- ☐ Used to retrieve data from a database Where Clause
  - Boolean expression consisting of column names, column values, relational operators and logical operators

The most important use of SQL for many users is for retrieving data contained in a database. The SELECT statement specifies a database query:

```
SELECT columnList1 FROM table1 ,..., tablem [WHERE condition]
```

**Note:** An asterisk (\*) may replace columnList1 to indicate that all columns of the table(s) are to be returned.



### Practice 9.7

The following is an example for select statement.

```
SELECT * FROM CUSTOMERS
```



```
CustName    Company    Cust_rep    Credit_limit
```



```
Joy anand radiant 29 100000.0000
```

The preceding example selects all the rows and columns from the Customers table.

### Where Clause

The WHERE clause is a Boolean expression consisting of column names, column values, relational operators, and logical operators. For example, suppose there are columns such as Department, Salary, and Bonus. The following WHERE clause could be used to find all employees in the Engineering department whose salaries are over 100,000 and bonus are less than 5,000.

```
WHERE Department = 'Engineering ' AND Salary >'100000' AND Bonus <'5000'
```

**Radiant™** **ASP.NET**  
RAY OF HOPE

**WHERE clause can be based on some relational and logical comparisons**

- ☐ Comparison Test
- ☐ Range Test
- ☐ Set Membership Test
- ☐ Pattern Matching Test
- ☐ Null Value Test
- ☐ Compound Search Conditions

### Comparison Test

Comparison test makes use of the relational operators.

They are listed below:

- = equal to
- <> not equal to
- < less than
- <= less than or equal to
- > greater than
- >= greater than or equal to

### Range Test

This test is used to test whether the value lies within the range or not. The keyword are:

- BETWEEN ... AND
- NOT BETWEEN ... AND



### Practice 9.8

The following statement selects rows from the **Customers** table subject to the condition that the credit limit lies between 10000 and 50000.

```
SELECT * FROM CUSTOMERS
```

```
WHERE CREDIT_LIMIT BETWEEN 10000 AND 50000
```

CustName	Company	Cust_rep	Credit_limit
Smith	radiant	29	40000.0000

The preceding example selects rows from **Customers** table that satisfy the condition that the Credit\_limit is between 10000 and 50000.

### Set Membership Test

This is to test whether the value is present in the list of values. The keyword is IN.

#### Example

```
SELECT * FROM CUSTOMERS WHERE CUST_REP IN (10, 15, 20, 25)
```

The above query selects the rows for which the Cust\_Rep takes one of the values 10, 15, 20 or 25.

### Pattern Matching Test

The Pattern matching test is employed to select rows from a table based on a field whose values match a given pattern. The keywords LIKE, %, \_ are the wildcard characters used in the pattern matching test. % is equivalent to \* in DOS. \_(Underscore) is equivalent to ? in DOS. The \$ is used to remove the above special meaning of % and \_.

#### Example

```
SELECT * FROM CUSTOMERS WHERE CUSTNAME LIKE 'S%H'
```

The above query searches for the customers whose first character is S and the last character is H.

```
SELECT * FROM CUSTOMERS WHERE CUSTNAME LIKE 'S$_C'
```

This will search for customers whose names start with S, end with C and contain the character \_.

### Null Value Test

This is used to test whether the column contains null value. The keyword is IS NULL.

### Compound Search Conditions

This is used to test for more than one condition. The keyword AND/OR/NOT is used.

Radiant™

ASP.NET

φ **Order by clause**  
- Used to order result set by specified column(s) of a table

φ **Union**  
- Used to combine two or more tables

φ **Joins**  
- Used when results of two or more queries have to be combined

### Order By

The ORDER BY clause is used to order the result set by the specified column(s) of a table. Each of the column names in the column list may be followed by the **ASC** or **DESC** keywords. If **DESC** is specified, the result set is ordered in descending order. Otherwise, the result set is ordered in ascending order. The following query retrieves the rows of the **Customers** table ordered by the **CustName** field.

```
SELECT * FROM CUSTOMERS ORDER BY CUSTNAME
```

### Union

It is used to combine two or more tables. The necessary criteria are:

- The tables must contain the same number of columns
- Data type of each of the columns of one table should match that of the other

When the keyword **UNION ALL** is used, all the rows of both the tables are displayed. When the keyword **UNION** is used, only distinct rows from both the tables are displayed.

### Example:

```
SELECT * FROM A
UNION (SELECT * FROM B
UNION SELECT * FROM C)
ORDER BY NAME
```

The preceding statement produces a combined output from tables A,B and C. The output is arranged in the ascending order of Names.

### Joins

Joins are used when the results of two or more queries have to be combined. This session discusses two of the joins namely, Equi joins and Non-Equi joins.

### Equi-join

An equi-join is a join with a join condition containing an equality operator. An equi-join combines rows that have equivalent values for the specified columns.



---

### Practice 9.9

The following example performs an equi-join with **Order** and **Customer** tables where both the tables are described below:

**Order** table contains the following columns

```
Ordernum
order_date
cust_rep
qty
amount
```

**Customer** Table contains the following columns

```
cust_num
company
cust_rep
credit limit
```

The query is as follows:

```
select ordernum, amount, credit_limit
from order, customer
where order.cust_rep = customer.cust_rep
```



Ordernum	amount	credit_limit
1	30000.0000	100000.0000
3	20000.0000	50000.0000

This **equi-join** returns the order number, amount and the credit limit of the Customer table where **cust\_rep** of Order table is equal to **cust\_rep** of Customer table.

### Non-Equi-join

A non-equi-join is a join with a join condition containing any operator other than the equality operator.

### Example

The following query selects the order number, amount and credit limit of the Customer table where **cust\_rep** of Order table is not equal to **cust\_rep** of Customer table.

```
select ordernum, amount, credit_limit
from order, customer
where order.cust_rep <> customer.cust_rep
```

**Radiant**<sup>TM</sup> **ASP.NET**  
3 A Y O F H O P E

- φ **Functions**  
- Used to compute against a column or columns of data returned from a query
- φ **Group by**  
- Gather all rows that contain data in specified column(s)
- φ **Having**  
- Allows to specify conditions on rows for each group
- φ **Distinct**  
- Returns information from table columns without repetition

### Functions

Functions are used to compute against a column or columns of data returned from a query. Some of the important functions are

SUM()	totals the column
AVG()	returns the average of the column
MIN()	returns the minimum value of the column
MAX()	returns the maximum value of the column
COUNT()	counts the number of rows for the specified column
COUNT(*)	counts the number rows for all the columns

### Example

```
SELECT CUST_REP, SUM(AMOUNT) FROM ORDERS ORDER BY CUST_REP
```

The above query displays the number of each salesperson and the total sum of orders for each of them.

### **Group by**

The GROUP BY clause will gather all of the rows together that contain data in the specified column(s) and will allow aggregate functions to be performed on the one or more columns.

#### **Example:**

```
SELECT company, SUM(credit_limit)
FROM customer
GROUP BY company
```

This query select the sum of credit\_limit from customer table for each company.

### **Having**

The HAVING clause allows to specify conditions on the rows for each group - in other words, which rows should be selected will be based on the conditions specified in the Having clause. The HAVING clause should follow the GROUP BY clause if used.

#### **Example:**

```
SELECT company, SUM(credit_limit)
FROM customer
GROUP BY company having sum(credit_limit) > 5000
```

This query select the sum of credit\_limit from customer table for each company where sum of credit\_limit is greater than 5000.

### **Distinct**

The SQL SELECT statement with Distinct keyword returns information from table columns without repetition.

## **9.7 Manipulation of database in ASP.Net**

### **Client server computing**

The term client/server computing means connecting a client machine to a server machine and sharing the load of processing between the two. The client requests services and the server provides the requested services. The client does not request data at a file level, but sends a request to the server to execute a query and return specific records.

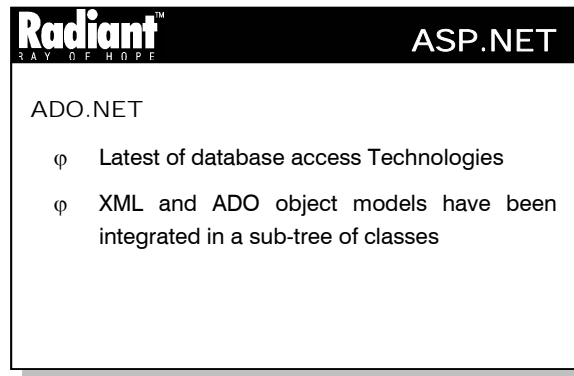
A special type of client/server architecture consisting of three well-defined and separate processes, each running on a different platform is as follows:

- The user interface, which runs on the user's computer (the *client*)
- The functional modules that actually process data. This *middle tier* runs on a server and is often called the *application server*
- A Database Management System (DBMS) that stores the data required by the middle tier. This tier runs on a second server called the *database server*

To accomplish the task of connecting to data sources that are open to many applications, the application and the database must agree on a common method of accessing the database. This agreement is implemented using a complete set of API calls and a complete SQL syntax set.

The database side of this open connectivity is provided by drivers. These drivers will transform the API functions into function calls supported by the particular data source being used. Similarly, the drivers transform the SQL syntax into syntax accepted by the data source. These drivers were originally produced by the manufacturers of the databases, but now there are many third-party vendors as well. The main benefit of this API is that many database formats can now be accessed by applications as their own native databases.

## 9.8 Introduction to ADO.Net



ADO.NET is the latest of the database access technologies that began with the Open Database Connectivity (ODBC) application programming interface (API). Microsoft introduced open data base connectivity with the promise of creating a singular common access methodology for databases. ODBC has come a long way since those early days. Almost every major database in use today supports ODBC drivers, and third party developers provide optimized driver versions. The primary focus of the ODBC is to provide a consistent interface to database data sources.

As time passed by COM landed at the database territory and started a colonization process that culminated with OLE DB. OLE DB is a set of COM-based interfaces that exposes data from a variety of sources. OLE DB interfaces provide applications with uniform access to data stored in diverse information sources, or data stores. These interfaces support the amount of DBMS functionality appropriate to the data store, enabling the data store to share its data.

With .NET, Microsoft is offering a general-purpose framework—the Framework Class Library—that will cover all the existing Windows API and more. In particular, it will include a number of frequently used libraries now available through separate COM objects. The XML and ADO object models have been integrated in a subtree of classes called ADO.NET

### Advantages of ADO.Net over ADO

The advantages of ADO.NET over ADO are as follows:

- ADO.NET is the base that will form the foundation of data-aware .NET applications
- Unlike ADO, ADO.NET has been purposely designed for more general, and less database-oriented, guidelines
- ADO.NET gathers all the classes that allow data handling. Such classes represent data container objects that feature typical database capabilities—indexing, sorting, views

- ADO.NET is the definitive solution for .NET database applications, it shows off an overall design that is not as database-centric as the ADO model
- ADO.NET is quite different from ADO. It is a new data access programming model that requires full understanding, commitment, and a different mindset
- ADO.NET is not ADO adapted to fit into the .NET infrastructure. This is apparent when looking at ADO.NET in terms of syntax, code design, and migration

### 9.9 Short Summary

- Data are facts concerning people, places, events or other objects or concepts
- Information is data that have been processed and refined and then given in the format that is convenient for decision making or other organizational activities
- A database is a shared collection of interrelated data designed to meet the varied information needs of an organization
- A DBMS (Database Management System) is a computerized record-keeping system that stores, maintains and provides access to information
- SQL also allows users to define the data in a database, and manipulate that data
- SQL is mainly classified into Data Definition Language (DDL) and Data Manipulation Language (DML)
- One of the primary uses of SQL is to update the data contained in a database
- Functions are used to compute against a column or columns of data returned from a query
- ADO.NET is the latest among of the database access technologies that started with the ODBC API

### 9.10 Brain Storm

1. What is a database?
2. What are the advantages of using a database to store data?
3. What is a DBMS?
4. What is SQL?
5. What are DDL and DML statements?
6. What is a function?
7. List the advantages of ADO over ADO.NET.

## Lecture 10

---

# ADO.NET - II

---

### Objectives

In this lecture you will learn the following

- + What is meant by Managed Provider?
- + Learning about ADO connection object
- + What is the user of command object?



---

## Coverage Plan

---

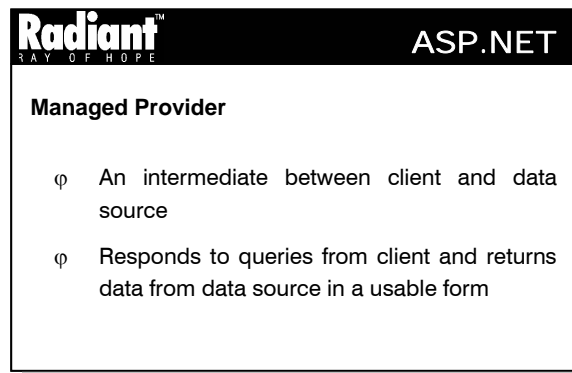
<b>Lecture 10</b>
10.1 Snap Shot
10.2 Managed Providers
10.3 Connection Object
10.4 Command Object
10.5 DataReader
10.6 Short Summary
10.7 Brain Storm

## 10.1 Snap Shot

This session is designed to introduce the user to Manager Providers, Command and Connection objects. It also introduces the DataReader, a forward-only stream used to store the retrieved data.

The Managed Provider acts as an intermediate between the client and the data source. The Connection object is used to connect to the data source. The Command object is used to create and execute commands against the data source so that the data is retrieved in an useful form to the client. The DataReader is a forward-only stream which is used to store the retrieved data, for use in the application.

## 10.2 Managed Providers



The **Managed Provider** is an intermediate between the client and the data source. It responds to queries from the client and returns data from the data source in a usable form. The .NET Framework provides two types of Providers:

- ADO
- SQL

The ADO Provider is used to connect to any data source, whereas the SQL Provider is specially designed for use with the SQL database. The support for the ADO Provider is given by the **System.Data.ADO** Namespace. The support for the SQL Provider is given by the **System.Data.SQL** Namespace.

The important classes contained in the **System.Data.ADO** Namespace are:

- ADOCommand
- ADOConnection
- ADODataReader
- ADODataSetCommand
- ADOError
- ADOErrors
- ADOException
- ADOParameter
- ADOParameters
- ADOProperty
- ADOProperties

The important classes contained in the **System.Data.SQL** Namespace are:

- SQLCommand

- SqlConnection
- SqlDataReader
- SQLDataSetCommand
- SQLError
- SQLErrors
- SQLException
- SqlParameter
- SQLParameters

The usage of the above classes will be dealt with while discussing the corresponding objects.

The following figure (Figure 10.1) shows the object model of ADO.NET

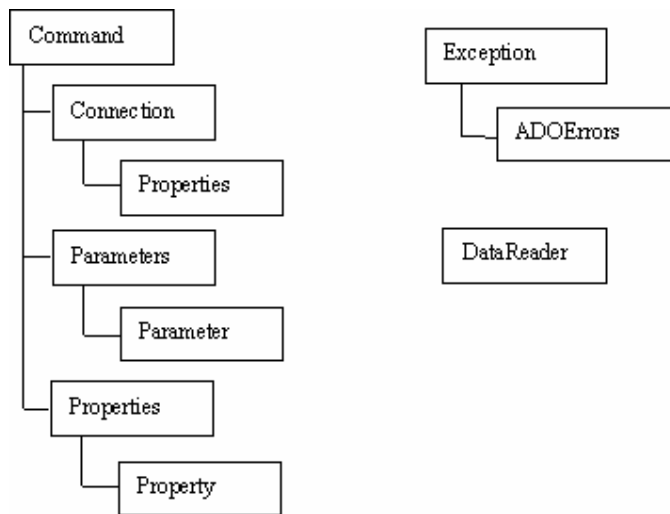


Figure 10.1

In the above figure, Properties, Parameters and ADOErrors represent collections.

**Radiant™**  
3 A Y O F H O P E

**ASP.NET**

**Parameter object**

- ☐ Represents and parameter or argument associated with a Command object
- ☐ Enables reuse of command

**Property object**

- ☐ Contains metadata and setting for a property in an ADO data source

**ADOException**

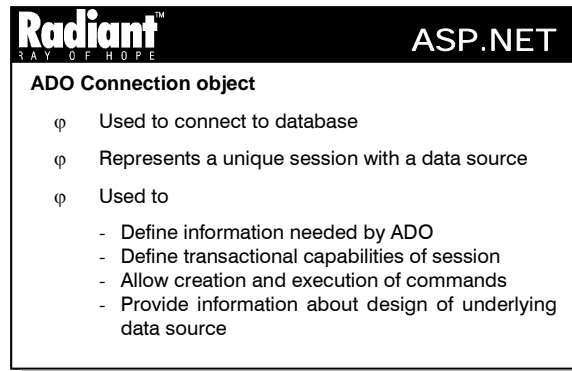
- ☐ Instance created when ADO encounters a situation that it cannot handle

The **parameter** object represents a parameter or argument associated with a Command object based on a parameterized query or stored procedure. The parameter object enables reuse of the same command, by altering only the values passed to it. For example, by using the parameter object, an insert statement can be reused by changing the values of the fields passed as parameters to the command. The **property** object

contains the metadata and settings for a property in an ADO data source. When ADO encounters a situation that it cannot handle, it creates an instance of the **ADOException** class. Any operation involving the objects in ADO.NET can generate one or more provider errors. Whenever such errors occur, the ADO creates an instance of the **ADOError** class. These instances are created and managed by the **ADOErrors** class which in turn is created by the **ADOException** class.

The following paragraphs deal with the Connection, Command and DataReader objects.

### 10.3 Connection Object

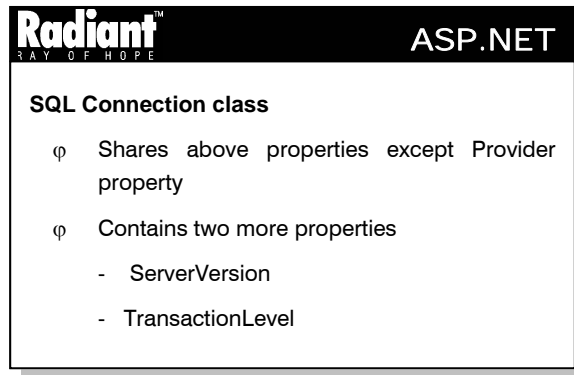


There are four basic operations involved in creating an ADO application. They are: retrieving data, examining data, editing data and updating data. To get the data from the database, a connection has to be established. The connection to the database is established using the **ADOConnection** object. A connection object represents a unique session with a data source. It is used to:

- Define the information needed by the ADO to communicate with data sources and create sessions
- Define the transactional capabilities of the session
- Allow the creation and execution of commands against the data source
- Provide information about the design of the underlying data source

The properties of the ADOConnection class are:

- **ConnectionString** – Indicates the string to be used to open a data store which includes the username, password etc.
- **ConnectionTimeout** – Denotes the time to wait to establish a connection, before terminating the attempt and generating an error
- **Database** – Represents the database to be used for the connection
- **DataSource** – Represents the source of the data to which the connection has to be established
- **IsolationLevel** – Represents the isolation level used for local transactions
- **Password** – Represents the password to be used while establishing the connection
- **Provider** – Indicates the name of the provider to be used for connecting to the data source
- **Site** – Denotes the site of the Component
- **State** – Denotes the current state of the connection
- **UserID** – Specifies the user ID to be used while connecting



The SQLConnection class also shares the above properties except the **Provider** property. The SQL Server Managed Provider uses the private protocol called *tabulardata stream* to communicate with SQL Server. It does not use OLEDB, ADO or ODBC. So, the Provider property is not present in this class.

Instead, the SQLConnection class contains two more properties listed below:

- **ServerVersion** – Denotes the version of the SQL Server to which the connection has been established
- **TransactionLevel** – Specifies the transaction level

An instance of the ADOConnection class can be created using one of the following two constructors:

- **ADOConnection()** – Creates a new instance of ADOConnection with no parameters
- **ADOConnection(string connectionString)** – Creates a new instance of ADOConnection with the specified connection string that indicates the server in which the data is stored, user name, password etc.

An instance of the SQLConnection class can be created using one of the following four constructors:

- **SQLConnection()** – Creates a new instance of the SQLConnection class with no parameters
- **SQLConnection(string connectionString)** – Creates a new instance of the SQLConnection class using the specified connection string that indicates the server in which the data is stored, user name, password etc.
- **SQLConnection(string dataSource, string userID, string password)** – Creates a new instance of the SQLConnection to the specified dataSource, using the specified user ID and password
- **SQLConnection(string dataSource, string userID, string password, string database)** – Creates a new instance of the SQLConnection to the specified database in the specified dataSource using the specified user ID and password

Some of the important methods common to both the classes are listed below:

- **Open** – Opens a database connection with the current settings
- **BeginTransaction** – Begins a database transaction
- **SaveTransaction** – Saves a point within the transaction
- **RollbackTransaction** – Rolls back a transaction from a pending state
- **CommitTransaction** – Commits the transaction

- **Dispose** – Disposes of the current SqlConnection object
- **Close** – Closes the connection to the database

The following part of code illustrates how to connect to a database using the SqlConnection class:

```
String connStr = "server = localhost; uid = swathy ; pwd = radiant;
    database = acctmaster";
SqlConnection conn = new SqlConnection(connStr);
conn.Open();
```

In the above lines, the SQL Server resides in the same machine as the client. So, the server is mentioned as *localhost*. The user id (uid) is *swathy* and password (pwd) is *radiant*. The database to which the connection has to be established is *acctmaster*.

**Note:** While using SqlConnection, the provider need not be specified since the default SQL Provider will be used. But, while using ADOConnection, the provider needs to be specified.



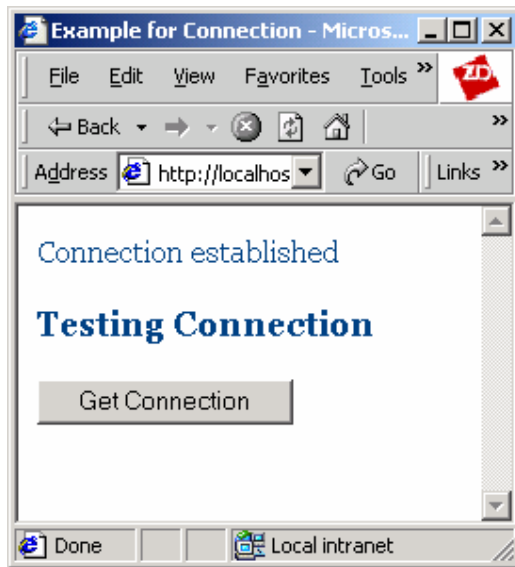
### Practice 10.1

The following example uses the SqlConnection object to connect to the **master** database.

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

<html>
<title> Example for Connection </title>
<script language="C#" runat="server">
SqlConnection myConnection;
void Connect(Object Src, EventArgs E)
{
    myConnection = new SqlConnection(
        "server=localhost;uid=sa;pwd=;database=master");
    myConnection.Open();
    Response.Write("Connection established");
    myConnection.Close();
}
</script>
<body>
    <form runat="server">
        <h3>Testing Connection</h3>
        <asp:button OnClick="Connect" Text="Get Connection" runat="server" />
    </form>
</body>
</html>
```





In the above code, when the user clicks on the “Get Connection” button, its **OnClick** event is fired. The event calls the **Connect** method that in turn establishes the connection with the SQL Server database named **master** that is present in the same system as the client as the user **sa**. All these are specified in the connection string when an instance of **SqlConnection** is created. The **Open** method of the connection object actually establishes the connection.

#### 10.4 Command Object

**Radiant™**  
RAY OF HOPE

**ASP.NET**

**Command Object**

- ⊕ Used to request any type of operation from the provider
- ⊕ Very useful when a command is to be reused or needs to receive input parameters
- ⊕ Properties common to ADOCommand and SqlCommand classes

Once a connection to a database has been established, the queries have to be sent and the results retrieved. This is achieved through the Command object. The Command object can be used to request any type of operation from the provider, if the provider understands the command string properly. It is not necessary for a command to be executed only through the Command object. The command object is very useful when a command is to be reused or needs to receive input parameters.

The following are the properties of the Command object common to both the ADOCommand and SqlCommand classes:

- **ActiveConnection** – Represents the ADOConnection / SqlConnection used by the current instance of ADOCommand / SqlCommand

- **CommandBehavior** - Represents the behavior that the ADOCommand / SQLCommand should exhibit when executed
- **CommandText** - Denotes the SQL text command to run against the data source
- **CommandTimeout** - Denotes the time to wait before which the attempt to execute the command is terminated and an error is generated
- **CommandType** - Represents how the CommandText property is interpreted
- **Parameters** - Represents the collection of ADOParameters / SQLParameters
- **RecordsAffected** - Represents the number of records affected by the execution of the command
- **UpdatedRowSource** - Represents how the command results are applied when they are used by the Update method

The methods of the SQLCommand class are:

- **Clone** - Creates a duplicate of the current instance
- **Equals** - Checks if the specified object is the same instance as the current object
- **Execute** - Executes the specified CommandText against the ActiveConnection and builds a DataReader
- **ExecuteNonQuery** - Executes a SQL command that does not return any rows against the ActiveConnection
- **GetHashCode** - Serves as a hash function for a particular type
- **GetType** - Gets the Type of the object
- **ResetParameters** - Resets the parameters within the parameters collection
- **ToString** - Returns a string representing the current object

In addition to the above methods, the ADOCommand class contains the following methods:

- **Cancel** - Cancels the execution of a command
- **Prepare** - Creates a compiled version of the command on the data source
- **ResetCommandTimeout** - Resets the CommandTimeout property to the default value
- **ShouldPersistCommandTimeout** - Determines whether the CommandTimeout property should remain

There are five constructors in the SQLCommand class that can be used to create a command object. They are:

- **SQLCommand()** - Creates a new instance of SQLCommand without any parameters
- **SQLCommand(string cmd)** - Creates a new instance of SQLCommand using the specified command text
- **SQLCommand(string cmd, SqlConnection conn)** - Creates a new instance of SQLCommand using the specified connection object and command text
- **SQLCommand(string cmd, string connectionString)** - Creates a new instance of SQLCommand using the specified connection string and command text
- **SQLCommand(SqlConnection activeConnection, string cmd, CommandType type, SqlParameter[] param, UpdateRowSource src)** - Creates a new instance of SQLCommand using the specified connection, command text, parameters, source for updaterow and of the specified command type.



The ADOCommand class contains five similar constructors to create an ADOCommand object. The fifth constructor alone contains an additional boolean parameter that determines whether or not the command uses named parameters. The constructor is shown below:

```
ADOCommand(ADOConnection activeConnection, string cmd, CommandType type, bool
namedparams, SqlParameter[] param, UpdateRowSource src)
```

The following lines show how to create a command object:

```
String st = "select * from emp";
SqlCommand cmd = new SqlCommand(st, conn);
```

The lines above create a command with the query string "select \* from emp" and with the connection object "conn".



## Practice 10.2

The following example uses the SqlCommand object to insert a record into the **emp** table of the **master** database.

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

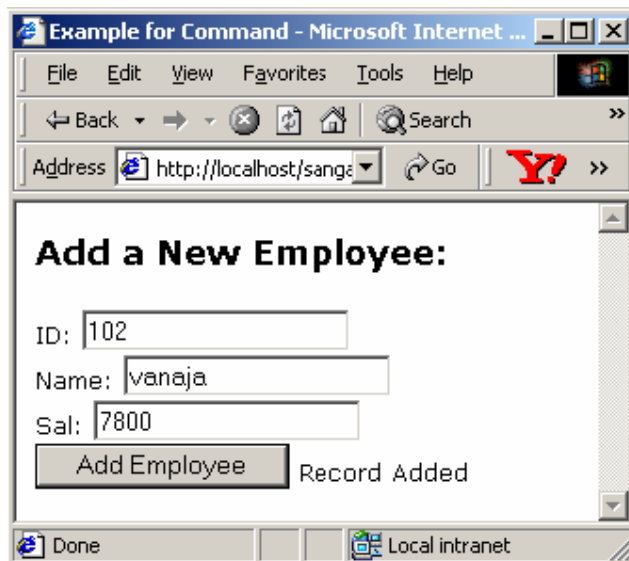
<html>
<title> Example for Command </title>
<script language="C#" runat="server">
SqlConnection myConnection;
protected void Page_Load(Object Src, EventArgs E)
{
    myConnection = new SqlConnection
        ("server=localhost;uid=sa;pwd=;database=master");
}

public void AddEmp(Object sender, EventArgs E)
{
    if(eid.Value == "" ||ename.Value == "" ||esal.Value == "")
    {
        lb.Text = "ERROR: Null values not allowed for
            employee ID, Name or salary";
        return;
    }
    String insertCmd = "insert into emp values (@eno, @eName, @sal)";
    SqlCommand myCommand = new SqlCommand(insertCmd, myConnection);
    myCommand.Parameters.Add(new SqlParameter("@eno",
        SqlDbType.SmallInt, 5));
    myCommand.Parameters["@eno"].Value = eid.Value;
    myCommand.Parameters.Add(new SqlParameter("@eName",
        SqlDbType.VarChar, 40));
    myCommand.Parameters["@eName"].Value = ename.Value;
    myCommand.Parameters.Add(new SqlParameter("@sal",
        SqlDbType.SmallMoney, 20));
    myCommand.Parameters["@sal"].Value = esal.Value;
    myCommand.ActiveConnection.Open();
```

```

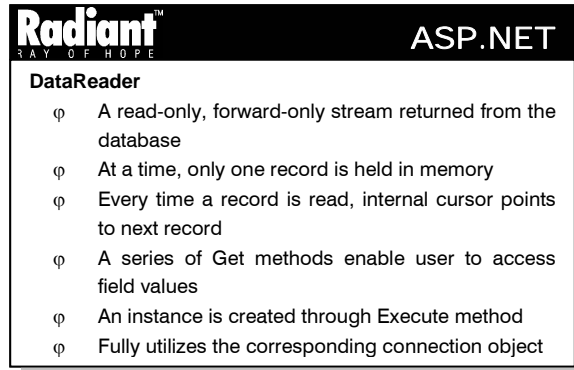
        try
        {
            myCommand.ExecuteNonQuery();
            lb.Text = "Record Added";
        }
        catch (SQLException e)
        {
            if (e.Number == 2627)
                lb.Text = "ERROR: A record already exists with
                    the same primary key";
            else
                lb.Text = "ERROR: Could not add record, please ensure
                    the fields are correctly filled out";
        }
        myCommand.ActiveConnection.Close();
    }
</script>
<body style="font: 10pt verdana">
    <form runat="server">
        <h3>Add a New Employee:</h3>
        ID: <input type="text" id="eid" value="0" runat="server"> <br>
        Name: <input type="text" id="ename" value="" runat="server"> <br>
        Sal: <input type="text" id="esal" value="0" runat="server"><br>
        <input type="submit" OnServerClick="AddEmp" value="Add Employee"
            runat="server">
        <asp:label id="lb" runat="server" />
    </form>
</body>
</html>

```



The above example receives the employee ID, name and salary from the user to add a new record to the **emp** table of the **master** database. The connection object is created when the page is loaded. When the user enters the required information and clicks the "Add Employee" button, its OnServerClick event is fired. This event invokes the **AddEmp** method. It checks if any of the input values is empty. If so, it prints out an appropriate message and returns without inserting the record. If not, a command object is created using the connection defined in the **Page\_Load** event. Values are passed to the object using the **Parameters** collection. Then the command is executed and the record is added to the table.

## 10.5 DataReader



Data can be retrieved from the database and held in memory for usage by the application. But when a large amount of data is retrieved, maintaining the memory used by the data for a long time will be inefficient. In this case it is useful to make use of the DataReader. The DataReader can also be used when the user has to iterate over the stream of data retrieved from the database.

The DataReader is a read-only, forward-only stream returned from the database. At a time, only one record is held in memory. Every time a record is read, the internal cursor will automatically point to the next record. A series of Get methods provided by the DataReader enable the user to access the field values.

An instance is created for **ADODataReader** through the **Execute** method of the **ADOCCommand** class and not through its constructor. Further, when the ADODataReader is being used, the corresponding ADOConnection is fully utilized by it. So, no other operation (except the Close operation) can be performed using the connection until the ADODataReader is closed. The same holds true for **SQLDataReader**.

The properties of both the classes are as follows:

- **FieldCount** - Represents the number of fields in the current record
- **HasMoreResults** - Indicates whether or not there are more results to be read
- **HasMoreRows** - Indicates whether or not there are more rows to be retrieved
- **IsClosed** - Indicates whether or not the DataReader is closed
- **Item** - Indicates the column value in its native format

The ADODataReader class contains one more property named **RowFetchCount** that indicates the number of rows to be retrieved at one time.

The following methods are common to both ADODataReader and SqlDataReader:

- **Close** - Closes the data reader object
- **Equals** - Determines whether the specified Object is the same instance as the current Object
- **GetBoolean** - Returns the value of the specified column as a boolean
- **GetByte** - Returns the value of the specified column as a byte
- **GetBytes** - Returns the value of the specified column as a byte array
- **GetChar** - Returns the value of the specified column as a character
- **GetChars** - Returns the value of the specified column as a character array
- **GetDataTypeName** - Returns the name of the back-end data type
- **GetDateTime** - Returns the value of the specified column as a DateTime object
- **GetDecimal** - Returns the value of the specified column as a Decimal object
- **GetDouble** - Returns the value of the specified column as a double-precision floating point number
- **GetFieldType** - Returns the data type of the object
- **GetFloat** - Returns the value of the specified column as a single-precision floating point number
- **GetHashCode** - Serves as a hash function for a particular type
- **GetInt16** - Returns the value of the specified column as a 16-bit signed integer
- **GetInt32** - Returns the value of the specified column as a 32-bit signed integer
- **GetInt64** - Returns the value of the specified column as a 64-bit signed integer
- **GetName** - Returns the name of the specified column
- **GetOrdinal** - Returns the column ordinal, given the name of the column
- **GetString** - Returns the value of the specified column as a string
- **GetType** - Returns the type of the object
- **GetValue** - Returns the value of the specified column in its native format
- **GetValues** - Returns all the attribute fields in the collection for the current record
- **IsNull** - Checks for non-existent values
- **NextResult** - When reading the results of batch SQL statements, this method advances the data reader to the next result
- **Read** - Advances the data reader to the next record
- **ToString** - Returns the String representing the current Object

The important methods specific to the SqlDataReader class are:

- **GetSQLBinary** - Returns the SQLBinary value of the specified column
- **GetSQLBit** - Returns the SQLBit value of the specified column
- **GetSQLGuid** - Returns the SQLGuid value of the specified column
- **GetSQLMoney** - Returns the SQLMoney value of the specified column
- **GetSQLNumeric** - Returns the SQLNumeric value of the specified column
- **GetSQLSingle** - Returns the SQLSingle value of the specified column

The important methods specific to **ADODataReader** class are:

- **GetSByte** - Returns the value of the specified column as a SByte
- **GetGuid** - Returns the value of the specified column as a globally-unique identifier.



### Practice 10.3

---

The following example uses DataReader to retrieve and display the records of the table **emp** in the **master** database.

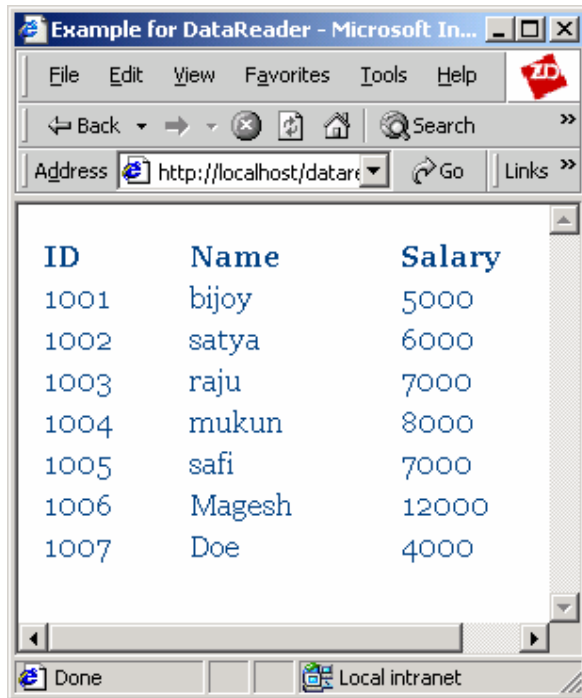
```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SQL" %>

<html>
<title> Example for DataReader </title>
<script language="C#" runat="server">

SQLConnection myConnection;
protected void Page_Load(Object Src, EventArgs E)
{
    myConnection = new SQLConnection(
        "server=localhost;uid=sa;pwd=;database=master");
    SqlCommand myCommand = new SqlCommand("select distinct * from
        emp", myConnection);
    myConnection.Open();
    SqlDataReader dr;
    myCommand.Execute(out dr);
    Response.Write("<Table width=300>");
    Response.Write("<TR>");
    Response.Write("<TD>");
    Response.Write("<B>ID </B>");
    Response.Write("</TD>");
    Response.Write("<TD>");
    Response.Write("<B>Name </B>");
    Response.Write("</TD>");
    Response.Write("<TD>");
    Response.Write("<B>Salary </B>");
    Response.Write("</TD>");
    Response.Write("</TR>");

    while (dr.Read())
    {
        Response.Write("<TR>");
        Response.Write("<TD>");
        Response.Write(dr["eno"].ToString() + " ");
        Response.Write("</TD>");
        Response.Write("<TD>");
        Response.Write(dr["ename"].ToString() + " ");
        Response.Write("</TD>");
        Response.Write("<TD>");
        Response.Write(dr["sal"].ToString() + " ");
        Response.Write("</TD>");
        Response.Write("</TR>");
    }
    myConnection.Close();
}
</script>
```

```
</html>
```



The above example establishes a connection with the **master** database. It then executes a select command to retrieve rows from the **emp** table and stores the retrieved rows in the SqlDataReader **dr**. Then it reads each row from it and displays in a table.

## 10.6 Short Summary

- The Managed Provider acts as an intermediate between the client and the data source
- ADO.NET provides two types of providers: ADO and SQL
- The parameter object represents a parameter or argument associated with a Command object based on a parameterized query or stored procedure
- The property object contains the metadata and settings for a property in an ADO data source
- When the ADO encounters a situation that it cannot handle, it creates an instance of the ADOException class
- A connection object represents a unique session with a data source
- The Command object can be used to request any type of operation from the provider, if the provider understands the command string properly
- The DataReader is a read-only, forward-only stream returned from the database. It is useful when the user has to iterate over the stream of data retrieved from the database

## 10.7 Brain Storm

1. What is a Managed Provider?
2. What is the role of the property object in ADO?
3. What are the kinds of commands that can be executed using the Command object?
4. What is a DataReader?
5. When an ADODataReader is open, can the corresponding Connection object be used to execute some other command?

❧

## Lecture 11

---

# ADO.NET - III

---

### Objectives

In this lecture you will learn the following

- + What is meant by DataSet?
- + Learning about DataTable
- + Knowing about DataView



---

## Coverage Plan

---

### **Lecture 11**

- 11.1 Snap Shot
- 11.2 DataSet
- 11.3 DataTable
- 11.4 DataView
- 11.5 Short Summary
- 11.6 Brain Storm

## 11.1 Snap Shot

This session introduces the user to DataSet, DataRelation, DataTable and DataView.

The DataSet is designed to handle the actual data from a data store and provides access to multiple tables, rows and columns. Each DataSet can contain multiple tables and maintain the relationship between them. The advantage of the DataSet is that since it is designed for disconnected data, multiple tables can be simultaneously passed around the tiers of a Web application, along with the relationships.

DataView is a custom view of data table, whose prime purpose is to aid in data binding. DataView is the equivalent of ADO RecordSet.

**Note:** A DataSet is not a RecordSet. In terms of analogies, a DataView is more like a RecordSet.

## 11.2 DataSet

**Radiant™**  
RAY OF HOPE

**ASP.NET**

**Data set**

- ☐ Contains any number of data tables
- ☐ Constitutes a “disconnected” view of database data
- ☐ Enables greater scalability
- ☐ Resides in System.Data namespace

The centerpiece of any software solution using ADO.NET is the DataSet. A DataSet is an in-memory copy of the database data. A DataSet contains any number of data tables, each of which typically corresponds to a database table or view. A DataSet constitutes a “disconnected” view of the database data. That is, it exists in memory without an active connection to a database containing the corresponding tables or views. This disconnected architecture enables greater scalability by only using database server resources when reading or writing from the database.

The DataSet class resides in **System.Data** namespace. The DataSet object model is shown in Figure 11.1.

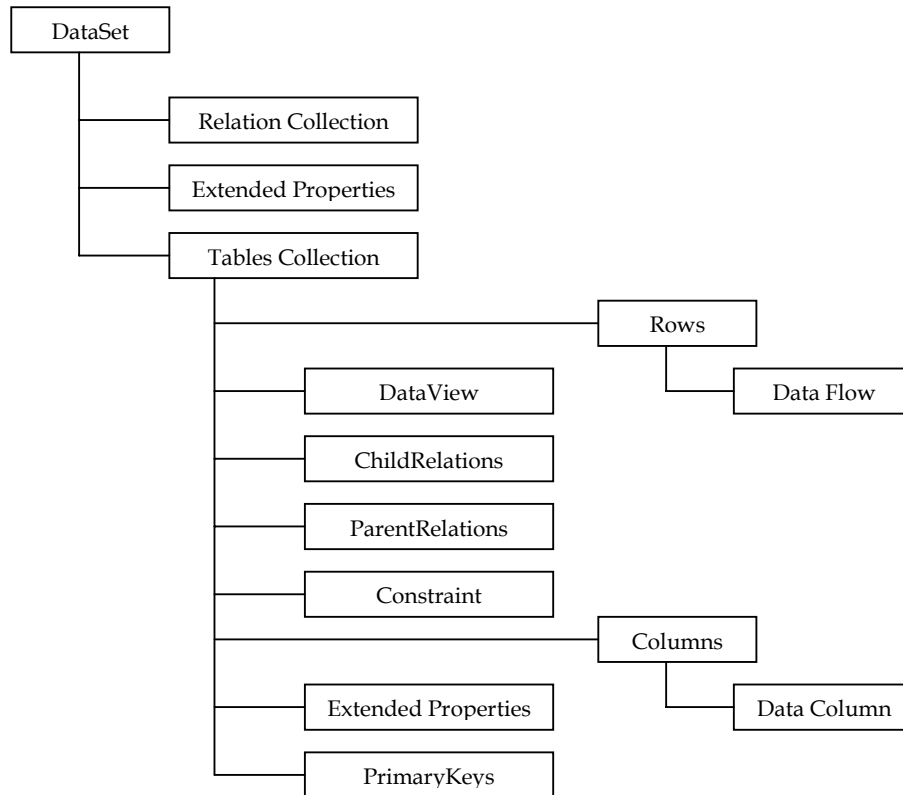


Figure 11.1

### TablesCollection object

An ADO.NET DataSet is a collection of one or more tables represented by DataTable objects. The TablesCollection object contains all the DataTable objects in a DataSet. A DataTable is defined by System.Data and represents a single table of memory-resident data. It contains a collection of columns represented by the ColumnsCollection object, which defines the schema and rows of the table. It also contains a collection of rows represented by the RowsCollection object, which orders the data in the table. Along with the current state, a DataTable object retains its original state and tracks all changes that occur to the data. The DataSet is able to persist and reload its contents through XML.

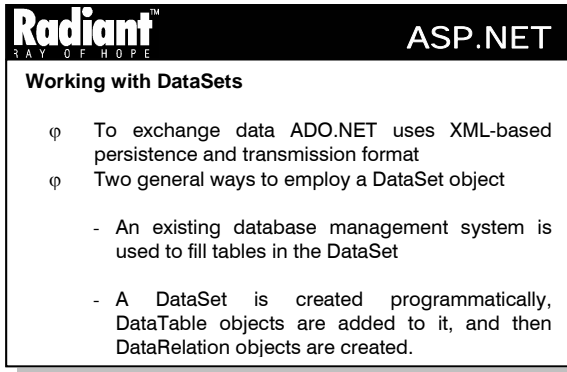
### RelationsCollection object

A typical DataSet contains relationships contained by the RelationsCollection object. A relationship, represented by the DataRelation object, associates rows in one data table with rows in another data table. It is analogous to the foreign-key relationship in a relational database. A DataRelation identifies matching columns in two tables of a DataSet. Relationships that enable navigation from one table to another within a DataSet. The essential elements of a DataRelation are the name of the relationship, the two tables being related, and the primary key and foreign key columns in the tables. Relationships can also be built with more than one column per table, with an array of DataColumn objects for the primary and foreign keys. When a DataRelation is created, ADO.NET verifies if the relationship can be established. Once ADO.NET adds a relationship to the RelationsCollection, it disallows any changes that would invalidate the relationship.

## ExtendedProperties

The ExtendedProperties object contains customized user information, such as a password or the time when data should be refreshed for a DataSet object.

## Working with DataSets



At run time, data will be passed from the database to a middle-tier business object and then down to the user interface. To accommodate the exchange of data, ADO.NET uses an XML-based persistence and transmission format. That is, to transmit data from one tier to another, an ADO.NET solution expresses the in-memory data (the DataSet) as XML and then sends the XML to the other component.

There are two general ways to employ a DataSet object.

- An existing database management system, such as SQL Server can be used to fill tables in the DataSet. In this method, one SQLDataSetCommand can be used per table to fill the DataTable object with data. However, the DataRelation objects should be created between each table
- A DataSet can be programmatically created, DataTable objects can be added to it, and then DataRelation objects can be created to link each table

On the client application, the data is displayed using a combination of controls, such as the DataGrid. The user can add, delete, or edit the data. After the user finishes working with the data, the DataSet is again converted into an XML document for transmitting back to the server component. DataSet, the middle tier component, is created using an adapter. The DataSet is then converted into an XML document for transport back to the requester.

The DataSet stores data using COM+ types. The COM+ decimal type allows a maximum of 28 significant digits, while the SQL decimal type allows 38 significant digits. Currently, the FillDataSet method throws an exception if it encounters a valid SQL decimal with more than 28 significant digits. Currently there are currently no ways to get a SQL decimal of greater than 28 significant digits into a DataSet object. If the application requires the added precision, a SQLDataReader object can be used. The GetSQLNumeric method can then be called to get the SQL decimal value.

Figure 11.2 shows the working of DataSet in ADO.NET solution.

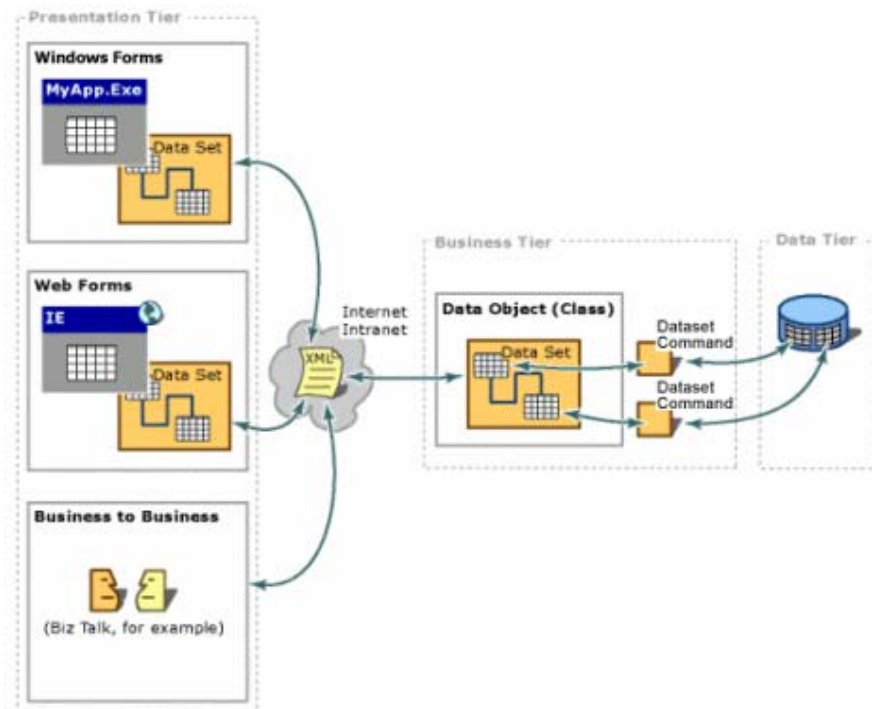


Figure 11.2

### Constructors of DataSet

There are two constructors for a DataSet.

The first constructor initializes a new instance of the DataSet class, as shown below:

```
public DataSet();
```

The second constructor initializes a new instance of a DataSet class with the given name, as given below:

```
public DataSet(string);
```



### Practice 11.1

The following application performs a select query from a SQL database bounded to a DataGrid.

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

<html>
<script language="C#" runat="server">
void Page_Load(Object Src, EventArgs E)
{
    SqlConnection con = new SqlConnection
        ("server=localhost;uid=sa;pwd=");
```

```

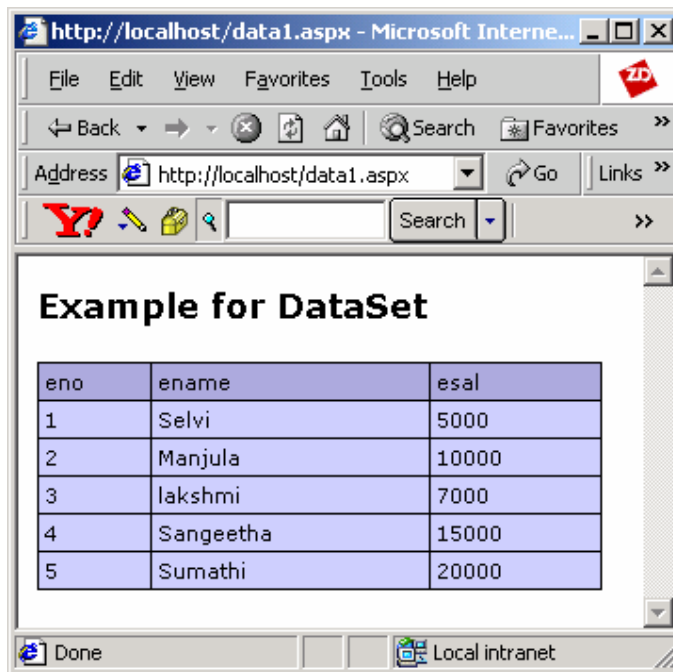
SQLDataSetCommand com = new SQLDataSetCommand("select * from
        emptab", con);
DataSet ds = new DataSet();
    com.FillDataSet(ds, "emptab");
d.DataSource=ds.Tables["emptab"].DefaultView;
    d.DataBind();
}
</script>
<body>

    <h3><font face="Verdana">Example for DataSet</font></h3>

    <ASP:DataGrid id="d" runat="server"
        Width="300"
        BackColor="#ccccff"
        BorderColor="black"
        CellPadding=3
        CellSpacing="0"
        Font-Name="Verdana"
        Font-Size="8pt"
        HeaderStyle-BackColor="#aaaadd"
        />

</body>
</html>

```



In the above example, the connection object **con** is created in the **Page\_Load** event. A **SQLDataSetCommand** object **com** is also created with the query to retrieve all the records from the **emptab** table. The command is executed and the **DataSet ds** is filled with the resulting rows using the **FillDataSet()** method. Then, the **DataSource** property of the **DataGrid** is set to the **DataSet**. Finally, the **DataBind()** method is called to populate the **DataGrid** with the values.

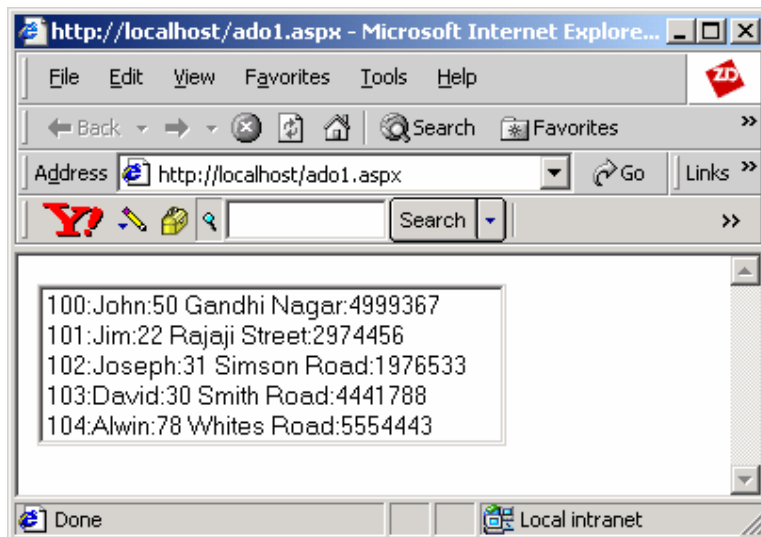


## Practice 11.2

The following application executes a select query from a database using ADOConnection and ADODataSetCommand classes and displays the data in a listbox.

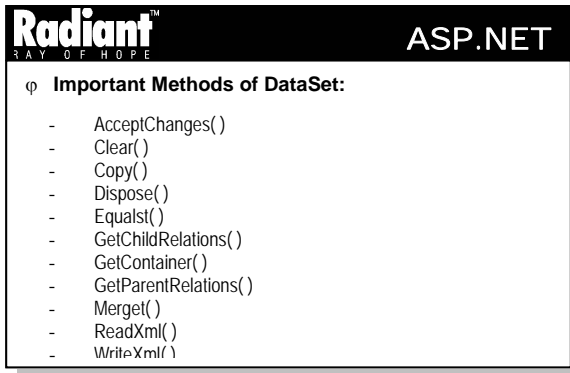
```
<%@ Import Namespace="System" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.ADO" %>

<script language="C#" runat="server">
void Page_Load(Object Src, EventArgs E )
{
    string s= "Provider=Microsoft.Jet.OLEDB.4.0;Data Source =
                E:/Inetpub/wwwroot/emp.mdb " ;
    ADOConnection con = new ADOConnection(s);
    ADODataSetCommand com = new ADODataSetCommand("select * from
                customer", con);
    DataSet ds = new DataSet();
    com.FillDataSet(ds,"customer");
    DataTable dt = ds.Tables["customer"];
    foreach(DataRow dr in dt.Rows)
    {
        l.Items.Add(dr["CusID"] + ":" + dr["CusName"] + ":" +
                +dr["CusAdd"]+":" +dr["CusPho"]);
    }
}
</script>
<asp:Listbox id=l runat=server/>
</html>
```



In the above example, a connection with the database has been created using `ADOConnection`. The Query has been passed to `ADOSetCommand`. The `DataSet` is populated with the results from the query using the **FillDataSet** method. The result is then viewed in a listbox using the **foreach** statement. The result can also be viewed by binding it to a `DataGrid`.

### Methods of DataSet



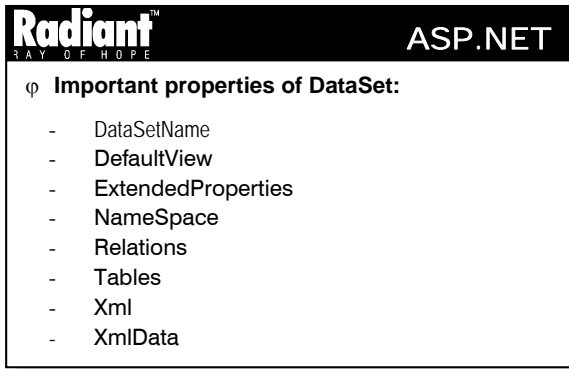
The various methods of `DataSet` are discussed below:

- **AcceptChanges()** - Commits all the changes made to this `DataSet` since it was loaded or the last time `AcceptChanges` was called
- **Clear()** - Clears the `DataSet` of any data by removing all rows in all tables
- **Clone()** - Clones the structure of the `DataSet`, including all `DataTable` schemas, relations, and constraints
- **Copy()** - Copies both the structure and data for this `DataSet`
- **Dispose()** - Disposes of the Component
- **Equals()** - Determines whether the specified Object is the same instance as the current Object
- **GetChanges()** - Returns a copy of the `DataSet` containing all changes made to it since it was last loaded, or since `AcceptChanges` was called
- **GetChildRelations()** - Gets the collection of child relations which belong to a specified table
- **GetContainer()** - Returns the `IContainer` that contains the Component
- **GetHashCode()** - Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table
- **GetParentRelations()** - Gets the collection of parent relations which belong to a specified table
- **GetType()** - Gets the Type of the Object
- **Merge()** - Merges this `DataSet` with a specified `DataSet`
- **ReadXml()** - Reads XML schema and data into the `DataSet`
- **ReadXmlData()** - Reads data from the specified XML source and maps the data into the `DataSet` object's schema
- **ReadXmlSchema()** - Reads an XML schema into the `DataSet`
- **ToString()** - Returns a String that represents the current Object
- **WriteXml()** - Writes the current schema and data for the `DataSet[cref, System.Data]` to an XML document
- **WriteXmlData()** - Overloaded. Writes the data contained in the `DataSet` to an XML document



- **WriteXmlSchema()** - Overloaded. Writes the DataSet structure as an XML schema

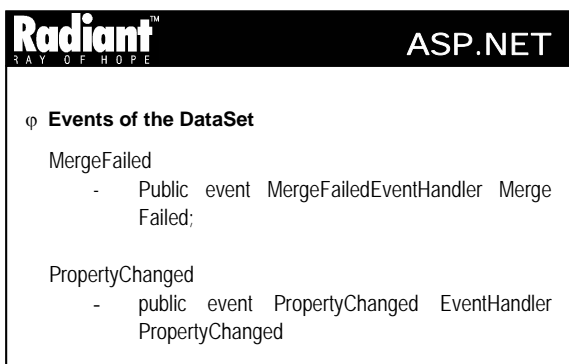
### Properties of DataSet



The properties of DataSet are as follows:

- **CaseSensitive** - gets or sets a value indicating whether string comparisons within datatable objects are case-sensitive
- **Datasetname** - gets or sets the name of the dataset
- **Defaultview** - gets a custom view of the data contained by the dataset, one that allows filtering, searching, and navigating through the custom data view
- **Extendedproperties** - gets the collection of custom user information
- **Haserrors** - gets a value indicating whether there are errors in any of the rows in any of the tables of the dataset
- **Locale** - gets or sets the locale information used to compare strings within the table
- **Namespace** - gets or sets the namespace of the dataset
- **Relations** - get the collection of relations that link tables and allow navigation from parent tables to child tables
- **Tables** - gets the collection of tables contained in the dataset
- **Xml** - gets or sets the xml data for the dataset
- **Xmldata** - gets or sets the xml data for the dataset
- **Xmlschema** - gets or sets the xml schema of the DataSet

### Events of DataSet



The various events of the DataSet are given below:

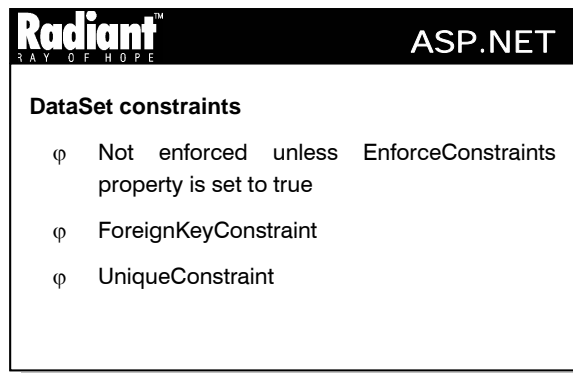
**MergeFailed** - Occurs when a target and source DataRow have the same primary key value, and EnforceConstraints is set to true. Its syntax is as follows:

```
public event MergeFailedEventHandler MergeFailed;
```

**PropertyChanged** - Occurs when a property value changes. Its syntax is as follows:

```
public event PropertyChangedEventHandler PropertyChanged;
```

### DataSet Constraints



The DataSet constraints are as follows:

#### ForeignKeyConstraint

The ForeignKeyConstraint represents an action restriction enforced on a set of columns in a primary key/foreign key relationship when a value or row is either deleted or updated. In a parent/child relationship between two tables, deleting a value from the parent table can affect the child rows in one of the following ways:

- The child rows can also be deleted (a cascading action)
- The values in the child column (or columns) can be set to null values
- The values in the child column (or columns) can be set to default values
- An exception can be thrown

#### UniqueConstraint

The UniqueConstraint can be enforced to a DataTable when it is essential that all the values of a column must be unique. It can be added either to a single column or to an array of columns of the same DataTable.

Constraints are not enforced unless the EnforceConstraints property is set to true. When a DataSet is merged with a second DataSet, constraints are not enforced until all merges are completed.

## DataRelation

Radiant™

ASP.NET

**DataRelation**

- ☐ Represents a parent/child relationship between two tables
- ☐ Associates rows in one table to rows in another
- ☐ Identifies matching columns in two tables of a DataSet
- ☐ DataSet can contain either related or unrelated data
- ☐ DataSet can handle both hierarchical and foreign key relationship

The DataRelation represents a parent/child relationship between two tables. The DataSet represents a complete set of data including related tables, constraints, and relationships among the tables. A typical dataset contains relationships contained by the RelationsCollection object. A relationship, represented by the DataRelation object, associates rows in one table to the rows in another table. It is similar to the foreign-key relationship in a relational database. A DataRelation identifies matching columns in two tables of a DataSet.

A DataSet can contain either related or unrelated data. It can be thought of as a document of Data. In fact, an XML data document is similar to it except for the fact that it is based on a hierarchical paradigm. Since data is often stored in relational databases, the DataSet can handle both hierarchical relationships and foreign key relationships. Relationships can also have different types of enforcement. By default, deletions and updates are cascaded. A DataSet contains a Relations collection. It is easy to add a relationship to this collection using the column or columns (in a multi-column key) of the related tables.



### Practice 11.3

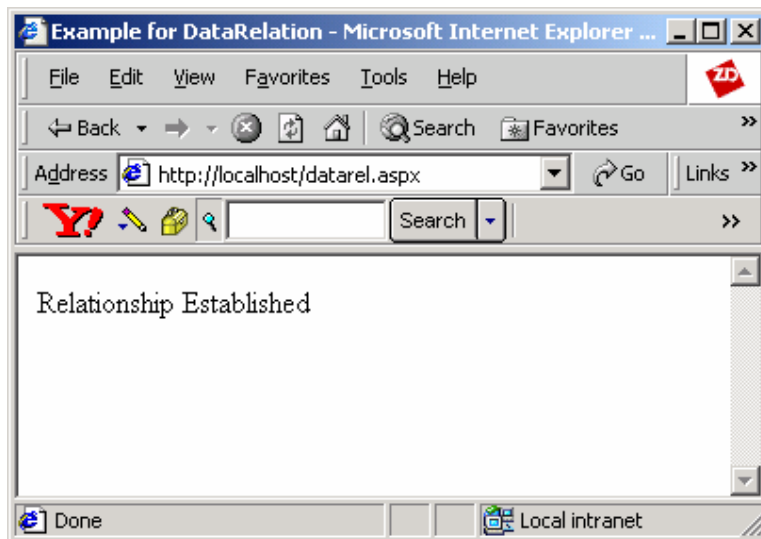
The example given below presumes that two DataTable objects exist in the DataSet. Both tables have a column named "CustomerID" which serves as the link between the two tables. The example adds a single DataRelation to the Relations collection of the DataSet object. The first argument ("CustOrders") specifies the name of the relationship. The second and third arguments are the DataColumn objects that link the two tables.

```
<%@ Import Namespace="System" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SQL" %>
<html>
<title> Example for DataRelation </title>
<script language="C#" runat="server">
public DataSet ds;
void Page_Load(Object Src, EventArgs E )
{
    string s="server=localhost;uid=sa;pwd='';database=master";
    SqlConnection con = new SqlConnection(s);
    SQLDataSetCommand com1 = new SQLDataSetCommand("select * from
        customers", con);
    SQLDataSetCommand com2 = new SQLDataSetCommand("select * from
```

```

        orders", con);
    ds = new DataSet();
    com1.FillDataSet(ds,"Customers");
    com2.FillDataSet(ds,"Orders");
    DataRelation dr;
    dr = new DataRelation( "CustOrders",ds.Tables["Customers"].
        Columns["CustomerId"],
        ds.Tables["Orders"].Columns["CustomerId"]);
    ds.Relations.Add(dr);
    Response.Write("Relationship Established");
}
</script>
</html>

```



The example above constructs an instance **ds** of a DataSet and fills data into it from two SQLDataSetCommand objects **com1** and **com2**. Then a DataRelation **dr** is added to the Relations collection specifying its name, and the appropriate DataColumn objects as arguments. It adds a relation between the **CustomerId** Key on the **Customers** table and the **CustomerId** foreign key on the **Orders** table in the DataSet.

Data can be retrieved from the relationship defined in the example above using the lines:


```

foreach (DataRow Customer in ds.Tables["Customers"].Rows) {
    Response.Write("Customer: " + Customer["ContactName"].ToString());
    foreach (DataRow Order in Customer.GetChildRows(
        ds.Relations["CustOrders"])) {
        Response.Write("Order #" + Order["OrderId"].ToString());
    }
}

```

A primary function of the `DataRelation` is to allow navigation from one table to another within the `DataSet`. In practice, this allows us to retrieve all the related `DataRow` objects in one table when given a single `DataRow` from a related table. The above example gives a `DataRow` from the `Customers` table. All the orders for a particular customer can be retrieved from the `Orders` table.

### 11.3 DataTable



**ASP.NET**

**DataTable class**

- ⊕ Present in `System.Data` namespace
- ⊕ Represents a table of in-memory data
- ⊕ Central object in ADO.NET library
- ⊕ Contains a collection of `Constraint` object

The `DataTable` class is present in the `System.Data` namespace. It represents a table of in-memory data. The `DataTable` is a central object in the ADO.NET library. Other objects which use the `DataTable` include the `DataSet` and the `DataView`. The `DataTable` contains a collection of `Constraint` objects that can be used to ensure the integrity of the data.

#### Syntax

```
Public Class DataTable Inherits Component Implements IListSource,
ISupportInitialize
```



#### Practice 11.4

This program displays the properties of a table such as its column names, types and column properties. It displays the result in a listbox.

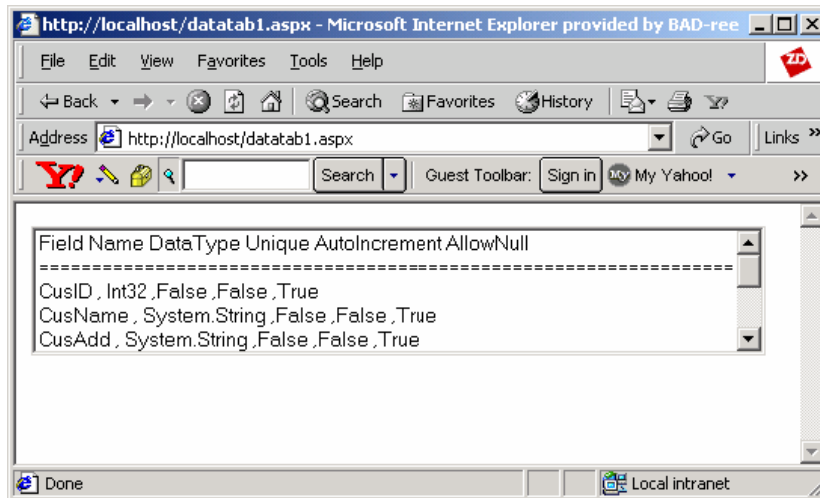
```
<%@ Import Namespace="System" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.ADO" %>

<script language="C#" runat="server">
void Page_Load(Object Src, EventArgs E )
{
    string s= "Provider=Microsoft.Jet.OLEDB.4.0;Data Source =
                E:/Inetpub/wwwroot/emp.mdb " ;
    ADOConnection con = new ADOConnection(s);
    ADODataSetCommand com = new ADODataSetCommand("select * from customer", con);
    DataSet ds = new DataSet();
    com.FillDataSet(ds, "customer");
    DataTable dt = ds.Tables[0];
    l.Items.Add("Field Name DataType Unique AutoIncrement AllowNull");
    l.Items.Add("=====");
    foreach( DataColumn dc in dt.Columns )
    {
```

```


        l.Items.Add(dc.ColumnName+" , "+dc.DataType + " ," +dc.Unique
                    +" ," +dc.AutoIncrement+" ," +dc.AllowNull );
    }
    foreach(DataRow dr in dt.Rows)
    {
        l.Items.Add(dr["CusID"] + ":" + dr["CusName"] + ":" +
                    +dr["CusAdd"]+" ":" +dr["CusPho"]);
    }
}
</script>
<asp:Listbox id=l runat=server/>
</html>

```



The example above creates a connection with database using the `ADOConnection` object `con` and passes a query in an `ADOSetCommand` object `com`. It then populates the `DataSet ds` with the results from the query using the `FillDataSet` method. Then it displays the table properties such as its column names, types and column properties in the listbox `l` using the `foreach` statement. The `Unique`, `AutoIncrement` and `AllowNull` properties return a boolean value.

## 11.4 DataView



**ASP.NET**

**DataView**

- ☐ Represents a databindable, customized view of a DataTable
- ☐ Allows data binding on both Windows forms and Web Forms
- ☐ Can be customized to present a subset of data from DataTable
- ☐ Components of DataRow of a DataTable within a given DataView

The DataView represents a databindable, customized view of a DataTable for sorting, filtering, searching, editing, and navigation. A major function of DataView is to allow data binding on both Windows Forms and Web Forms. A DataView can be customized to present a subset of data from the DataTable. This capability allows to have two controls bound to the same DataTable, but showing different versions of the data. For example, one control may be bound to a DataView showing all the rows in the table, while the second control may be configured to display only the rows that have been deleted from the DataTable. The DataTable also has a DefaultView property, which returns the default DataView for the table.

The DataRow of a DataTable within a given DataView have three components: what the data was (the original state), what the data is now (the current state), and what the value of the data is going to be (the proposed state). When changes are made to the data, it affects the current state or proposed state, but its original state is left unaffected until it is changed explicitly by using the DataRow.AcceptChanges method.

### Syntax

```
Public Class DataView Inherits Component Implements IList,
ICollection, IEnumerable, _ ISortedList, IIndexedlist, ITypedList
```



### Practice 11.5

The following example creates a single DataTable with three columns and five rows using DataView.

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

<html>
<script language="C#" runat="server">
void Page_Load(Object Src, EventArgs E)
{
    SqlConnection con = new SqlConnection(
        "server=localhost;uid=sa;pwd=");
    SQLDataSetCommand com = new SQLDataSetCommand("select * from
        emptab", con);
    DataSet ds = new DataSet();
    com.FillDataSet(ds, "emptab");
    DataView dv = new DataView(ds.Tables["emptab"]);
    s.InnerHtml = dv.Table.TableName;
}
```

```

        d.DataSource = dv;
        d.DataBind();
    }

</script>

<body>

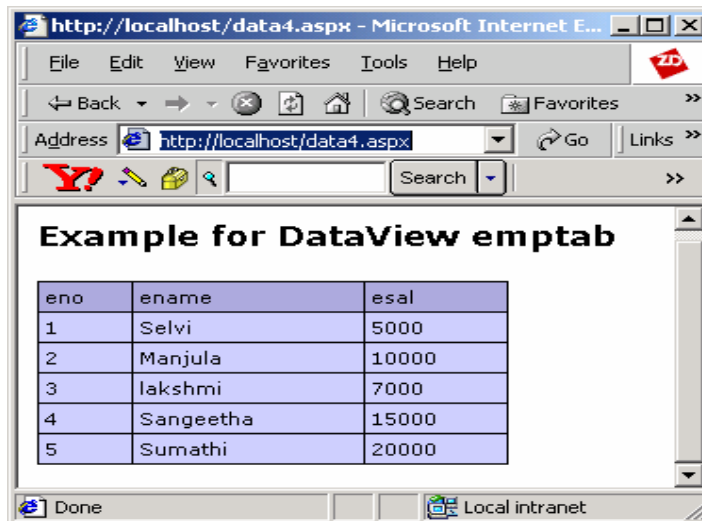
    <h3><font face="Verdana">Example for DataView <span runat="server"
        id="s" /></font></h3>

    <ASP:DataGrid id="d" runat="server"
        Width="250"
        BackColor="#ccccff"
        BorderColor="black"
        CellPadding=3
        Font-Name="Verdana"
        Font-Size="8pt"
        HeaderStyle-BackColor="#aaaadd"
    />

</body>
</html>

```

A `SqlConnection` object `con` is created to perform a select query on the **emptab** table of the SQL database. A `SqlCommand` object `com` that contains the query statement is then constructed. A `DataSet` object is created and populated with the results from the query through the `FillDataSet` method. The `DataView` `dv` represents a subset of data from the `DataTable`. Finally, the output is displayed by binding the `DataView` with the `DataGrid` `d`.



## 11.5 Short Summary



- The DataSet is designed to handle the actual data from a data store and provides access to multiple tables, rows and columns
- The DataSet class resides in System.Data package
- The TableCollection object, RelationsCollection object and ExtendedProperties are the three major parts of the DataSet object model
- The DataRelation represents a parent/child relationship between two tables
- The DataTable contains a collection of Constraint objects that can be used to ensure the integrity of the data
- The DataView represents a databindable, customized view of a DataTable for sorting, filtering, searching, editing, and navigation. A major function of the DataView is to allow data binding on both Windows Forms and Web Forms

### 11.6 Brain Storm

1. What is a DataSet?
2. Name the two events of DataSet.
3. How can two DataTables be related?
4. Differentiate between DataTable and DataView.
5. What is the main use of DataView?

☺☺☺

## Lecture 12

---

# Data Aware Control

---

### Objectives

In this lecture you will learn the following

- + Learning about Databinder
- + Knowing about Repeater Control

---

## Coverage Plan

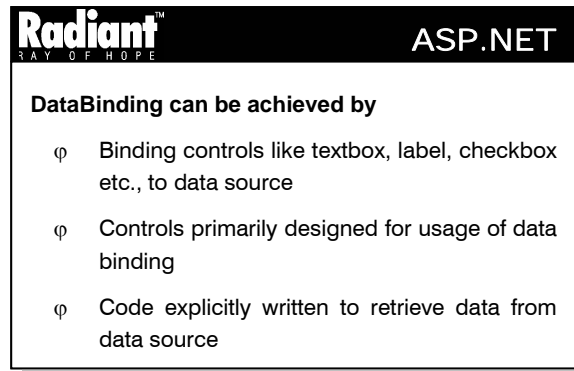
---

<b>Lecture 12</b>
12.1 Snap Shot
12.2 Databinder
12.3 Repeater Control
12.4 Short Summary
12.5 Brain Storm

## 12.1 Snap Shot

This session teaches the techniques that can be employed to bind ASP.NET controls to any data source. It also introduces the Repeater control that is a data-bound control.

In the previous sessions, we have learnt how to use ASP.NET controls. This session deals with binding the controls to the data source, that is, associating the control with the datasource so that the data for the control is supplied by the data source. In ASP.NET, the data source can be a database, an XML file, a control etc. The Web Forms page enables the user to bind any control's property to the information in any kind of data store.



DataBinding can be achieved in one of the following ways:

- Controls like textbox, label, checkbox etc., can be bound to the data source
- Controls like DataGrid, DataList etc., that are primarily designed for the usage of data binding can be used
- Programmer can write code explicitly to retrieve the data from the data source and display it in the controls

The Web Forms pages access data in the form of classes that expose data as properties and define methods for updating the underlying data source. The simple controls like radio button, checkbox etc., bind to a single value. They can bind to the public property of the page or any of its controls. Complex controls like Repeater, DataList and DataGrid can bind to any structure that implements the ICollection interface.

A control can be bound to the data by using either of the following steps:

- DataSource property of a complex list control is set to refer to a data class
- A property of the control is bound using a data binding expression which when evaluated provides a value that is loaded into the bound property

The databinding expression has the following syntax:

```
<% expression %>
```

where *expression* is the data binding expression which is a reference to any publicly-available property. The data binding expression is delimited by the <% and %> tags. Suppose the **Text** property of a Label has to be bound to the **deptName** field of the table **emp** then the data binding expression for it appears as follows:

```
<asp: Label id="dept" Text='<% emp(0).deptName %>' runat=server>
```

The expression is evaluated only when the **DataBind** method is called. This method is generally called at two instances: when the page is loaded or when an event-handling method has changed the data set. The DataBind method can be called for each control or for the whole page. When it is called for the whole page, it is cascaded down to each control on the page. Similarly, when it is called on a parent control, it is cascaded to all of its child controls.

## 12.2 Databinder

Radiant™

ASP.NET

**Databinder**

- ☐ Makes code simpler and more readable
- ☐ Eval()
  - Static method that evaluates data binding expressions against Container object at runtime
  - Optionally formats result as a string
  - Useful when data binding against controls in a templated list

Syntax = Eval(Object *container*, string *expression*, string *format*) n not be used

The DataBinder class is provided by ASP.NET to make code simpler and more readable. The static method Eval() of this class evaluates data binding expressions against the **Container** object at runtime and optionally formats the result as a string. This method is particularly useful when data binding against controls in a templated list. It can take one of the two forms:

```
Eval(object container, string expression)
Eval(object container, string expression, string format)
```

In the above forms, the first argument is the object reference against which the expression is evaluated. For the controls Repeater, DataList and DataGrid, the value of this parameter should be **container.dataitem**. If the binding is done against the page then the parameter should take the value **page**. The second argument is the navigation path from the *container* to the property value to be placed in the bound control property. It should be a string of property or field names separated by dots. The third argument is a format string used to convert the result of evaluating the expression into a string type to be displayed by the requesting browser.

### Binding to Non-DataBase Datasources

Radiant™

ASP.NET

**ASP.NET data binding syntax supports binding to**

- ☐ Public variables
- ☐ Properties of the page
- ☐ Properties of other controls on the page

The ASP.NET data binding syntax supports binding to public variables, properties of the page and properties of other controls on the page. The following example illustrates binding to public variables and simple properties on the page.

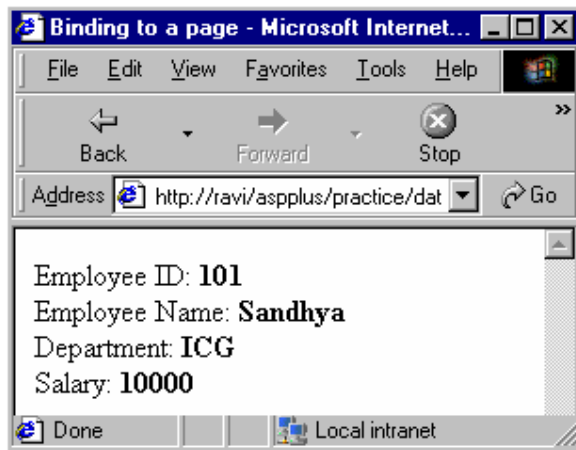


### Practice 12.1

The following example binds four properties to display the details of an employee.

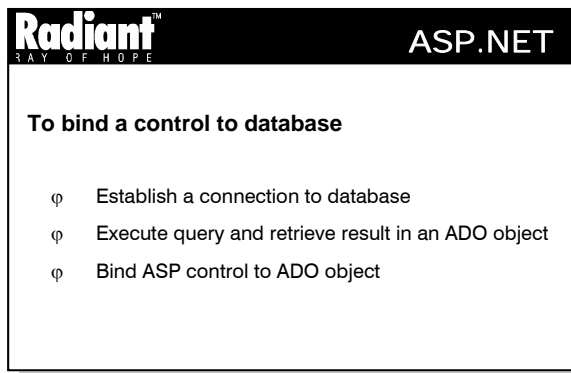
```
<html>
<head>
<script language="C#" runat="server">
void Page_Load(Object sender, EventArgs e) {
    Page.DataBind();
}
int EmpID {
    get {
        return 101;
    }
}
string EmpName{
    get {
        return "Sandhya";
    }
}
string Dept{
    get {
        return "ICG";
    }
}
int Salary{
    get {
        return 10000;
    }
}
</script>
</head>
<body>
<form runat=server>
    Employee ID: <b><%# EmpID %></b><br>
    Employee Name: <b><%# EmpName %></b><br>
    Department: <b><%# Dept %></b><br>
    Salary: <b><%# Salary %></b>
</form>
</body>
</html>
```





The above example uses the databinding expression to bind four properties namely, EmpID, EmpName, Dept and Salary to the corresponding elements of the page. The **DataBind()** method is present in the Page\_Load event so that the data is displayed as soon as the page is loaded.

### Binding to a DataBase



The controls like ListBox, DropDownList, CheckBoxList and RadioButtonList can be used to display data from a database. They can display one field from the source and use another field as the item value. The **DataSource** property of the control should be set to the source of the data. The **DataTextField** property denotes the name of the field whose value is displayed in the list. The **DataValueField** property is the name of the field to be used for the **Value** property of each list item.



### Practice 12.2

The following example uses the **ListBox** control to display the list of publisher names from the **Publishers** table.

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SQL" %>
<title> Binding a List to the database </title>

<script language="C#" runat="server">
void Page_Load ( Object src, EventArgs e)
{
```

```

DataSet ds;
SqlConnection conn;
SQLDataSetCommand cmd;
    string connStr, query;

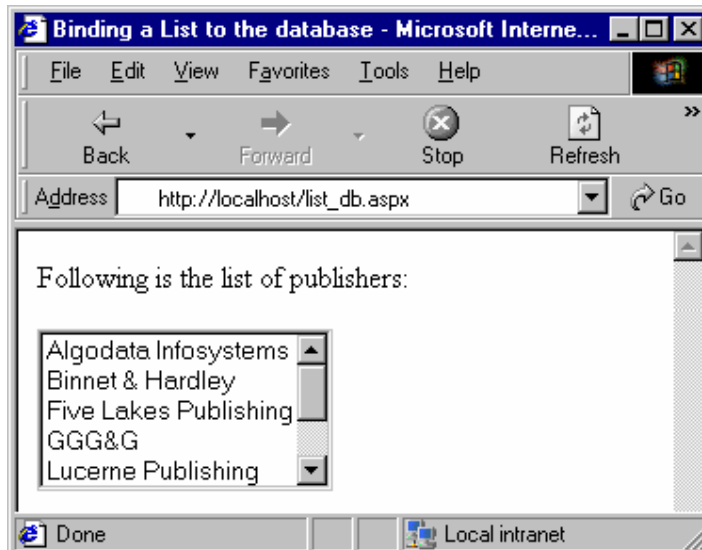
connStr="server=localhost;uid=sa;database=pubs";
    conn = new SqlConnection(connStr);

query="select distinct pub_name from Publishers";
cmd = new SQLDataSetCommand(query, conn);
ds = new DataSet();
cmd.FillDataSet(ds, "publishers");
Lb.DataSource= ds.Tables["publishers"].DefaultView;
Lb.DataTextField="pub_name";
Lb.DataBind();
}
</script>

<body>
<form runat=server>
<asp:Label id="lbl" text="Following is the list of publishers: "
    runat="server" />

<br><br>
<asp:ListBox id="Lb" runat="server"/>
</form>
</body>
</html>

```

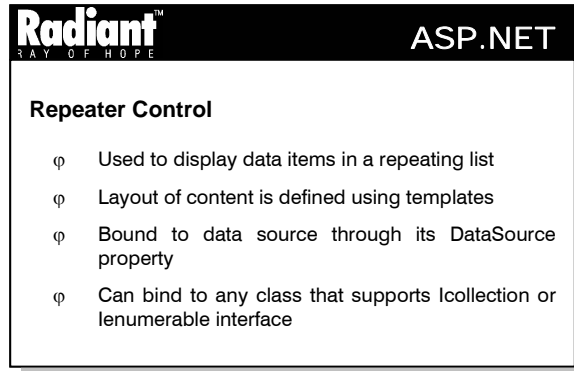


The above example binds the listbox to the database in the **Page\_Load** event. It creates a connection object named **conn** using the connection string **connStr**. A command object **cmd** is then created using this connection and the query string **query**. The command is executed and the results are retrieved in the



DataSet **ds**. Then the DataSet is linked to the listbox through the properties **DataSource** and **DataTextField**.

### 12.3 Repeater Control



The Repeater control is used to display the data items in a repeating list. The layout of the control's content is defined using templates.

A template is a set of HTML elements and controls that make up the layout for a particular portion of a control. Templates allow the programmer to specify both the contents and appearance of controls like the Repeater, DataGrid and DataList. Each of these controls support a slightly different set of templates that specify layouts for different portions of the control such as the header, footer, item and selected item. The type of template is specified as the NAME attribute of the template element. Templates can be created directly in the .aspx file containing the control which uses the template. Since the template also specifies the content to be displayed, the databinding expressions can be specified inside the template.

The Repeater control supports the following templates:

- **HeaderTemplate** – This defines the layout of the elements to be rendered once before all the data-bound rows are displayed
- **ItemTemplate** – This defines the layout of the elements to be rendered once for each row in the Repeater control. This template is used to bind one or more ASP.NET server controls to a data source. This template is mandatory
- **AlternatingItemTemplate** – This is similar to the ItemTemplate, but it is rendered for every alternate row in the Repeater control
- **SeparatorTemplate** – This defines the layout of the elements to be rendered between each row, such as line breaks, lines etc
- **FooterTemplate** – This defines the layout of the elements to be rendered once after all the data-bound rows are displayed

The Repeater control is bound to the data source through its **DataSource** property. It can bind to any class that supports the ICollection or IEnumerable interface.



#### Practice 12.3

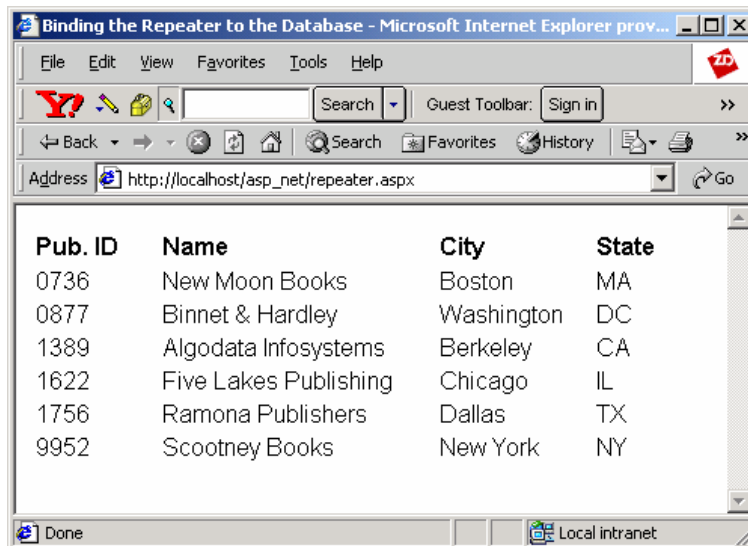
The following example uses the **Repeater** control to retrieve the rows whose country is **USA**, from the **Publishers** table.

```

<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SQL" %>
<html><head>
<title>Binding the Repeater to the Database</title>
<script language="C#" runat="server">
void Page_Load ( Object src, EventArgs e)
{
    DataSet DS;
    SqlConnection conn;
    SQLDataSetCommand cmd;
    string connStr, query;
    connStr="server=localhost;uid=sa;database=pubs";
    conn = new SqlConnection(connStr);
    query = "select * from Publishers where country='USA'";
    cmd = new SQLDataSetCommand(query,conn);
    DS = new DataSet();
    cmd.FillDataSet(DS, "publishers");
    rep.DataSource=DS.Tables["Publishers"].DefaultView;
    rep.DataBind();
}
</script>
</head>
<body bgcolor="#FFFFFF">
    <ASP:Repeater id="rep" runat="server">
        <template name="headertemplate">
            <table >
                <tr align=left >
                    <th width=80> Pub. ID </th>
                    <th width=180> Name </th>
                    <th width=100> City </th>
                    <th> State </th>
                </tr>
            </table>
        </template>
        <template name="itemtemplate">
            <tr>
                <td><%# DataBinder.Eval(Container.DataItem, "pub_id")%> </td>
                <td> <%# DataBinder.Eval(Container.DataItem, "pub_name")%></td>
                <td> <%# DataBinder.Eval(Container.DataItem, "city")%> </td>
                <td> <%#DataBinder.Eval(Container.DataItem, "state")%></td>
            </tr>
        </template>
        <template name="footertemplate">
            </table>
        </template>
    </ASP:Repeater>
</body>
</html>

```





The above example binds the Repeater control to the database in the **Page\_Load** event. It creates a connection object named **conn** using the connection string **connStr**. A command object **cmd** is then created using this connection and the query string **query**. The command is executed and the results are retrieved in the DataSet **DS**. Then the DataSet is linked to the repeater through the properties **DataSource**. The **itemtemplate** of the repeater defines how the data of each row has to be displayed.

## 12.4 Short Summary

- The Web Forms page enables the user to bind any control's property to the information in any kind of data store
- The Web Forms pages access data in the form of classes that expose data as properties and define methods for updating the underlying data source
- The property of a control can be bound to a datasource using the databinding expression. The databinding expression is evaluated only when the DataBind method is called
- The ASP.NET data binding syntax supports binding to public variables, properties of the page and properties of other controls on the page
- List server controls like ListBox, DropDownList etc., use a collection as a datasource
- The controls like ListBox, DropDownList, CheckBoxList and RadioButtonList can be used to display a single field from a table
- The Repeater control is used to display data items in a repeating list

## 12.5 Brain Storm

1. What are the ways in which Databinding can be achieved?
2. What is the importance of the DataBind method?

3. List the steps involved in binding a control to the database.
4. List the templates supported by the Repeater control.
5. Differentiate between AlternatingItemTemplate and ItemTemplate.

❧

## Lecture 13

---

# DataGrid and DataList Server Controls

---

### Objectives

In this lecture you will learn the following

- + Knowing about DataGrid Server Control
- + What is meant by paging in DataGrid

---

## Coverage Plan

---

<b>Lecture 13</b>
13.1 Snap Shot
13.2 DataGrid Server Control
13.3 DataList Server Control
13.4 Short Summary
13.5 Brain Storm

## 13.1 Snap Shot

This session is planned to throw light on DataGrid Server control and various types of columns that are allowed in it. Further, the focus is on DataList control and selecting items in it. It also aimed at teaching methods to access database using DataGrid and DataList Server controls.

DataGrid Server control displays information in tabular form with columns. It Provides mechanisms that allow editing and sorting. DataList Server control is similar to Repeater control, but with more formatting and layout options, including the ability to display information in a table. The DataList control also allows to specify editing behavior.

## 13.2 DataGrid Server Control

Radiant™

ASP.NET

☐ Allows to add specific functionality, including selecting, editing, sorting, and paging data

☐ Must be bound to a data source through its DataSource property

☐ The grid displays one row, one item for every row in the data source

☐ 0

By default, the DataGrid control generates a bound column for each field in the data

The DataGrid ASP.NET sever control is a multi-column, data-bound grid that displays tabular data. The control allows to define various types of columns, both to lay out the contents of the grid and to add specific functionality, including selecting, editing, sorting, and paging the data.

The DataGrid ASP.NET server control must be bound to a data source via its DataSource property or it will not be accomplished on the page.

When data binding, a data source is specified for the DataGrid control as a whole. The grid displays one row, one item for every row in the data source. By default, the DataGrid control generates a bound column for each field in the data source. When the page runs, the controls DataBind method is called to load the grid with data. The method can be called each time the page makes a round trip to the server (for example, in an event-handling method) to refresh the grid.

### A simple datagrid



#### Practice 13.1

The following program shows a simple DataGrid with some values.

```
<%@ Import Namespace="System.Data" %>
<html>
<title> Simple DataGrid </title>
<script language="C#" runat="server">
```

```

ICollection CreateDataSource() {
    DataTable dtbl = new DataTable();
    DataRow drow;

    dtbl.Columns.Add(new DataColumn("Serial No", typeof(Int32)));
    dtbl.Columns.Add(new DataColumn("Product No", typeof(string)));

    for (int i = 1; i < 6; i++) {
        drow = dtbl.NewRow();

        drow[0] = i;
        drow[1] = "prd 00" + Int32.ToString(i);
        dtbl.Rows.Add(drow);
    }

    DataView dview = new DataView(dtbl);
    return dview;
}

void Page_Load(Object sender, EventArgs e) {
    MyDataGrid.DataSource = CreateDataSource();
    MyDataGrid.DataBind();
}
</script>

<body>

<h3><font face="Verdana">A Simple DataGrid </font></h3>

<form runat=server>

    <ASP:DataGrid id="MyDataGrid" runat="server"
        BorderWidthblh="1"
        GridLines="Both"
        CellPadding="3"
        CellSpacing="0"
        Font-Name="Courier"
        Font-Size="14pt"
        HeaderStyle-BackColor="#aaaadd"
    />

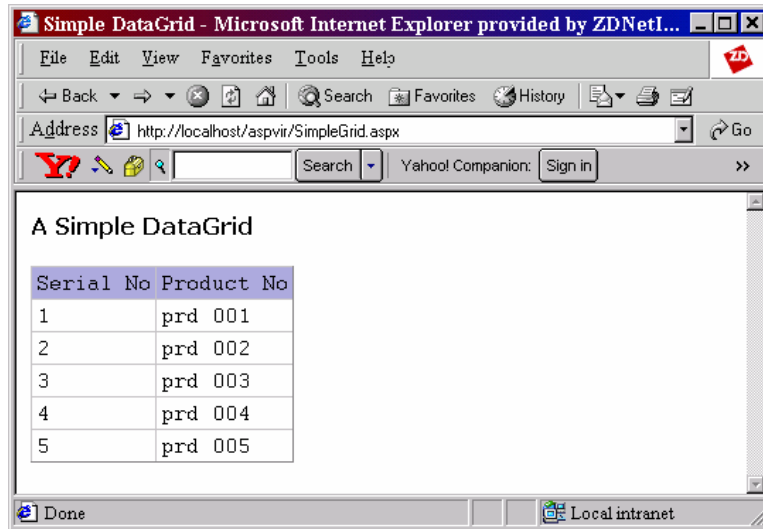
</form>

</body>
</html>

```







The above example shows a simple DataGrid control which shows Serial No and Product No. These two Columns has been created using DataColumn object in DataTable. Similarly rows has been inserted using DataRow object in DataTable.

### DataGrid Columns

The **DataGrid** control allows to specify the columns it displays in a variety of ways. By default, the columns are generated automatically based on the fields in the data source. However, in order to manipulate the content and layout of columns more precisely, the following types of columns can be defined:

- **Bound columns**

This columns allow to specify which database fields to be displayed and in what order it must be displayed and allow to specify the format and style of the display.

- **Hyperlink columns**

This columns display information as hyperlinks. A typical use is to display data (such as Area number or Area name) as a hyperlink that users can click to navigate to a separate page that provides details about that information.

- **Button columns**

These columns allow to display buttons next to each item in the grid and define custom functionality for the buttons. For example, a button might be created and labeled as "Add to Shopping Cart" that runs the custom logic when users click it.

- **Edit command columns**

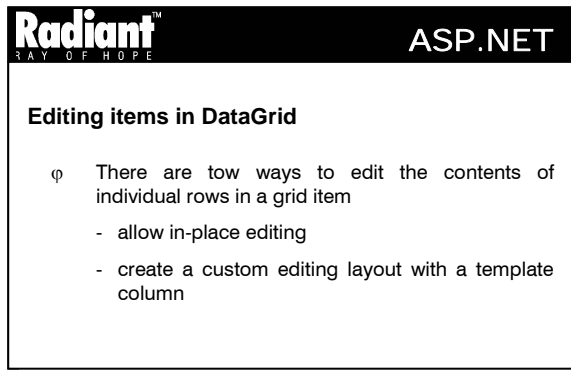
This column displays Edit, Update, and Cancel links in response to changes in the DataGrid's EditItemIndex property.

- **Template columns**

These columns allow to create combinations of HTML text and server controls to design a custom layout for a column. The controls within a template column can be data bound. Template columns provide the ultimate flexibility in defining the layout and functionality of the grid contents.

**Note:** To execute the following program, an MSAccess database named **sample.mdb** with a table **Employee** has to be created with the fields **EmployeeID**, **FirstName**, **Salary**.

### Editing Items In DataGrid



There are two ways to edit the contents of individual rows in a grid item. They are:

- Allow in-place editing
- Create a custom editing layout with a template column

#### Allow in-place editing

This is the easiest method of allowing editing. A special column can be included in the grid with buttons labeled "Edit". When users click the edit button, the control automatically redisplay the current row with editable fields for all columns (for example, text boxes for characters and numbers and checkboxes for Boolean data). The column with the edit button is redisplayed with update and cancel buttons.

#### Create a custom editing layout with a template column

It is possible to create columns that display the data in editable controls. This approach allows to manipulate very precisely the columns that can be edited and allows to choose how the user can edit the data. When a template column is used, the items in the grid typically do not switch between edit and display modes. Instead, the items are always in edit mode. In either approach, a custom logic must be added to the page to update the data source with the user's edits.

#### Sorting Columns in DataGrid

The **DataGrid** ASP.NET server control provides a way to add sorting to the grid using the following methods:

- **Default sorting**

All columns in the grid are sortable. The header for each column contains a **LinkButton** control (hyperlink) that users click to sort by that column.

- **Custom sorting**

It is possible to define which column to support sorting and what type of button in the column heading user click to sort.

In either case, the grid does not sort the rows. Instead, it notifies the sort request by raising an event. Then sorting is performed in the code, usually by passing rebinding to the data source with new sort parameters.

Data in a grid is commonly sorted by clicking the header of the column. This is enabled by setting **AllowSorting=true**. When enabled, the grid will render LinkButtons in the header for each column. When the button is clicked, the grid's OnSortCommand event is thrown. Since DataGrid always displays the data in the same order it occurs in the datasource, the typical logic sorts the datasource, then rebinds the data to the grid.



---

### Practice 13.2

The following example illustrates the sorting of columns in a DataGrid .

```
<% @ Import Namespace="System.Data" %>
<% @ Import Namespace="System.Data.ADO" %>
<html>
<head>
<script language="c#" runat="server" >
ADOConnection myconnection;
String conn;
void Page_Load(object source,EventArgs e)
{
conn="provider=microsoft.jet.oledb.4.0;data source=C:/sample.mdb";
myconnection=new ADOConnection(conn);

if (!IsPostBack)
    BindGrid("EmployeeID");
}

void dbsort(object src,DataGridSortCommandEventArgs e)
{
    BindGrid(e.SortField);
}
void BindGrid(String sortfield)
{
    //For datasetcommand connection will be established implicitly
    ADODataSetCommand mycommand=new ADODataSetCommand("select
EmployeeID,FirstName,Salary from Employees",conn);

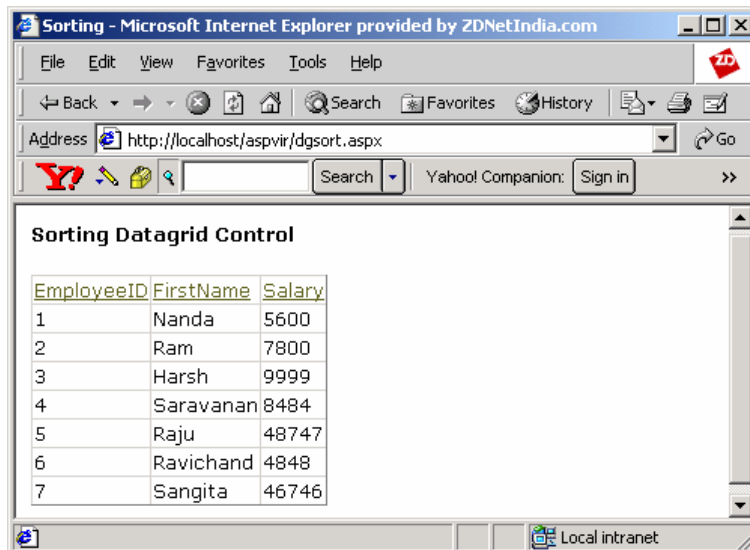
DataSet ds=new DataSet();
mycommand.FillDataSet(ds,"employees");

DataView dv= ds.Tables["employees"].DefaultView;
dv.Sort = sortfield;
```

```

    DataGrid1.DataSource = dv;
    DataGrid1.DataBind();
}
</script>
</head>
<form runat="server">
    <h4> Sorting Datagrid Control </h4>
    <asp:Datagrid id="DataGrid1" onsortcommand="dbsort" allowsorting="true"
runat="server" autogeneratecolumns="true"/>
</form>
</body>
</html>

```



The above example shows how to do sorting in datagrid. When we click any one of the header column the rows will be sorted in ascending order. The sortfield has been used to set the field to be sorted. Set the DataView's sort property to the sortfield so that data view will sort it out.

### Paging in DataGrid

**Radiant™**  
RAY OF HOPE

**ASP.NET**

- ☐ Provides the means to display a group of records from the datasource, then navigate to the "page" of next 10 records, and so on through the data.
- ☐ Enable in DataGrid by setting AllowPaging=true
- ☐ Grid displays page navigation buttons either "next/previous" buttons or as numeric buttons

The **DataGrid** ASP.NET server control provides a way to add paging to the grid using the following methods:

- **Using built-in paging controls**

The grid displays navigation buttons (either Next and Previous buttons or numeric page numbers). When the user clicks these, the grid updates the current page number and raises an event so that the data can be refreshed.

- **Providing custom navigation controls**


It is possible to provide own paging controls and manually set which page is to be displayed. This option allow to move any number of pages at a time, jump to a specific page, and so on.

### Automatic versus Manual Paging

The page displayed by the grid is determined by its **CurrentPageIndex** property. The built-in controls set this property automatically; if custom navigation controls are provided it has to be set. After the **CurrentPageIndex** is set, the grid should be re-bound to the data source. The grid will recreate the entire data set and automatically move to the appropriate place in the data set. It then displays enough rows to make up one page of the grid.

In case of working with a large data set, recreating the entire data set each time users navigate to a new page can require too many overheads. In that case, we might prefer to get data in page-size “chunks”, that is, retrieving just a page worth of records at a time. To allow that, turn off the automatic paging feature of the grid so that it doesn’t assume that it is working with the entire data set. Then correct number of rows should be retrieved manually to fill the grid.

### Accessing a Database



**ASP.NET**

- φ SQL Connection class gets or sets the name of the current database or the database to be used once a connection is open
- φ Accessing a Database using DataGrid is achieved with the following steps:
  - Establish a connection
  - Execute a query and retrieve the results in a DataView, DataReader or DataSet
  - Bind the DataView, DataReader or DataSet to the DataGrid through the DataSource property

SqlConnection class gets or sets the name of the current database or the database to be used once a connection is open.

Accessing a Database using DataGrid is achieved with the following steps:

- Establish a connection
- Execute a query and retrieve the results in a DataView, DataReader or DataSet
- Bind the DataView, DataReader or DataSet to the DataGrid through the DataSource property



The following example shows how a DataGrid is used to access the data in the SQL Server database.

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

<html>
<script language="C#" runat="server">
void Page_Load ( Object src, EventArgs e)
{
    DataSet DS;
    SqlConnection MyConnection;
    SQLDataSetCommand MyCommand;
    string strconn, strSQL;

    strconn="server=saravanan;uid=sa;database=pubs";
    MyConnection = new SqlConnection(strconn);
    strSQL="select * from Publishers";
    MyCommand = new SQLDataSetCommand(strSQL,MyConnection);
    DS = new DataSet();
    MyCommand.FillDataSet(DS, "publishers");
    MyDataGrid.DataSource = DS.Tables["Publishers"].DefaultView;
    MyDataGrid.DataBind();
}
</script>

<body>
<form runat=server>
<ASP:DataGrid id="MyDataGrid" runat="server"
    BorderColor="black"
    BorderWidth="1"
    GridLines="Both"
    CellPadding="3"
    CellSpacing="0"
    Font-Name="Verdana"
    Font-Size="8pt"
    HeaderStyle-BackColor="#aaaadd" />

</form>
</body>
</html>
```



The screenshot shows a Microsoft Internet Explorer browser window titled "Publishers Table - Microsoft Internet Explorer provided by ZDNetIndi...". The address bar contains "http://localhost/aspvir/dbase.aspx". The main content area displays a table with the following data:

pub_id	pub_name	city	state	country
0736	New Moon Books	Boston	MA	USA
0877	Binnet & Hardley	Washington	DC	USA
1389	Algodata Infosystems	Berkeley	CA	USA
1622	Five Lakes Publishing	Chicago	IL	USA
1756	Ramona Publishers	Dallas	TX	USA
9901	GGG&G	München		Germany
9952	Scootney Books	New York	NY	USA
9999	Lucerne Publishing	Paris		France

In the above example, data from the Publishers table are displayed in the DataGrid. When the "Show Values" button is clicked, the records are displayed as shown.

### 13.3 DataList Server Control

**Radiant**  
3 A Y O F H O P E

**ASP.NET**

- ⊕ Displays database information in a format that can be controlled very precisely using templates and styles
- ⊕ Useful for displaying rows of database information displays rows of data as *items* in the list
- ⊕ It is possible to use templates to define the layout of the items by including HTML text and controls.

The DataList control displays database information in a format that can be controlled very precisely using templates and styles. The DataList control is useful for displaying rows of database information. The DataList control displays rows of data as *items* in the list. It is possible to use templates to define the layout of the items by including HTML text and controls.

#### Templates for Control Layout

Templates define the HTML elements and controls that should be displayed for an item, as well as the layout of these elements. Style formatting such as font, color and border attributes — is set using the **styles** property. Each template has its own style property. For example, styles for the EditItemTemplate are set via the **EditItemStyle** property.

The following templates are supported by DataList control.

- **ItemTemplate**

This template defines the content and layout of items within the list.

- **AlternatingItemTemplate**  
If this template is defined, this determines the content and layout of alternating items. If it is not defined, ItemTemplate is used.
- **SelectedItemTemplate**  
If this template is defined, this determines the content and layout of the selected item. If it is not defined, ItemTemplate is used.
- **EditItemTemplate**  
This defines the layout of an item when it is in edit mode. This template typically contains editing controls, such as **TextBox** controls.
- **HeaderTemplate and FooterTemplate**  
This text defines the text and controls to render at the beginning and end of the list.
- **SeparatorTemplate**  
This defines the elements to render between each item. A typical example might be a line (using an **<HR>** element).

### A Simple DataList

DataList supports directional rendering through the **RepeatDirection** property. This means that it can render its items horizontally or vertically. Since page width is the dimension that the developer must control in Web UI, DataList permits the developer to control the number of “columns” that are rendered (**RepeatColumns**), regardless of whether the items are rendered horizontally or vertically.



#### Practice 13.4

This following example displays a simple DataList control.

```
<%@ Import Namespace="System.Data" %>
<html>
<script language = "C#" runat="server">
ICollection CreateDataSource() {
    DataTable dt = new DataTable();
    DataRow dr;
    dt.Columns.Add(new DataColumn("StringValue", typeof(string)));
    for (int i = 0; i <5; i++) {
        dr = dt.NewRow();
        dr[0] = "Item " + Int32.ToString(i);
        dt.Rows.Add(dr);
    }
    DataView dv = new DataView(dt);
    return dv;
}
```



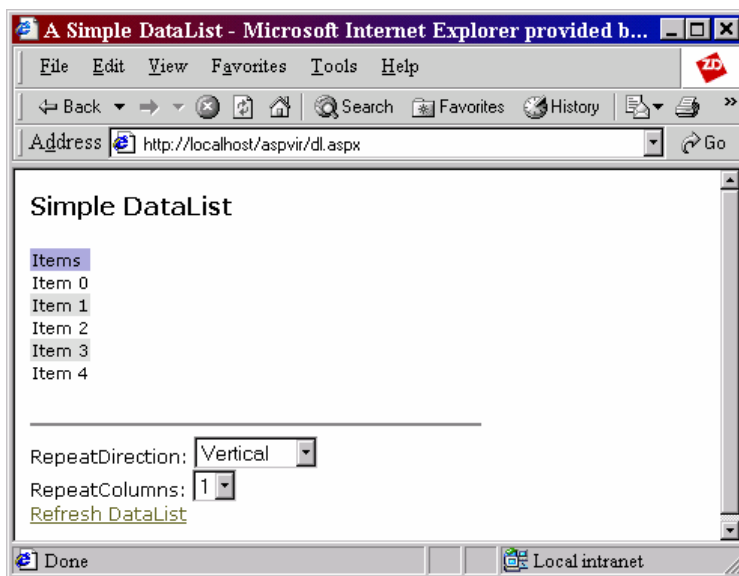
```
void Page_Load(Object Sender, EventArgs E) {
    if (!IsPostBack) {
        DataList1.DataSource = CreateDataSource();
        DataList1.DataBind();
    }
}

void Button1_Click(Object Sender, EventArgs E) {
    if (DropDown1.SelectedIndex == 0)
        DataList1.RepeatDirection = RepeatDirection.Horizontal;
    else
        DataList1.RepeatDirection = RepeatDirection.Vertical;
}
</script>
<body>
    <h3><font face="Verdana">Simple DataList Sample</font></h3>
    <form runat=server>
        <font face="Verdana" size="-1">
            <asp:DataList id="DataList1" runat="server"
                BorderColor="black"
                Font-Name="Verdana"
                Font-Size="8pt"
                HeaderStyle-BackColor="#aaaadd"
                AlternatingItemStyle-BackColor="Gainsboro"
            >
                <template name="HeaderTemplate">
                    Items
                </template>
                <template name="ItemTemplate">
                    <%# DataBinder.Eval(Container.DataItem, "StringValue") %>
                </template>
            </asp:DataList>
        </font>
        <p>
            <hr noshade align="left" width="300px">
                RepeatDirection:
                <asp:DropDownList id=DropDown1 runat="server">
                    <asp:ListItem>Horizontal</asp:ListItem>
                    <asp:ListItem>Vertical</asp:ListItem>
                </asp:DropDownList><br>
                RepeatColumns:
                <asp:DropDownList id=DropDown3 runat="server">
                    <asp:ListItem>1</asp:ListItem>
                    <asp:ListItem>2</asp:ListItem>
                    <asp:ListItem>3</asp:ListItem>
                    <asp:ListItem>4</asp:ListItem>
                </asp:DropDownList>
            </p>
        </form>
    </body>
```

```

        </asp:DropDownList><br>
        <asp:LinkButton id=Button1 Text="Refresh DataList"
        OnClick="Button1_Click" runat="server" />
    </font>
</form>
</body>
</html>

```



Above example shows a simple DataList control. As it can be seen from the above program a table is created using DataTable class and the rows are added using NewRow method. RepeatDirection property is used to render the items horizontally or vertically. With the help of RepeatColumns property, the number of columns to be repeated can be shown.

### Selecting Items in DataList

It is possible to customize the content and appearance of the selected item via the **SelectedItemTemplate** property. The SelectedItemTemplate is controlled by the **SelectedIndex** property. By default the value of SelectedIndex is -1, meaning none of the items in the list is selected. When SelectedIndex is set to a particular item, that item is displayed using the SelectedItemTemplate.

### Accessing a Database

SqlConnection class represents an open connection to a SQL Server data source. SQLDataSetCommand represents a set of data commands and a database connection which are used to fill the DataSet and

update the data source. DataBinder Class provides design-time support for Rapid Application Designers (RAD) designers to generate and parse DataBinding expression syntax.

Accessing a Database using DataList is achieved with the following steps:

- Establish a connection
- Execute a query and retrieve the results in a DataView, DataReader or DataSet
- Bind the DataView, DataReader or DataSet to the DataList through the DataSource property



### Practice 13.5

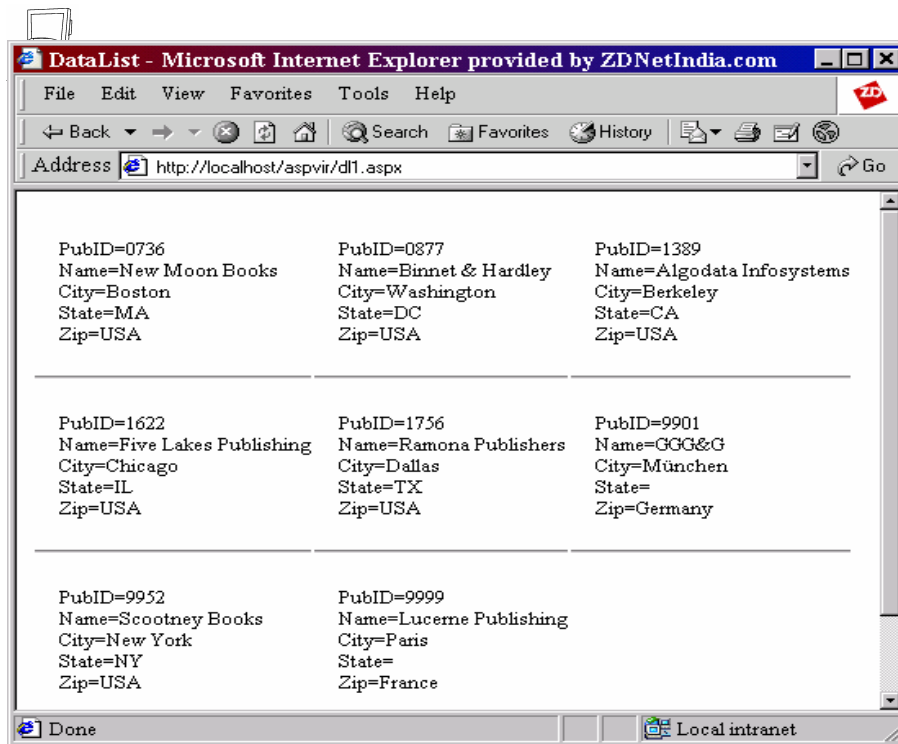
The following example shows how a DataGrid is used to access the data in the database.

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SQL" %>
<script language="C#" runat="server">
void Page_Load ( Object src, EventArgs e)
{
    DataSet DS;
    SqlConnection Mycon;
    SQLDataSetCommand Mycomm;
        string strconn, strSQL;
    strconn="server=myserver;uid=sa;database=pubs";
        Mycon = new SqlConnection(strconn);
    strSQL="select * from Publishers";
    Mycomm = new SQLDataSetCommand(strSQL,Mycon);
    DS = new DataSet();
    Mycomm.FillDataSet(DS, "publishers");
    datlis.DataSource=DS.Tables["Publishers"].DefaultView;
    datlis.DataBind();
}
</script>
<html><head>
    <title>DataList</title>
</head>
<body bgcolor="#FFFFFF">
    <ASP:DataList id="datlis" repeatcolumns="3" repeatdirection="horizontal"
        repeatlayout="table" runat="server">
    <template name="headertemplate">
        </template>
    <template name="itemtemplate">
        PubID=<#%# DataBinder.Eval(Container.DataItem, "pub_id")%><br>
        Name=<#%# DataBinder.Eval(Container.DataItem, "pub_name")%><br>
        City=<#%# DataBinder.Eval(Container.DataItem, "city")%><br>
        State=<#%# DataBinder.Eval(Container.DataItem, "state")%><br>
```

```

Zip=<%# DataBinder.Eval(Container.DataItem, "country")%><br>
<hr>
</template>
<template name="footertemplate">
</template>
</ASP:DataList>
</body>
</html>

```



As seen from the above example, a connection to the database is created using a Connection object. SQL query is passed to the `SQLDataSetCommand` object `mycom`. The `FillDataSet` method executes the command and populates the `DataSet DS` with the result. The `DataSource` property of the `DataList` is used to bind the `DataList` to the `DataSet`. Then the `DataBind` method is used to list out the values in the `DataList`.

### 13.4 Short Summary

- The `DataGrid` ASP.NET server control is a multi-column, data-bound grid that displays tabular data
- The `DataGrid` ASP.NET server control must be bound to a data source via its `DataSource` property
- The `DataGrid` control allows to specify the columns it displays in a variety of ways
- The types of columns that can be defined in `DataGrid` are Bound columns, Hyperlink columns, Button columns, Edit command columns and Template columns

- The two ways to edit the contents of individual rows in a grid item are Allow in-place editing and Create a custom editing layout with a template column
- Data in a grid is commonly sorted by clicking the header of the column and is enabled by setting AllowSorting=true
- The DataGrid provides the means to display a group of records from the datasource
- SqlConnection class gets or sets the name of the current database or the database to be used once a connection is open
- The DataList control displays database information in a format that can be controlled very precisely using templates and styles
- DataList supports directional rendering through the RepeatDirection property
- It is possible to customize the content and appearance of the selected item through the SelectedItemTemplate property
- SqlConnection class represents an open connection to a SQL Server data source

### 13.5 Brain Storm

1. What is a DataGrid Control?
2. Explain various possible columns in a DataGrid.
3. What are the two ways to edit the contents of individual rows in a grid item?
4. What is a DataList control?
5. What are the Templates supported by DataList control?



## Lecture 14

---

# Tracing, Error-Handling and Debugging

---

### Objectives

In this lecture you will learn the following

- + Compare the tracing in ASP and ASP.NET
- + Knowing about the Error-Handling
- + Knowing about Debugging

---

## Coverage Plan

---

### **Lecture 14**

- 14.1 Snap Shot
- 14.2 Tracing
- 14.3 Error-Handling
- 14.4 Debugging
- 14.5 Short Summary
- 14.6 Brain Storm

## 14.1 Snap Shot

This session gives an overview of the processes of tracing, error-handling and debugging the ASP.NET applications.

ASP.NET includes an easy-to-use functionality that helps to debug Web applications. Tracing functionality allows debugging print statements to be inserted into the code to output variables or structures, assert whether a condition is met, or trace through the execution path of the application.

Debugging is used to find and correct the logical errors that keep the application or stored procedures from running correctly. Tracking down problems in the code can be confusing without the appropriate tool support. Fortunately, the compiled nature of ASP.NET means that debugging Web applications is no different from debugging any other managed application and the .NET Framework SDK includes a lightweight visual debugger that is perfectly suited for this task.

## 14.2 Tracing

### Tracing with ASP

In an ASP application, the **Response.Write()** statement is used to dissect a troublesome section of code.

Here is some pseudo code to illustrate this point:

```
<%
On Error Resume Next
Response.Write("About to do data access...")
Set objDB = Server.CreateObject("Custom.DataAccess")
objDB.Open("PersonalizationData")
user = objDB.Get("username")
Response.Write("Done calling data access. Value of username is:" + user)
%>
```

It is evident from the **Response.Write()** calls in the code given above that an error has occurred in the behavior of the code. The error is not necessarily an application or system error, such as a failure to load the object, but rather a coding error in which the code executes as expected but does not return the expected value. Of course, debugger can also be used, but sometimes it is just faster to get a output to find out what the code is doing or to simply know how a particular section of code is being executed. Although **Response.Write()** statements do make it easy to debug the application rather quickly, they introduce unnecessary code into an application that can result in real bugs that break deployed applications. The same functionality of **Response.Write()** statements is supported within ASP.NET.

### Tracing with ASP.NET

**Radiant™**  
RAY OF HOPE

**ASP.NET**

**Trace**

- ☐ Used to track particular types of actions in a deployed application
- ☐ Can monitor application's efficiency
- ☐ Allow to simulate `Response. Write()` statements
- ☐ Trace class provides methods and properties to trace execution of codes



Trace feature is used to track particular types of actions in a deployed application as they occur (for example, database connections), and can thus monitor the application's efficiency. As another example, a developer might write a program that must connect to the Internet in order to obtain updates to a file. The developer could include tracing code in the program to record what steps are being used to accomplish the task of connecting to the Internet. After the program is deployed in production, if a problem occurs during the connection process, the end user can turn on a Trace switch to enable the tracing options that the developer coded.

The new tracing features of ASP.NET allow to simulate **Response.Write()** statements. The users need not worry about removing the statements before deploying the applications. Instead of using `Response.Write()`, `Trace.Write()` is used. The Trace object is an intrinsic page object, similar to Request, Response, Server, etc. It is accessible directly with the page code.

### Trace class

Trace is exposed as a public property within ASP.NET pages. When the Trace property is used, an instance of the TraceContext class is defined in the System.Web namespace. Trace class provides a set of methods and properties that help to trace the execution of code. Instrumentation allows to monitor the health of the application running in real-life settings. Tracing helps to isolate problems and fix them without disturbing a running system.

There are two different ways to enable tracing:

- In C# or Managed Extensions for C++, the `/d:TRACE` flag can be added to the compiler command line. In Visual Basic, the `/d:TRACE=True` flag can be added to the compiler command line
- The `#define TRACE` can be added to the top of the file. This syntax is compiler specific. If the user is using a compiler other than the ones specified above, the compiler's documentation has to be referred to enable conditional compilation

### Trace Methods

The methods of the Trace class are listed below.

- **Assert()** - Checks for a condition, and displays a message if the condition is false
- **Close()** - Flushes the output buffer, and then closes the Listeners so that they no longer receive debugging output
- **Fail()** - Emits an error message
- **GetType()** - Gets the Type of the Object
- **ToString()** - Returns a String that represents the current Object
- **GetHashCode()** - Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table
- **Equals()** - Determines whether the specified Object is the same instance as the current Object
- **Warn()** - Writes trace information, along with optional exception data, to the trace log. All warnings appear as red text. It has two forms. The first form writes trace information to the trace log including any user defined categories. Its syntax is:

```
public void Warn(string, string);
```

The second form writes trace information to the trace log including any user defined categories, trace messages and error information. Its syntax is:

```
public void Warn(string, string, Exception);
```

- **Write()** - Writes trace information to the trace log. It has two forms. The first form writes trace information to the trace log, including any user defined categories and trace messages. Its syntax is:

```
public void Write(string, string);
```

The second form writes trace information to the trace log including any user defined categories, trace messages and error information.

```
public void Write(string, string,
    Exception);
```

The category parameter is used to identify the category name to write the message value into. This will become more apparent when the Write() method is used. The third parameter that both Warn() and Write() support in their overloaded methods is errorInfo. This allows to pass exception details into the trace system.

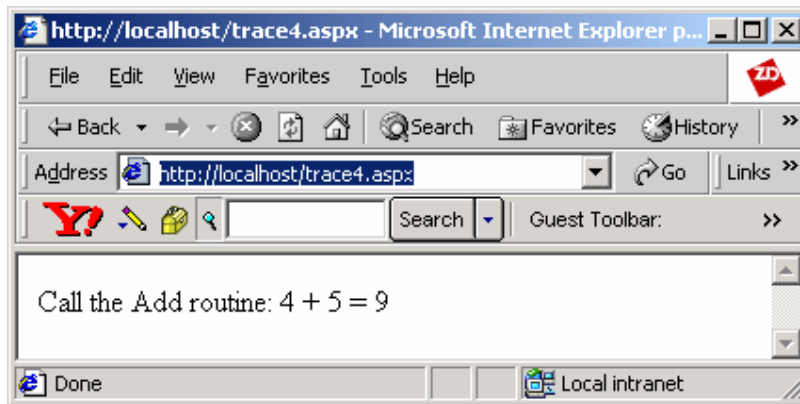


### Practice 14.1

An example to show the application of the Write method.

```
<html>
<script Language=C# runat=server>
public int Add(int a , int b)
{
    Trace.Write("Inside Add() a: ", a.ToString());
    Trace.Write("Inside Add() b: ", b.ToString());
    return a + b;
}
</script>
<body>
    Call the Add routine: 4 + 5 = <%=Add(4,5)%>
</body>
</html>
```





Here the Add() method is called. However, unlike the Response.Write() statements, the Trace.Write() statements do not appear within the resulting output. To view the results of the Trace.Write() statements, we need to enable tracing for this page or for the application.

To enable tracing, either a page directive or a configuration option has to be used.

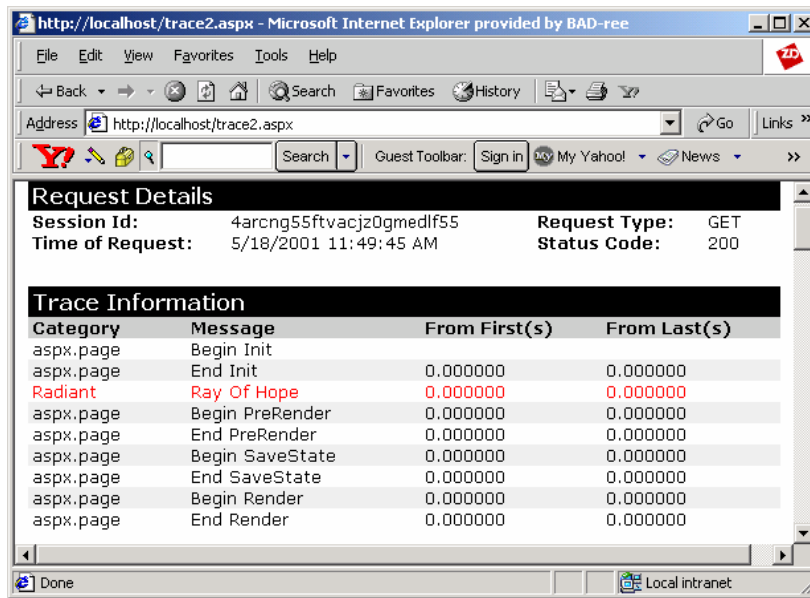


### Practice 14.2

The following application shows how to enable tracing.

```
<%@ Page Language="C#" Trace=True %>
<html>
<script runat="server">
void Page_Load(Object o,EventArgs e)
{
    Trace.Warn("Radiant","Ray Of Hope");
}
</script>
</html>
```





Here the warn method writes trace information, along with optional exception data, to the trace log. All warnings appear as red text. If Write method is used instead of Warn method, all warnings appear in black.

### Trace Properties

The **TraceContext** class supports two public properties: **IsEnabled** and **TraceMode**.

#### IsEnabled

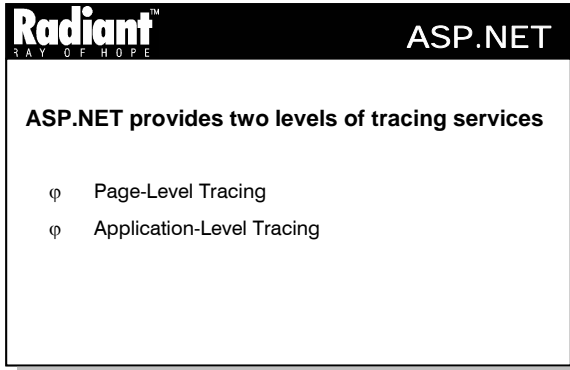
It indicates whether tracing is enabled on the page or for the application. This property can be set to true for a page by including a trace="true" attribute in the @Page directive. For an entire application this property can be set to true in the application's **config.web** file.

**Note :** If this property is set to true for an entire application, then for any page in that application for which the tracing information need not be displayed, this property must be explicitly set to false for that page.

#### TraceMode

It sets or returns the sorting mode that tracing is using. It indicates the order in which trace messages should be output to a requesting browser. Trace messages can be sorted in the order in which they are processed, or alphabetically by user-defined category.

## Tracing Levels



### Page-level Tracing

ASP.NET aids the debugging and testing process by providing a trace capability that does two things when enabled:

- Automatically appends a table of performance data to the end of the page
- Allows to insert custom diagnostic messages to be inserted throughout the code

Tracing messages are appended to the HTML output stream that is sent to the requesting browser. This information helps to clarify errors or undesired results as the framework processes a page request. The trace capability is encapsulated in the TraceContext class, which is exposed as the Page.Trace property.

Essentially, there are two steps involved in viewing trace statements in a page, irrespective of whether they are generated from a code-behind file or in an HTML editor. The first is to write the statements to be included to the trace log, and the second is to enable the page to display those messages and other trace information.

At the page-level, developers can use the TraceContext intrinsic to write custom debugging statements that appear at the end of the client output delivered to the requesting browser. ASP.NET also inserts some helpful statements regarding the start/end of lifecycle methods, like Load() and Dispose() as well as the inputs and outputs to a page, such as Form and querystring variables or headers, and important statistics about the page's execution (control hierarchy, session and application state). Because tracing can be explicitly enabled or disabled for a page, these statements may be left in the production code for a page with no impact to the page's performance. Each statement is associated with a user-defined category for organizational purposes, and timing information is automatically collected by the ASP.NET runtime. The resulting output may be ordered by either time or category.

Page directives are special instructions that ASP.NET uses when processing a request for an ASP.NET resource. Page directives can be used to override or apply configuration settings for an ASP.NET page.

A directive must be the first element on a page. It is declared using the following syntax:

```
<@ Directive Attribute="[Value]" %>
```

Using the above syntax, in ASP.NET the page can be enabled as:

```
<@ Page Trace="true" %>
```

### Example

The above page directive is added to the sample code **Practice - 1**. The output changes as shown in the following figure:

Tracing is enabled by setting the Page directive to true. The tracing output is added to the bottom of the page. Trace output is *always* added at the end of the page.

### Application-Level Tracing

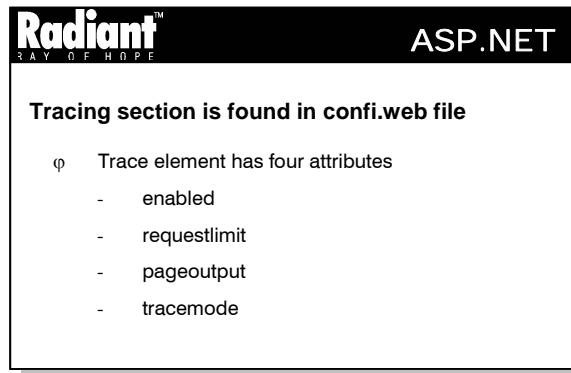
Application level tracing provides a view of several requests to an application's pages at once. Like page-level tracing, it also displays inputs and outputs to a page, such as Form and querystring variables or headers, as well as some important statistics about the page's execution (control hierarchy, session and application state). Application-level tracing is enabled via the ASP.NET configuration system, and accessed as a special mapped URL into that application ("trace.axd"). When application tracing is enabled, page-level tracing is automatically enabled for all pages in that application.

ASP.NET uses XML configuration files to set configuration settings for ASP.NET applications. Every ASP.NET installation installs a configuration file in the [systemdrive]\WinNt\Microsoft.NET\Framework\[version]\ directory named config.web.

The configuration file sets the default ASP.NET configuration, and is currently known as the root configuration file. Changes made within this file affect all the ASP.NET Web application. The configuration files can also be used in Web applications to add or remove settings acquired from the default **config.web**.

**Note :** If tracing is enabled at the application level, it is automatically enabled for each page in the application. However, to view trace information on a page the **PageOutput** attribute must be used in the **config.web** file.

### Trace section



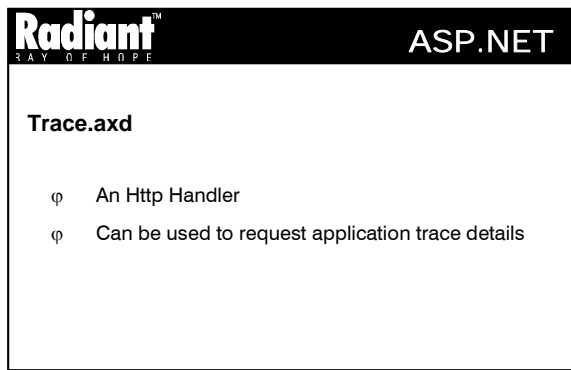
Tracing section is found in the root **config.web** file.

```
<configuration>
  <trace
    enabled="false"
    requestlimit="10"
    pageoutput="false"
    tracemode="SortByTime"
  />
</configuration>
```

There are four attributes for the trace element:

- **enabled = "[true/false]"** – The enabled option can be set to true or false. Tracing is either enabled at an application level, or it is disabled at the application level. If enabled is set to false, page tracing is still supported using the Trace directive discussed earlier
- **requestlimit = "[int]"** – Indicates the total number of trace requests to keep cached in memory on a per-application basis. Tracing exposes a special resource **Trace.axd**, that is used to view trace output when pageoutput is set to false
- **pageoutput = "[true/false]"** – When tracing is enabled through the configuration file, the administrator can either enable or disable tracing on each page. The pageoutput tracing enables tracing details for every page within an application. However, pageoutput tracing may be turned off while application-level tracing is still enabled (enabled = "true"). This keeps trace requests in memory, such that they are available through **trace.axd**, but not displayed within the output of a page
- **tracemode = "[SortByTime | SortByCategory]"** – The tracemode setting gives us control over how trace detail information is output. Data may be sorted by time or category, where category is differentiated between the settings made by the system and the Trace.Write() settings enabled by the developer

### Trace.axd



Trace.axd is an Http Handler. An Http Handler is a coding option that allows request/responses to be handled at the most basic level. Http Handlers are the equivalent of ISAPI extensions. However, unlike ISAPI, they can be authored using any .NET language. Trace.axd is an Http Handler that can be used to request application trace details. When requested, a trace log of the last n requests are given; n is determined by the value set by requestlimit="[int]" in the configuration file.

To use **trace.axd**, it can be requested from the same application directory from where the request for the sample application has been made, as shown in Figure 14.1:

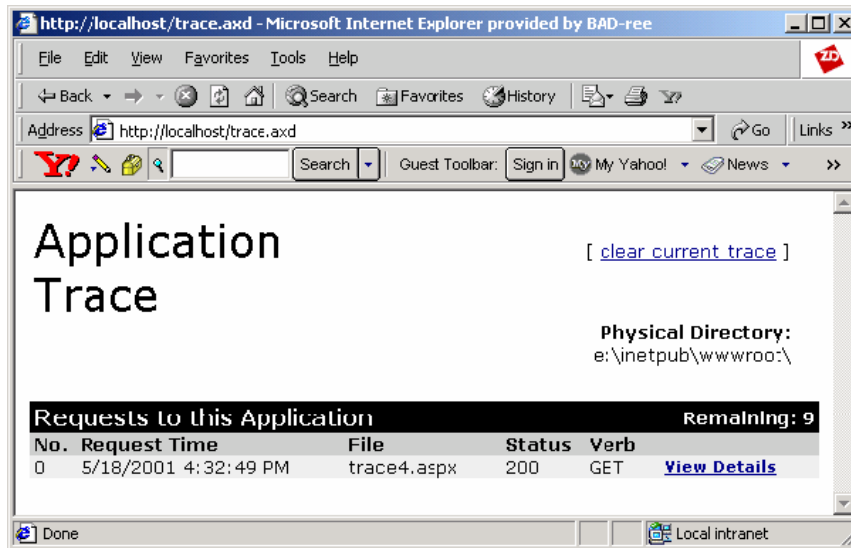


Figure 14.1

The screen shot, presents a list of traces that can be viewed (Figure 14.2). These trace views are identical to the details that trace would add to the page, without the output from the page included:

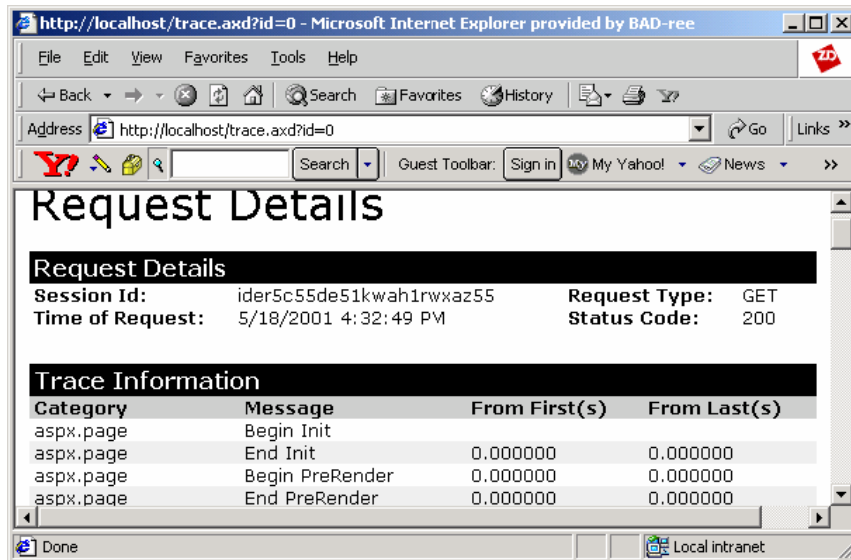


Figure 14.2

### Interpreting the Trace Output

The output provided by tracing view, either through Trace.axd or on a page, provides six sections of detail:

- **Request details**—This is basic information, such as the session id, time of the request, the Http request type, and the Http response status code
- **Trace information**—This section provides a table view of categories and messages. If Trace.Write() statements are used in the code, the two parameters that Trace.Write() accepts are used as the



Category and Message values, respectively. Additionally, it provides the time taken to trace from the first to the last byte

- **Control tree**—ASP.NET uses server controls to allow users to declaratively build applications. The Control Tree section provides information about controls within a page. The id, type, render size, and view state size of the control as provided. This information gives an idea of the controls that are being used, as well as the associated cost (render size and viewstate)
- **Cookies collection**—Any cookies that the client sends in the request headers are parsed, and their names, values, and sizes are displayed
- **Headers collection**—Http headers presented by the client to the server are provided in this section. It is a simple table that lists Name/Value pairs
- **Server variables**—The server variables section presents a Name/Value pair table of server variables

### Using Tracing

Tracing is a powerful new ASP.NET feature designed to make the development of Web applications easier. However, it is worthwhile to know the situations for which tracing should be used.

- **Deployed Applications**

Tracing adds additional overhead to requests and should not be enabled for deployed applications. `Trace.Write()` statements, however, can be retained because they are ignored when tracing is not enabled

The classes found in the `System.Diagnostics` namespace are used to capture data for deployed applications. These classes allow to write directly to the Windows event log. Tracing can be used when an application is built, but it should be disabled when the application is deployed

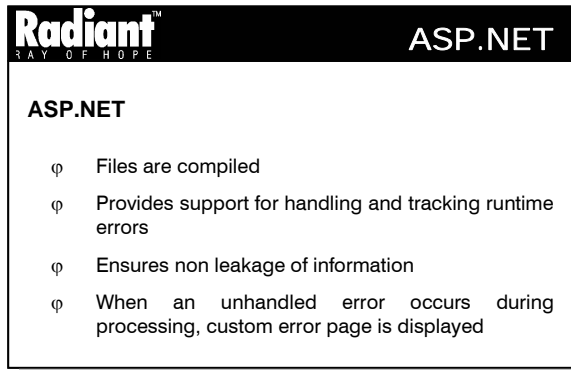
- **Disabling Trace.axd**

If ASP.NET is installed on a production Web server, it is recommended to disable **trace.axd** explicitly. This can be accomplished by adding the following in the `<httphandlers>` section of the `config.web` file

```
<configuration>
  <add verb="*"
        path="trace.axd"
        type="System.Web.Handlers.TraceHandler" />
  <remove verb="*" path="trace.axd" />
</configuration>
```

<p><b>Note :</b> The entire <code>&lt;add/&gt;</code> entry can be removed, but the <code>&lt;remove&gt;</code> option is better because it leaves the code in the configuration file and accomplishes the same goal by disabling <b>trace.axd</b>.</p>
---

## 14.3 Error-Handling



ASP.NET error messages are more descriptive and visually appealing when compared to ASP. ASP.NET files are compiled, and not interpreted like ASP files. Interpreted scripts process a file one line at a time and send data to the client as each line is being processed.

One of the new features of ASP.NET is its rich support for handling and tracking runtime errors. ASP.NET also provides administrators with additional information or notifications about problems that occur on production sites.

### Ensuring non leakage of information

ASP.NET will generate an html error page in the event that either a runtime or compile time error occurs on a server. Unlike ASP, however, ASP.NET ensures that the “secure” information is not leaked as a result of the occurrence of errors.

To view the details of the error, a user will either have to

- Go back to the local server box
- or
- Modify the configuration settings within the machine’s or application’s **config.web** file to enable remote access

```
<configuration>
  <customerrors mode="off" />
</configuration>
```

### Using Custom Error Pages

In the event, where an unhandled error occurs during the processing of a request, ASP.NET displays the user’s “custom error page”. This in turn displays the user’s own branding and the required error message.

Custom error support can be added into the ASP.NET application very easily. Initially, web page must be authored (it can be any type of file: .htm, .aspx, .asp etc). Then modify the configuration settings within the application’s config.web file to point to the ASP file.

**<customerrors> Section**

The <customerrors> section defines custom error messages for an ASP.NET application.

**Syntax**

```
<customerrors defaultredirect="[url]" mode="[on/off/remote]">
    <error statuscode="[statuscode]" redirect="[url]" />
</customerrors>
```

The defaultredirect attribute (optional) indicates the default url that a browser should be redirected to if an error occurs. The mode tag attribute indicates whether or not custom errors are enabled, disabled, or only shown to remote machines.

The <customerrors> section tag supports multiple <error> sub-elements that each supports two attributes:

Directive	Description
Statuscode	Indicates an error status code that should cause a browser to be directed to a non-default error page.
Type	The URL to which the browser should be redirected in the event when an error occurs.

**Example**

```
<configuration>
    <customerrors defaultredirect="genericerror.htm" mode="remote">
        <error statuscode="500" redirect="InternalError.htm" />
    </customerrors>
</configuration>
```

**Customer Error Message Directive**

The ErrorPage directive will redirect an ASP page to a custom error page. The following directive will redirect **any** ASP.NET page to **any** custom error page for **any** error.

**Example**

```
<%@page Errorpage = "/custom_error.aspx"%>
```

**Practice 14.3**

The following example shows how the configuration settings override two specific http status codes, and then falls back to a default page for custom error.

```
//default.aspx
<html>
<head>
</head>
<body>
    <form method="post" action="eventlog2.aspx" name="form1" id="number">
        <asp:Button id="abutton" type="submit" text="Click Me to generate an
error"runat="server" />
    </form>
</body>
```

```
</html>

//eventlog2.aspx
<html>
<script language="C#" runat=server>
void Page_Load(Object sender,EventArgs e)
{
    int x,y,z;
    x = 1;
    y = 0;
    z = x/y;
}
</script>
<body>
    Stuff was added to the event log
</body>
</html>

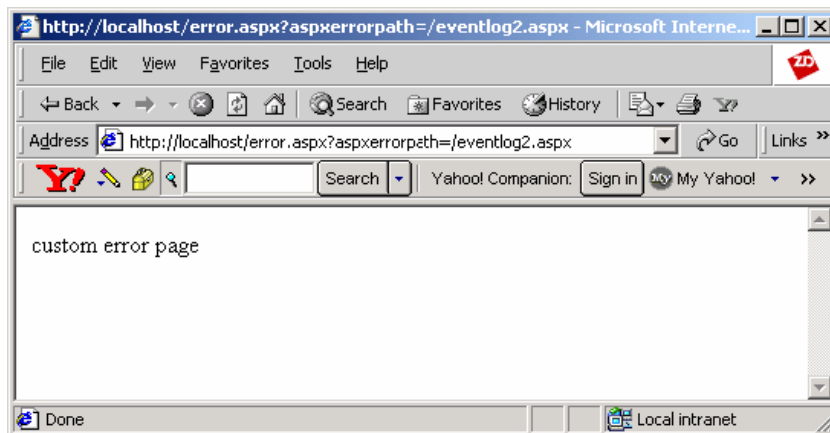
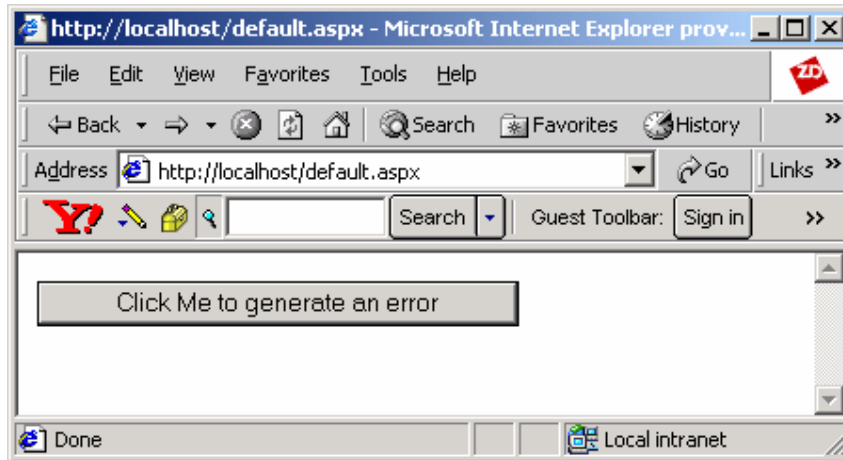
//403page.aspx
<html>
<head>
</head>
<body>
    <h1>403 Error page</h1>
</body>
</html>

//404page.aspx
<html>
<head>
</head>
<body>
    <h1>404 Error page</h1>
</body>
</html>

//customerrorpage.aspx
<html>
<head>
</head>
<body>
    <h1>custom error page</h1>
</body>
</html>

//config.web
<configuration>
    <customerrors mode="On" defaultredirect="customerrorpage.aspx">
        <error statusCode="404" redirect="404Page.aspx"/>
        <error statusCode="403" redirect="403page.aspx"/>
    </customerrors>
</configuration>
```





When the button is submitted, it links to the custom error page since unpredictable error (ie infinity) appears in eventlog2.aspx file.

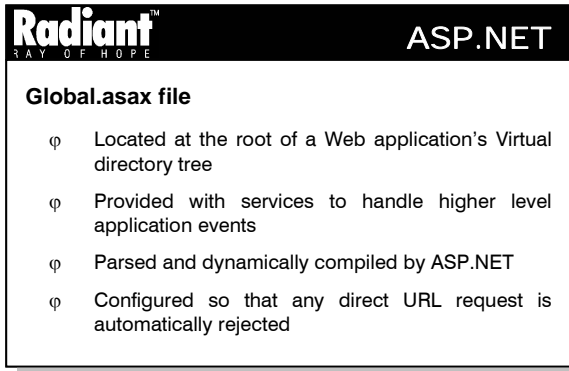
Custom error support can be added into an ASP.NET application very easily. Web page can be any type of file: .htm, .aspx, .asp, etc. The configuration settings can be modified within the application's config.web file to point to this file. The "defaultredirect" attribute of the <customerrors> tag defines the "default" page that a user will be redirected to in the event of an error. Optionally, the default value with alternative pages is used to redirect depending upon the http status code of the response (for example: redirect to a special "file not found" error page, "illegal access" error page, "server crash" error page, etc).

Custom error pages are very useful for production scenarios. It can often be difficult to deal with custom error pages during development. Since bugs are expected during development, the developer would want to see the actual error message. To help with this, the <customerrors> tag supports a "mode" attribute with three values:

- **On** - Indicates custom error page is always sent out
- **Off** - Indicates custom error page is never sent out (you always see the original error message)

- **Remoteonly** - Indicates custom error page is only sent out when remote browsers hit the site (developers hitting the site on the actual machine see the detailed error message)

### Global Exception Handler in ASP.NET



The “application level” logic and event handling code can be added into the Web applications. This code is responsible for handling higher-level application events - **Application\_Start**, **Application\_End**, **Session\_Start**, **Session\_End**, etc. The **Global.asax** file that is being located at the root of a particular Web application’s Virtual directory tree provides these services. ASP.NET automatically parses and compiles this file into a dynamic .NET Framework class - that extends the **HttpApplication** base class.

The Global.asax file is parsed and dynamically compiled by the ASP.NET into a .NET Framework class for the first time when any resource or URL within its application namespace is activated or requested. The Global.asax file itself is configured so that any direct URL request is automatically rejected.

Implementing a global exception handler involves four steps.

- The first step is to modify the config.web file to route the users of the Web application to a friendly error page. The following line should be added to the config.web file of the Web application:
 

```
<configuration>
  <customerrors mode="on" defaultredirect="error.aspx" />
</configuration>
```
- The second step is to build the error.aspx page that user will be re-directed for an exception
 

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

<%
Response.Write("custom error page");
%>
```
- The third step is to build a global exception handler in which an event handler is added to the global.asax file to capture all exceptions that occur in the Web application

### Error Event Handler in Global.asax

```
<%@ Application Language="C#" %>
<%@ Import Namespace="System.Diagnostics" %>
<script language="C#" runat="server">
void Application_Error(object sender, EventArgs e)
{
    Exception ex = Context.Error.GetBaseException();
```

```
EventLog.WriteEntry("Test Web",
    "MESSAGE: " + ex.Message +
    "\nSOURCE: " + ex.Source +
    "\nFORM: " + Request.Form.ToString() +
    "\nQUERYSTRING: " + Request.QueryString.ToString() +
    "\nTARGETSITE: " + ex.TargetSite +
    "\nSTACKTRACE: " + ex.StackTrace,
    EventLogEntryType.Error);
}
```

</script>

The `EventLog` class in `System.Diagnostics` namespace is to write exception details to the Windows 2000 Event Log. The `Application_Error` event handler is one of the event handlers which is supported by the `HttpApplication` class.

An `Exception` object instance is initialized through a call to `Context.Error.GetBaseException()`. The `Context` object is one of the intrinsic objects available in any ASP.NET page (including `global.asax`). It contains an `Error` property that gets populated when an exception occurs. The `Error` property of the `Context` object will simply contain a reference to a generic `HttpException`. `GetBaseException()` method is called in order to get access to the original exception.

There are three methods by which a reference can be obtained to the exception that occurred in ASP.NET page.

- `Context.Error.GetBaseException()`
- `Server.GetLastError().GetBaseException()` or
- `Context.ApplicationInstance.LastError.GetBaseException()`

`WriteEntry()` method of the `EventLog` class accepts three parameters. The first parameter is the source of the error. It will appear in the "Source" field of the Windows 2000 Event Log Viewer. The second parameter is the log data itself. The third and final parameter to the `WriteEntry()` method is an enumeration of type `EventLogEntryType`.

### Global Exception Handler Implementation

```
<%@ Page Language="C#" %>
<script language="C#" runat="server">
protected void button1_click(object sender, EventArgs e)
{
    try
    {
        // to generate fictional exception
        int x = 1;
        int y = 0;
        int z = x / y;
    }
    catch(DivideByZeroException ex)
    {
```

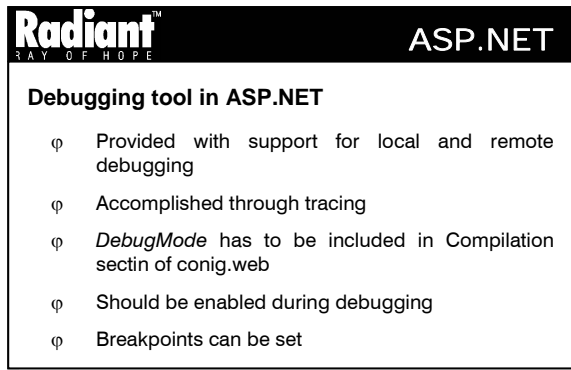
```

        throw(ex);
    }
}
</script>
<form runat="server">
    <asp:textbox id="textbox1" text="foobar" runat="server" />
    <asp:button id="button1" onclick="button1_click"
        text="click me" runat="server" />
</form>

```

The code above defines a Web Form with a textbox and a button. When the button is clicked, it fires the `button1_click` event handler. Generate a `DivideByZeroException` intentionally which will be caught by the catch block. `throw(ex)` is called to pass the exception to the global exception handler in order to be logged in the Windows 2000 Event Log. The `defaultredirect` attribute sets in the `config.web` file re-directs to the `error.aspx` page to display the custom error page.

## 14.4 Debugging



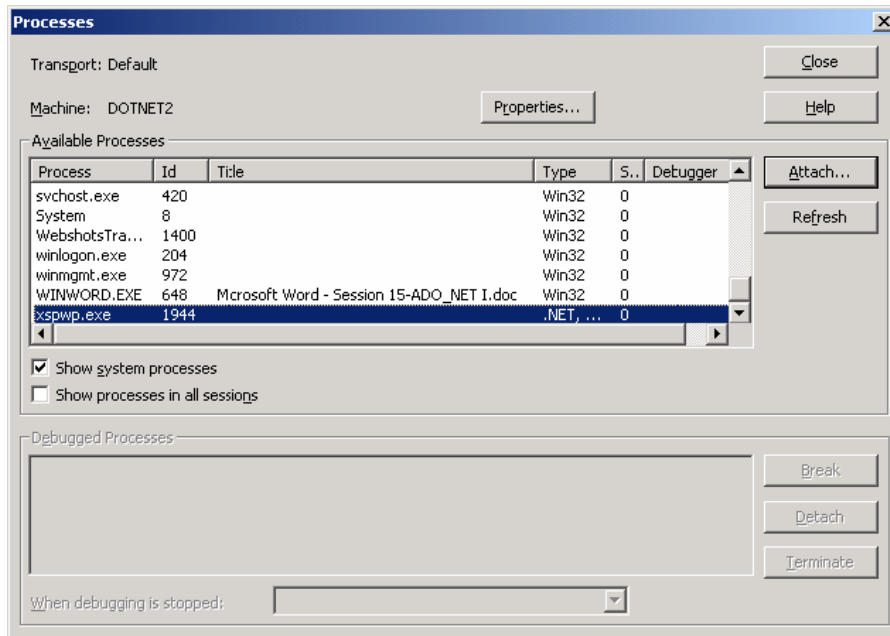
ASP.NET is supplied with a full-featured debugging tool, optimized for use with ASP.NET. It is also provided with support for local and remote machine debugging - including the setting of breakpoints and the ability to single-step through a code.

Debugging in ASP.NET is accomplished through tracing. An ASP.NET application is dynamically compiled at runtime. Symbols like `.pdb` files tell the debugger how to find the original source files for a binary, as well as how to map breakpoints in code to lines in those source files. To configure an application to compile with symbols, the `DebugMode` attribute has to be included in the `Compilation` section of the `config.web` file at the application's root directory. This setting should only be enabled during debugging an application, as it will significantly affect the performance of the application.

```
<configuration> <compilation debugmode="true"/></configuration>
```

After `DebugMode` has been enabled for the application, the page is requested for debugging. This will ensure that the ASP.NET runtime process (`xspwp.exe`) is created and the application is loaded into memory. When attached to the `xspwp.exe` process all threads in that process are frozen. Under no circumstances should a live production application be debugged, since client requests will not execute normally until the debugger is detached.





**Figure 14.3**

The steps given below have to be followed to begin debugging:

- Launch the .NET Framework debugger, DbgUrt.exe
- Use the File...Open menu to open the source file for the page you wish to debug
- From the Debug menu, choose "Processes". The dialog as in Figure 14.3 given below will appear
- Find the "xspwp.exe" process and double-click it to bring up the "Attach to Process" dialog
- Make sure the application appears in the list of running applications, and select "OK" to attach
- Close the "Programs" dialog

### Setting Breakpoints

A breakpoint can be set in the page, by clicking the left-hand margin on a line containing an executable statement or function/method signature. A red dot will appear where the breakpoint is set. The breakpoint ensures that it is appropriately mapped to the correct application instance in the **xspwp.exe** process. The request to the page has to be re-issued from the browser. The visual debugger will stop at the breakpoint and gain the current window focus. From here it sets variable watches, view locals, stack info, disassembly, etc. The intrinsic objects on the Page, like Request, Response, and Session can be viewed by using "this" (C#) or "Me" (VB) in the watch window, as shown in Figure 14.4 below:

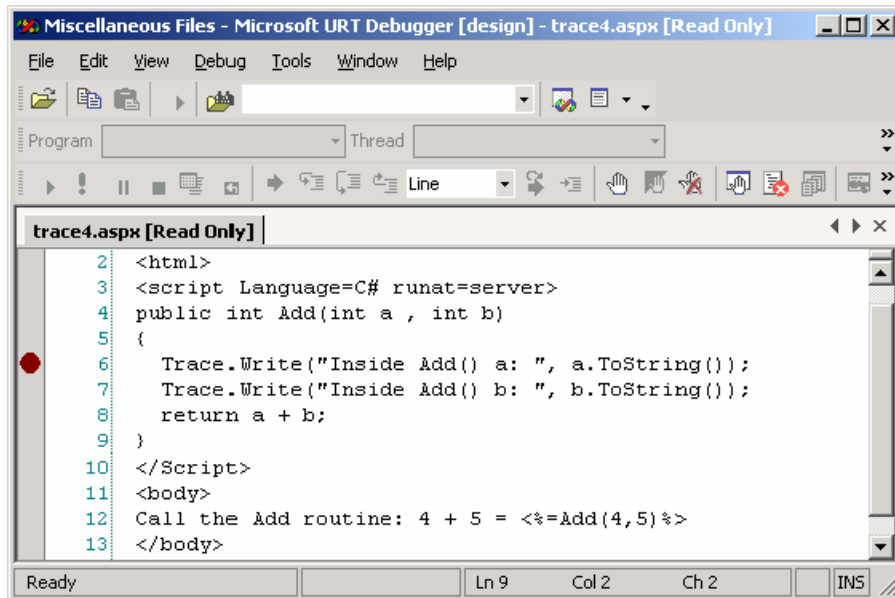


Figure 14.4

### Generating Symbols for Pre-compiled Components

To debug pre-compiled components such as business objects or code-behind files, compilation with symbolic information has to be done prior to debugging. Symbols for assemblies are typically found via a path-based search algorithm. The algorithm used by the PDB Library (mspdb69.dll) to find symbolic information is outlined below.

- Search same path as assembly (this is the normal location for pdb files). For local assemblies, place the symbols (.pdb files) in the application's /bin directory along with the dlls.
- Search path as specified in the PE file (the NB10 debug header).
- Search NT symbol file locations (environment variables NT\_SYMBOL\_PATH and NT\_ALT\_SYMBOL\_PATH).

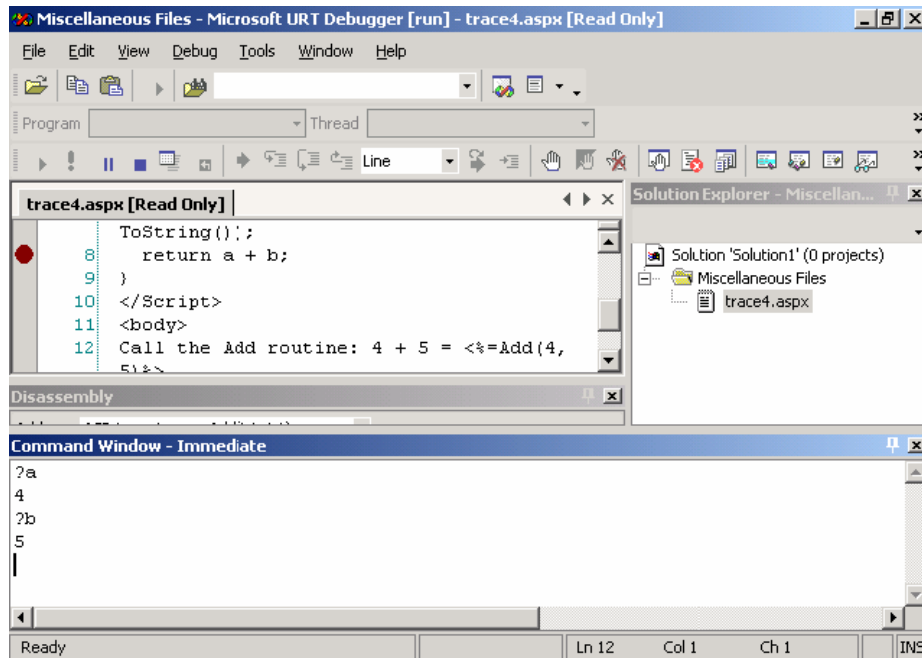


### Practice 14.4

The following example illustrates how a code can be debugged.

```
<%@page debug="true" trace="true"%>
<html>
<script Language=C# runat=server>
public int Add(int a , int b)
{
    Trace.WriteLine("Inside Add() a: ", a.ToString());
    Trace.WriteLine("Inside Add() b: ", b.ToString());
    return a + b;
}
</script>
<body>
    Call the Add routine: 4 + 5 = <%=Add(4,5)%>
</body>
```

```
</html>
```



To execute the above example, the debugger has to be run. The debugger is present in the path “Program files/microsoft.net/frameworkSDK/guidebug/dbgurt.exe” in the drive where the .NET Framework is installed. The file trace4.aspx has to be opened from the debugger and the “Processes” option of the “Debug” menu should be selected. A dialog box is opened in which there is a list box and two check boxes. The checkbox “Show system processes” should be selected. From the list, **xspwp.exe** (which executes the ASPX files) should be selected. On clicking OK, the dialog box is closed. A break point should be set at the point where debug has to be carried out. On executing the trace4.aspx file, the output is not visible initially. It will be visible only when the code is run in the debugger. The Application Trace can be viewed by giving the path “http://localhost/trace.axd” and Request details are viewed by clicking ViewDetails (which is in application trace).

### 14.5 Short Summary

- ASP.NET’s tracing feature is a powerful new way to trace the function of application
- Tracing the execution of an ASP.NET page can be done easily by using its methods Warn() and Write()
- Page-level tracing is enabled using a Trace=“true” attribute on the top-level Page directive
- Application-level tracing is enabled using a “trace” section in the configuration file at the application root directory
- Details for a series of requests may be accessed by requesting “trace.axd” from the application root

- The .NET Framework SDK visual debugger supports manual-attach debugging of processes on a local development computer. DebugMode is enabled by setting `<compilation debugmode="true"/>` in the application root's `config.web` file

### 14.6 Brain Storm

1. What is tracing? How is it useful to the programmer?
2. What is **Trace.axd** and why is it used?
3. How is **config.web** used?
4. Differentiate between page-level and application-level tracing.
5. How can the trace output be interpreted?



---

## Introduction to User Controls

---

### Objectives

In this lecture you will learn the following

- + Knowing about the User Controls
- + How to Create User Control?
- + How do you convert a Web Forms page into a User control

---

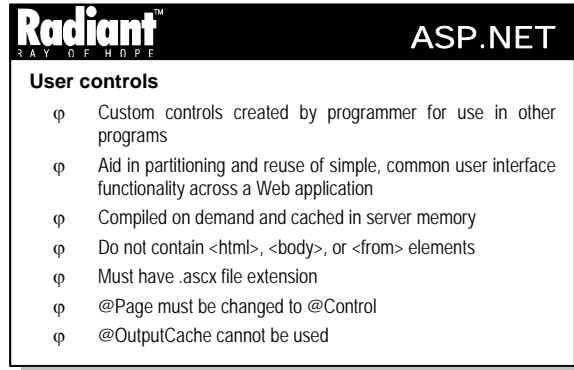
## Coverage Plan

---

Lecture 15
15.1 Snap Shot
15.2 Advantages of User Controls
15.3 Creating a User Control
15.4 Short Summary
15.5 Brain Storm

## 15.1 Snap Shot

This session introduces the learner to user controls and their advantages. It teaches how to create the user controls, add events and include them in a Web Forms page. It concludes with the procedure to be followed in order to include a user control programmatically in a Web page.



User controls are custom controls created by the programmer for use in other programs. ASP.NET enables creation of user controls. User controls aid in partitioning and reuse of simple, common user interface functionality across a Web application. They are compiled on demand and cached in server memory. User controls do not contain <html>, <body>, or <form> elements. These elements should be included in the page in which the user control is present. User controls must have .aspx file extension. If the page contains an @ Page directive, then it must be changed to @ Control directive. All the attributes (except the Trace attribute) supported by the @ Page directive are supported by the @ Control directive. Also, the @ OutputCache directive cannot be used in a user control.

## 15.2 Advantages of User Controls

### Object Model Support

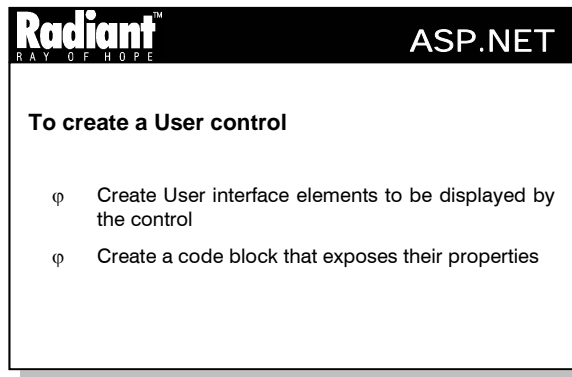
The Object model support offered by the user controls makes them more flexible than server-side includes. The properties of a user control can be programmed just like any other Web Forms control.

### Multiple Language Support on an Importing Page

User controls developed using different languages can be used in the same Web Forms page. For example, if one developer uses Microsoft Visual Basic to create a user control that imports data from an XML file, and another developer uses Microsoft C# to create a control that contains a login form, both the controls can be included in the same Web Forms page.

## 15.3 Creating a User Control

A user control can contain simple text or other Web Forms controls. It differs from an ordinary Web Forms page only in the fact that it does not contain the <html> and <body> elements. It should be noted that user controls that post events should not contain the <form runat="server"></form> element. Instead, it should be placed in the page containing the user control, around the user control tag.



Given below is the procedure to create a user control that receives the name, age and designation from the user.

- Create the User Interface elements to be displayed by the control

```
Name: <asp:TextBox id="name" runat="server" /> <br>
Age: <asp:TextBox id="age" runat="server" /> <br>
Desg: <asp:ListBox id="desg" runat="server" >
    <asp:ListItem>Manager</asp:ListItem>
    <asp:ListItem>Dep. Manager</asp:ListItem>
    <asp:ListItem>Asst. Manager</asp:ListItem>
    <asp:ListItem>Clerk</asp:ListItem>
    <asp:ListItem>Cashier</asp:ListItem>
</asp:ListBox>
<br>
```

- Create a code block that exposes the properties of the text boxes and the list box declared in the previous step so that they can be manipulated programmatically when they are inserted as a user control into a Web Form. This makes it possible to manipulate these properties declaratively or dynamically in any Web Forms page in which it is included

```
<script language="C#" runat="server">
public String UserName {
    get {
        return name.Text;
    }
    set {
        name.Text = value;
    }
}

public String UserAge {
    get {
        return age.Text;
    }
}
```



```
        set {
            age.Text = value;
        }
    }

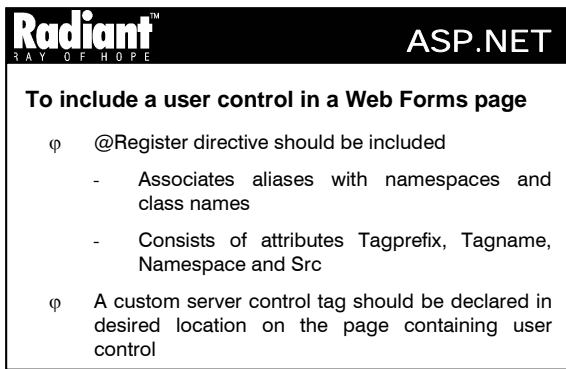
    public String UserDesg {
        get {
            return desg.SelectedItem.Text;
        }
        set {
            desg.SelectedIndex = System.Int32.Parse(value);
        }
    }
}
</script>
```

### Converting a Web Forms Page into a User Control

If a Web Forms page has to be converted into a user control so that its functionality can be used throughout another application, then the following alterations should be made to it:

- All <html>, <body> and <form> elements have to be removed from the page
- If there is an **@Page** directive in the Web Form it should be replaced with an **@Control** directive
- The control must be given a meaningful name and the file extension must be changed from **.aspx** to **.ascx**

### Including a User Control in a Web Forms Page



**Radiant™**  
RAY OF HOPE

**ASP.NET**

**To include a user control in a Web Forms page**

- ☐ **@Register** directive should be included
  - Associates aliases with namespaces and class names
  - Consists of attributes Tagprefix, Tagname, Namespace and Src
- ☐ A custom server control tag should be declared in desired location on the page containing user control

The following steps should be carried out in order to include a user control in a Web Forms page.

- An **@Register** directive should be included

The **@Register** directive associates aliases with namespaces and class names for precise notation in custom server control syntax. It consists of the attributes Tagprefix, Tagname, Namespace and Src.

  - **Tagprefix** is an alias to associate with a namespace
  - **Tagname** is an alias to associate with a class
  - **Src** is the location of the user control

- The **Tagprefix:TagName** pair will be used when an instance of the control is declared on another page

For example, the following code registers a control defined in the file **user1.ascx**, which has been given the tag prefix **User** and the tag name **Info**.

```
<%@ Register TagPrefix="User" TagName="Info" Src="user1.ascx" %>
```

- A custom server control tag should be declared in the desired location on the page containing the user control

For example, the following lines declare the control imported in the previous step:

```
<html>
<body>
  <form runat="server">
    <User:Info id="FirstUser" runat="server"/>
  </form>
</body>
</html>
```

The following example creates and uses a simple user control.



### Practice 15.1

This example creates a user control containing two text boxes, one to enter the user's name and another to enter the user's age. The user control is inserted into a page which also contains a button.

**The user control named "ctrl.ascx":**

```
Name: <asp:TextBox id="name" runat="server" /> <br><br>
Age: <asp:TextBox id="age" runat="server" />
```

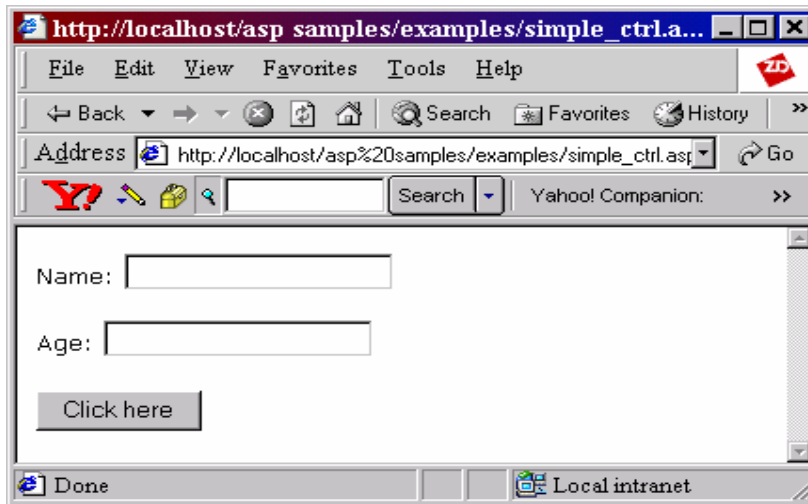
**The Web Forms page:**

```
<%@ Register TagPrefix="user" TagName="c1" Src="ctrl.ascx" %>
<html>
<body>
  <form runat="server">
    <user:c1 id="c" runat="server"/>
    <p>
      <asp:button id="btn" Text="Click here" runat="server" />
    </p>
  </form>
</body>
</html>
```



The Web Forms page should be run and not the user control file.





The above example creates a simple user control and saves it in the file named `ctrl.ascx`. The Web Forms page registers the control using the `@Register` directive, assigning the `TagPrefix` as `user` and `TagName` as `c1`. Then an instance of the control is created with the id `c`, along with an ASP button server control, and displayed.

### Manipulating User Control Properties

Once a user control has been created with the specified properties, it is possible to change the values of these properties both declaratively and in response to control events.

**Radiant™**  
RAY OF HOPE

**ASP.NET**

**To manipulate user control properties declaratively**

- ☐ Property name and a value have to be declared as an attribute/value pair in ActiveX control tag
- ☐ Value has to be assigned as the same type for the property as declared on the control

To manipulate user control property values declaratively, the property name and a value have to be declared as an attribute/value pair in the ActiveX control tag. The value has to be assigned as the same type for the property as declared on the control. For example, the following code establishes default settings for the properties created on the `User:Info` user control:

```
<Acme:Login id="FirstUser" BackColor="Gray" UserName = "Enter the user's name here" UserAge = "Enter the user's age here" runat="server" />
```

## Handling User Control Events

**Radiant™**  
RAY OF HOPE

**ASP.NET**

**Handling events in user controls is**

- ⊕ Almost similar to handling events from any other control
- ⊕ Event-handler can be included in
  - Page containing the control (or)
  - User control

Handling events in user controls is almost similar to handling events from any other control in Web Forms pages. The event handler can be included either in the page containing the control, or in the user control itself. In both the cases the event-handling code is the same, but there are a few precautions that should be taken care of before including event handlers in the user control.

While the user control's properties are exposed on the page containing the user control, the user control cannot access properties of other controls on the page. So all the functionality must be included within the user control for it to work as intended.

To handle the event in the page containing the user control, a code declaration block has to be included in the head of the Web Forms page in which the user control is included and an event-handling code has to be written to interact with the form.

To handle the event in the user control itself, a code declaration block has to be included in the user control that contains event-handling code for the form. All the controls involved in the event should be included in the control itself.

The following example creates a user control, handles the event in the control and uses it in a Web Forms page.

### Adding User Controls Programmatically

Like Web Forms controls, the user controls can also be instantiated programmatically using the **Page.LoadControl** method. The type of the control will be of the form *filename\_extension*. For instance, the user control created above which is saved as **user1.ascx** will belong to the type **user1\_ascx**. This is necessary to set the individual properties on the control.

**Radiant™**  
RAY OF HOPE

**ASP.NET**

**To add a user control programmatically**

- ⊕ Control instance has to be created
- ⊕ Property values have to be assigned as necessary
- ⊕ Control has to be added to Controls collection for the particular instance

To add a user control programmatically, either in a code-behind file or in a code declaration block in an **.aspx** file, a control instance has to be created, the property values have to be assigned as necessary, and the control has to be added to the Controls collection for the particular instance of the Page class.

For example, in the following code, **MyUserCtrl.aspx** is instantiated with its **BackColor** property set to **yellow**.

```
Control c = LoadControl("UserCtrl.aspx");
((UserCtrl.aspx)c).BackColor = "beige";
Page.Controls.Add(c);
```



### Practice 15.2

The following example adds a user control programmatically to a Web Forms page.

**The user control named 'user1.aspx':**

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

public String UserName {
    get {
        return name.Text;
    }
    set {
        name.Text = value;
    }
}

public String UserAge {
    get {
        return age.Text;
    }
    set {
        age.Text = value;
    }
}

public String UserDesg {
    get {
        return desg.SelectedItem.Text;
    }
    set {
        desg.SelectedIndex = System.Int32.Parse(value);
    }
}
</script>

Name: <asp:TextBox id="name" runat="server" /> <br><br>
Age: <asp:TextBox id="age" runat="server" /> <br><br>
Desg: <asp:ListBox id="desg" runat="server" >
    <asp:ListItem>Manager</asp:ListItem>
    <asp:ListItem>Dep. Manager</asp:ListItem>
```

```

<asp:ListItem>Asst. Manager</asp:ListItem>
<asp:ListItem>Clerk</asp:ListItem>
<asp:ListItem>Cashier</asp:ListItem>
</asp:ListBox>

```

### The Web Forms page:

```

<%@ Register TagPrefix="User" TagName="Info" Src="user1.ascx" %>

<head>
<script runat="server" language="C#">

void Ctrl(Object sender, EventArgs E) {
    Control c = LoadControl("user1.ascx");
    ((user1_ascx)c).UserName = "Aishwarya";
    ((user1_ascx)c).UserAge = "23";
    ((user1_ascx)c).UserDesg = "2";
    Page.Controls.Add(c);
}

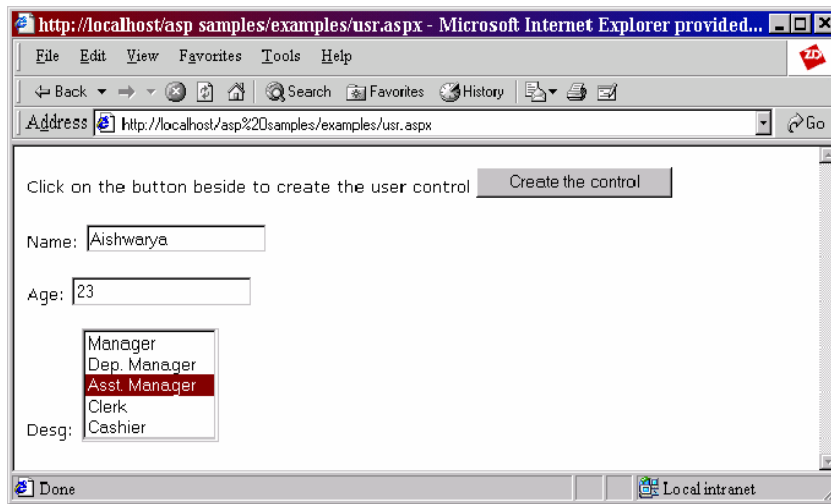
</script>
</head>

<body>
<form runat="server">
<asp:label id="title" Text="Click on the button beside to create the user
control" runat="server" />
<asp:button id="create" OnClick="Ctrl" Text="Create the control"
runat="server" />
</form>
</body>

```

In the above code, when the user clicks the "Create the control" button, its **OnClick** event is fired. This event calls the method **Ctrl**. This method creates an instance of the user control programmatically through the **LoadControl** method. Then it initializes the properties **UserName**, **UserAge** and **UserDesg** of the control. Then it adds the control to the **Controls** collection of the page.





### 15.4 Short Summary

- User controls are custom controls created by the programmer for use in other programs. ASP.NET enables creation of user controls
- A user control can contain simple text or other Web Forms controls
- The @Register directive associates aliases with namespaces and class names for precise notation in custom server control syntax
- Once a user control is created with specified properties, it is possible to change the values of these properties both declaratively and in response to control events
- Handling events in user controls is almost similar to handling events from any other control in Web Forms pages
- Like Web Forms controls, the user controls can also be instantiated programmatically using the Page.LoadControl method

### 15.5 Brain Storm

1. What is a user control? What are its advantages?
2. What is the difference between the @Page directive and the @Control directive?
3. What are the steps involved in converting a Web Forms page to a user control?
4. How can a user control be included in a Web page?
5. Which method is used to add a user control programmatically to a Web page?

❧

## Lecture 16

---

# ASP.NET Components

---

### Objectives

In this lecture you will learn the following

- + Knowing about the disadvantages of COM
- + Comparing COM and .NET
- + Deploying Components in ASP.NET



---

## Coverage Plan

---

### Lecture 16

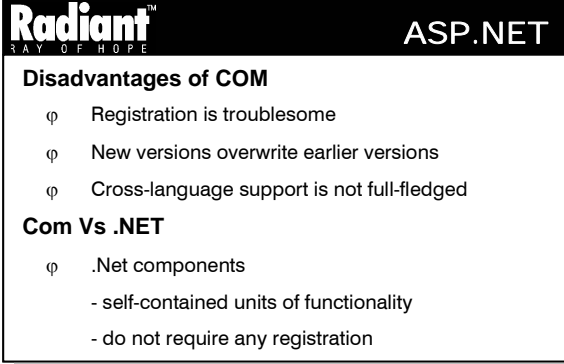
- 16.1 Snap Shot
- 16.2 Disadvantages of COM
- 16.3 ASP.NET Components
- 16.4 Components and System Architecture
- 16.5 Deploying Components in ASP.NET Application
- 16.6 Short Summary
- 16.7 Brain Storm

## 16.1 Snap Shot

This session is aimed at familiarising the learner with ASP.NET Components. COM rules the world of component the development. Many companies have invested a great amount of money and time in COM components. With the introduction of .NET, people are concerned about the future of COM. Microsoft has recognized this fact and provided a means to use classic COM components in .NET code and vice versa.

Encapsulating logic in business components is an essential part of any real-world application. In ASP.NET, business objects are the building blocks for multi-tiered web applications, such as those with a layer for data access or common application rules.

## 16.2 Disadvantages of COM



**Radiant™**  
RAY OF HOPE

**ASP.NET**

**Disadvantages of COM**

- ☐ Registration is troublesome
- ☐ New versions overwrite earlier versions
- ☐ Cross-language support is not full-fledged

**Com Vs .NET**

- ☐ .Net components
  - self-contained units of functionality
  - do not require any registration

Even though COM is the most successful technology to development components, it has its own shortcomings.

- **Registration:** Registration is the most troublesome part while using COM components. Every COM component stores information about it in Windows registry. The information is basically about GUIDs, CLSIDs, Path etc. Even though a COM component is to be used by only one application, this information is made available machine wide
- **Versioning:** Another problem with COM components is compatibility of version. Many times the newer versions of COM component overwrite its earlier version and breaks old applications
- **Cross-language Support:** It is said that the COM components developed in one language can be used for any other COM compatible tool. However, components developed in VB cannot be used easily in VC++

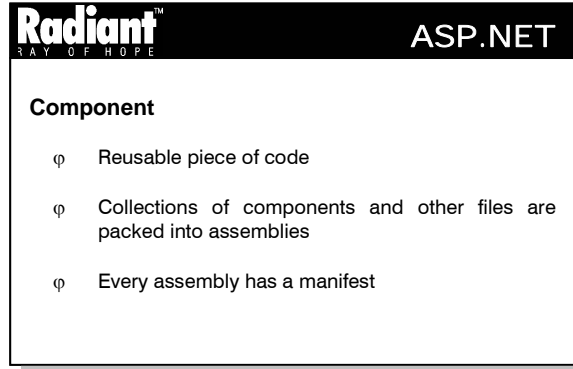
### Difference between COM and .NET Components

For a component to be called as COM compatible, it must support **IUnknown** interface. This interface is responsible for **reference counting** of the component. A COM component stores all the information about various methods and their parameters in a 'Type Library' (In case of VB type library information is embedded in the resulting EXE or DLL itself). To use COM component successfully, this information must be available along with some registration information.

In contrast, .NET components are very different. The component do not itself keep track of reference count. It is the responsibility of the CLR. Moreover, .NET components store all the information about the

component along with the assembly itself. This information is called as MetaData. .NET components are self-contained units of functionality that do not require any registration.

### 16.3 ASP.NET Components



Component is a reusable piece of code. Each CLR image carries the metadata for declarations, implementations, and references specific to that image, the image-specific metadata is referred to as **component metadata**, and the resulting component is said to be **self-describing**. Collections of CLR components and other files are packaged together for deployment into **assemblies**.

Every assembly has a **manifest** that declares the components that make up (or are allowed to make up) the assembly, the types to be exported, how type references are resolved, how assembly references are resolved, configuration rules, etc. An assembly may carry an explicit manifest or the manifest may be implicit.

ASP.NET attempts to solve the problems of previous versions by allowing components to be placed in a /bin directory, which is automatically found at run-time.

ASP.NET component offers the following advantages.

- **No Registration Required**

No registration is required to make an assembly available to pages in the application. It is available by virtue of its location in the /bin directory. Compiled code can be deployed by simply copying to the bin directory.

- **No Server Restart Required**

When any part of an ASP.NET application is changed (for example, replacing a .dll in /bin), new requests immediately begin the execution against the changed file(s). The Web server does not require a restart when the application is changed, even when replacing compiled code.

- **No Namespace Conflicts**

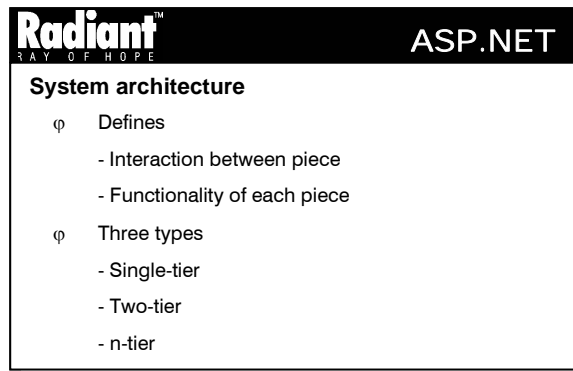
Each assembly loaded from /bin is limited in scope to the application in which it is running. This means that peer applications could potentially use different assemblies with the same class or namespace names, without conflicting.

- **Managed Execution**

Codes that executes under the control of ASP.NET are called as managed codes. The memory consumed by such a code is reclaimed automatically by CLR through garbage collection mechanism.

COM components are not under the control of .NET CLR and hence are treated as unmanaged code. Tasks like automatic memory management are not available to such components.

## 16.4 Components and System Architecture



One of the key elements of any application design is the system architecture. The system architecture defines how pieces of an application interact with each other, and the functionality of each piece. There are three main classes of application architecture. They can be characterized by the number of layers between the user and the data. Each layer generally runs on a different system or in a different process space on the same system than the other layers.

The three types of application architecture are

- Single-tier (or monolithic)
- Two-tier
- n-tier, where n can be three or more

### Single-tier WebForm

The monolithic application consists of a single application layer that supports the user interface, the business rules, and the manipulation of data. The data itself could be physically stored in a remote location, but the logic for accessing it is part of the application. Microsoft Word is an example of a monolithic application. The user interface is an integral part of the application. The business rules, such as how to paginate and hyphenate, are also part of the application. The file access routines, that manipulate the data of the document, are also part of the application. Even if there are multiple DLLs that handle the different functionality, it is still a monolithic application.

### Two-tier WebForm

There are two types of two-tier application

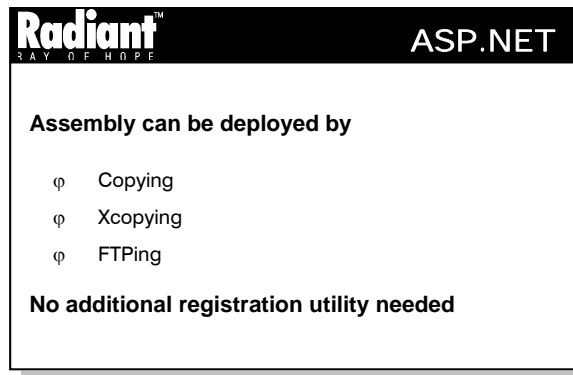
In the first type, the business rules and user interface remain as part of the client application. The data retrieval and manipulation is performed by another separate application which is usually found on a physically separate system. This separate application could be SQL Server or Oracle, which functions as a data storage device for the application. This type of application is widely used in the traditional client-server types of applications. PowerBuilder or Visual Basic with Oracle as backend are examples of tools that can be used to create client-server systems.

In the second type of two-tier application, the business rules are executed on the data storage system. This is the case in applications that use stored procedures to manipulate the database. A stored procedure is a database function that is stored in the database server. It can be executed in one of the two ways. A client application can explicitly call a stored procedure, which would then be run on the server. Alternatively trigger can also execute a stored procedure, which is the occurrence of a specific event in the data.

### Three-tier WebForm

With three-tier applications, the business rules are removed from the client and are executed on a system in between the user interface and the data storage system. The client application provides user interface for the system. The business rules server ensures that the business processing which includes validation of user input is done correctly. It serves as an intermediary between the client and the data storage. In this type of application, the client would never access the data storage system directly. This type of system allows for any part of the system to be modified without having to change the other two parts. Since the parts of the application communicate through interfaces, and then as long as the interface remains the same, the internal workings can be changed without affecting the rest of the system.

## 16.5 Deploying Components in ASP.NET Applications



An assembly can be deployed into an application's local assembly cache by simply copying, xcopying, or FTPing the appropriate file(s) into a directory that has been marked as an "assembly cache location" for that particular application. No additional registration utility is needed to run, once the appropriate files are copied and no reboot is necessary. By default, an ASP.NET application is automatically configured to use the "bin" sub-directory which is located immediately under the application root as its local assembly cache. The bin directory is also configured to deny any browser access .

ASP.NET applications are identified by a unique URL and live in the file system of the Web server. ASP.NET can use shared assemblies, which reside in the global cache, and application-specific assemblies, which reside in the "bin" directory of the application's virtual root. ASP.NET applications run in the context of Application Domains, which provide isolation and enforcement of security restrictions. Classes can be dynamically referenced through "classname, assemblyname". ASP.NET uses shadow-copies of assembly files to avoid locking and monitors the files, so that changes are picked up immediately.



## Practice 16.1

---

The following application demonstrates a simple component.

```
//comp2.cs
namespace comp2 {
using System;
using System.Text;

public class HelloWorld
{
    public String hello()
    {
        return "Hello World!";
    }
}
}
```

- Save the above file as comp2.cs
- Compile it using the command:

```
csc /t:library /r:System.dll /r:System.Text.dll comp2.cs
```

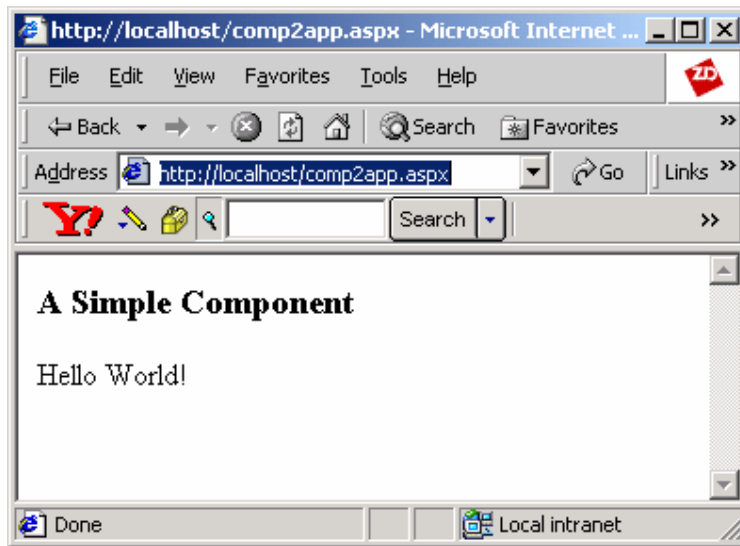
```
//comp2app.aspx
<%@ Import Namespace="comp2" %>

<html>

<script language="C#" runat="server">
public void Page_Load(Object sender, EventArgs E)
{
    HelloWorld hw = new HelloWorld();
    d.InnerHtml += hw.hello() + "<p>";
}
</script>

<body >
    <h3>A Simple Component</h3>
    <div id="d" runat="server"/>
</body>
</html>
```





### Practice 16.2

The following example is a two-tier Web Form that uses a component to connect to the database and execute a given query.

```
//comp1.cs
using System.Data;
using System.Data.SQL;
namespace comp1
{
    public class SQLRecords
    {
        string cst;
        string sst;

        public string SqlConnectionString    {
            get { return cst; }
            set { cst = value; }
        }

        public string SQLStatement
        {
            get { return sst; }
            set { sst = value; }
        }

        public SqlConnection conmethod()
        {
            SqlConnection con;
            con = new SqlConnection(SqlConnectionString);
            return con;
        }
    }
}
```

```

        public DataView getrecords()
        {
            SQLDataSetCommand cmd;
            cmd = new SQLDataSetCommand(SQLStatement, conmethod());
            DataSet ds;
            ds = new DataSet();
            cmd.FillDataSet(ds, "RETURNDATASET");
            return ds.Tables[0].DefaultView;
        }
    }
}

```

- Save the file as **comp1.cs**
- Compile it using the command:

```
csc /t:library /r:System.Data.dll comp1.cs
```

```
//complapp.aspx
```

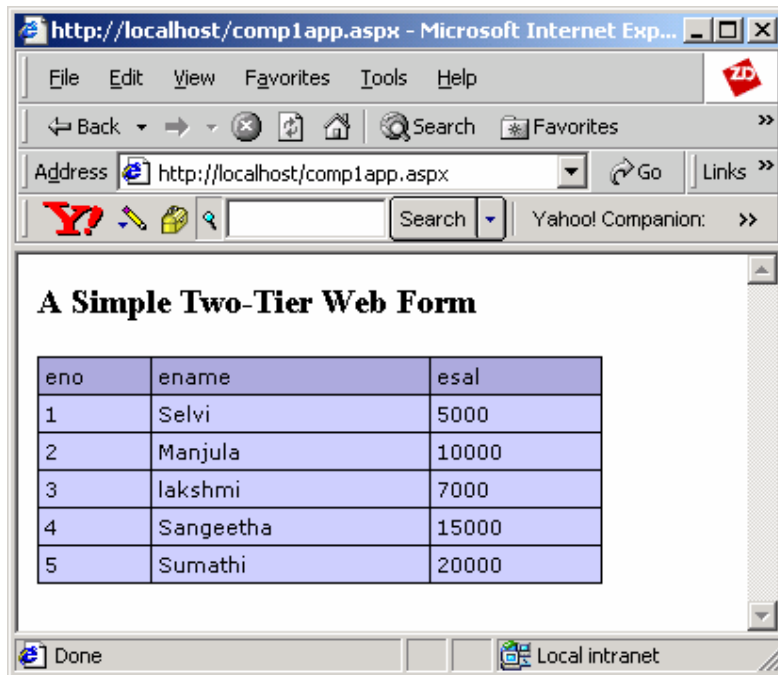
```

<%@ Import Namespace="comp1" %>
<html>
<head>
</head>
<script language="C#" runat="server">
public void Page_Load(Object sender, EventArgs E)
{
    SQLRecords sr;
    sr = new SQLRecords();
    sr.SQLStatement = "SELECT * FROM emptab";
    sr.SQLConnectionString = "server=localhost;uid=sa;database=master;pwd=";
    d.DataSource = sr.getrecords();
    d.DataBind();
}
</script>
<body>
    <h3>A Simple Two-Tier Web Form</h3>
    <p>
    <ASP:DataGrid id="d" runat="server"
        Width="300"
        BackColor="#ccccff"
        BorderColor="black"
        CellPadding=3
        CellSpacing="0"
        Font-Name="Verdana"
        Font-Size="8pt"
        HeaderStyle-BackColor="#aaaadd"
    />
    </form>
</body>
</html>

```







The component contains two properties namely **SQLConnectionString** and **SQLStatement** and two methods **connmethod()** and **getrecords()** in the **SQLRecords** class. The **connmethod()** creates and returns a **SQLConnection** object using the **SQLConnectionString** property. The **getrecords()** method uses the **connmethod()** to get the **ConnectionString**, establishes the connection, executes the query represented by the **SQLStatement** property and fills the DataSet **ds** with the results and returns the DefaultView of the DataSet.

The component is imported in the aspx file using the statement:

```
<%@ Import Namespace="comp1" %>
```

In the **Page\_Load** event of the Web Form, an instance of the **SQLRecords** class is created and the two properties **SQLConnectionString** and **SQLStatement** are assigned values. Then the **getrecords()** method is called to populate the DataGrid **d**.

## 16.6 Short Summary

- For a component to be called as COM compatible, it must support IUnknown interface. This interface is responsible for reference counting of the component
- Component is a reusable piece of code
- Every assembly has a manifest
- The three types of application architecture are single-tier (or monolithic), two-tier and n-tier, where n can be three or more

## 16.7 Brain Storm

1. Explain how ASP.NET component is advantageous over other components.
2. How is an ASP.NET component deployed?
3. In which directory are the ASP.NET components stored?
4. What are the advantages of three-tier Web applications over two-tier applications?
5. What is a managed code execution?

☞ ☞

## MS 3.2 ASP.NET

### Lecture 1 Introduction to ASP.NET

---

Web server- Virtual Directory- Introduction to ASP.NET -ASP.NET Architecture -ASP.NET Features - Advantages of ASP.NET -Dataflow in ASP.NET - A Simple ASP.NET Application.

### Lecture 2 HTML Server-Side Controls

---

Basics of Coding in ASP.NET -Structure of Code Blocks -HTML Server-Side Controls

### Lecture 3 ASP.NET Web Controls I

---

Web Controls-Web Controls Hierarchy - Button -Label - TextBox-CheckBox -RadioButton

### Lecture 4 ASP.NET Web Controls II

---

ASP.NET Image Controls - ImageButton - ASP.NET List Controls -LinkButton -Panel -AdRotator - Calendar

### Lecture 5 Validation Controls

---

Validation in ASP.NET - Validating for Multiple Conditions - Types of Validation - RequiredFieldValidator - CompareValidator - RangeValidator -RegularExpressionValidator - Custom Validator - Validation Summary Control

### Lecture 6 HttpRequest

---

HTML Form -HttpRequest Properties -Application Path - Browser -Client Certificate -ContentType - FilePath -RawUrl -Url - Cookies -Form - Query String

### Lecture 7 HttpResponse

---

HttpResponse Properties -BufferedOutput-SuppressContent -Charset -ContentType - ContentEncoding -StatusDescription -Cookies -IsClientConnected - StatusCode -HttpResponse Methods

### Lecture 8 Application, Session State Management and Cookies

---

HttpApplication -Managing Application - Global.asax -Applicatin Events in Global.asax - Application State -Session - Cookies - How to Create Cookie? - Server Object

### Lecture 9 ADO.NET I

---

Database -Benefits of the Database approach - DBMS -DBMS Standardization -Structured Query Language -Using SQL as a DataQuery Language -Manipulation of database in ASP.NET - Introduction to ADO.NET

### Lecture10 ADO.NET II

---

Managed Providers -Connection Object -Command Object -DataReader

### Lecture11 ADO.NET -III

---

DataSet - Working with Database -Constructors of DataSet -Methods of DataSet -Properties of DataSet -Events of DataSet -DataSet Constraints -DataRelation - Data Table - DataView

### Lecture12 Data Aware Controls

---

Databinder - Binding to Non-Database Datasources -Binding to DataBase -Repeater Control

Lecture13 DataGrid and DataList Server Controls

---

DataGrid Server Control -A Simple DataGrid - DataGrid Columns - Editing Items in DataGrid - Sorting Columns in DataGrid - Paging in DataGrid - Accesing a Database - DataList Server Control

Lecture14 Tracing, Error -Handling and Debugging

---

Tracing - Tracing with ASP -Tracing with ASP.NET - Tracing Levels -Trace Section -Error-Handling - Global Exception Handler in ASP.NET - Debugging - Setting Break Points

Lecture15 Introduction to User Controls

---

Advantages of User Controls -Creating a User Control -Converting a Web Forms Page into a User Control - Including a User Control in a Web Forms Page -Manipulating User Control Properties - Handling User Control Events -Adding User Controls Programmatically

Lecture16 ASP.NET Components

---

Disadvantages of COM - Difference between COM and .NET Components - ASP.NET Components - Components and System Architecture - Deploying Components in ASP.NET Applications