Lecture 15

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

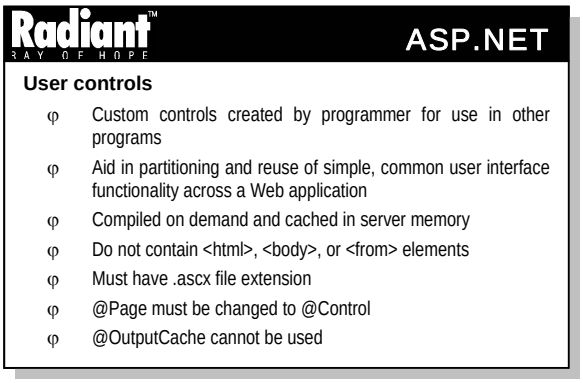
15.3 Creating a User Control

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 **.ascx** 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

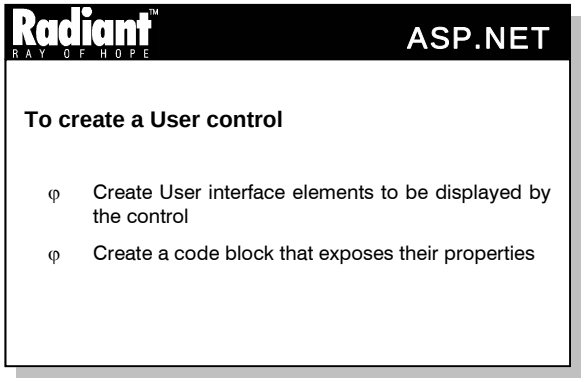
The Object model support offered by the user controls makes them more flexible than server-side includes. The properties of a user control can be programmed just like any other Web Forms control.

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;
     }
```

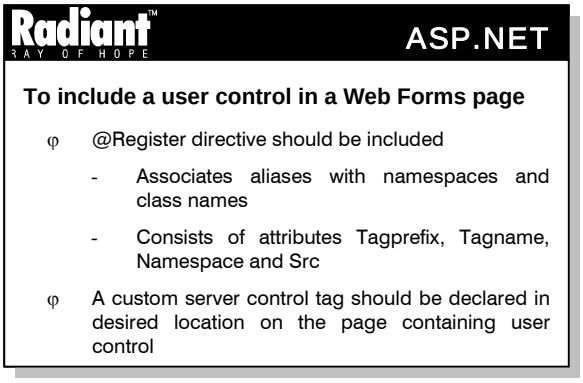
```
        set {
            age.Text = value;
        }
    }

    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



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":

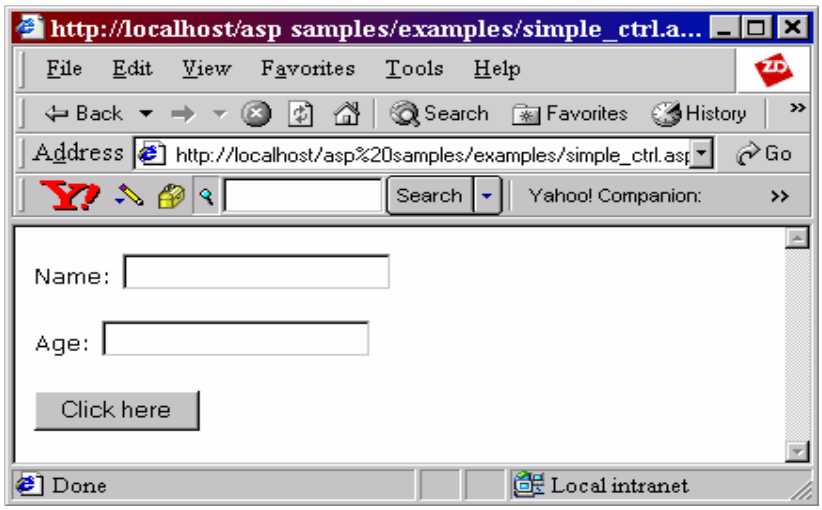
```
Name: <asp:TextBox id="name" runat="server" /> <br><br>
Age: <asp:TextBox id="age" runat="server" />
```

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

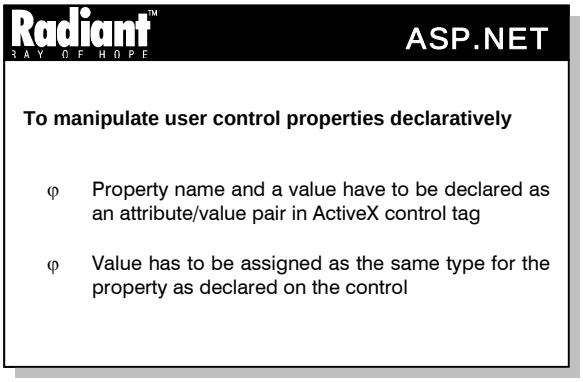
The Web Forms page should be run and not the user control file.



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

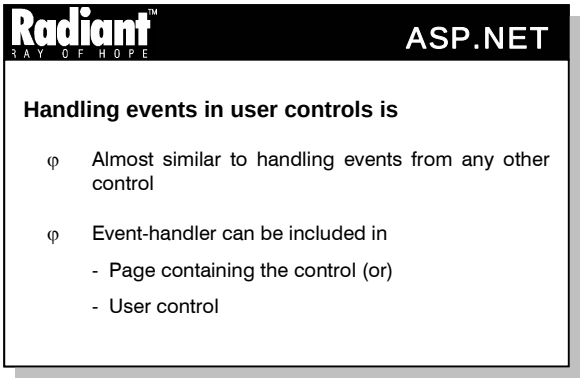
Once a user control has been created with the specified properties, it is possible to change the values of these properties both declaratively and in response to control events.



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



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.

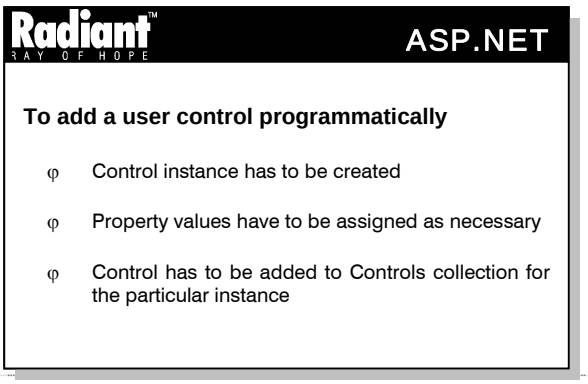
To handle the event in the page containing the user control, a code declaration block has to be included in the head of the Web Forms page in which the user control is included and an event-handling code has to be written to interact with the form.

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.



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.ascx** is instantiated with its **BackColor** property set to **yellow**.

```
Control c = LoadControl("UserCtrl.ascx");
((UserCtrl_ascx)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.ascx':

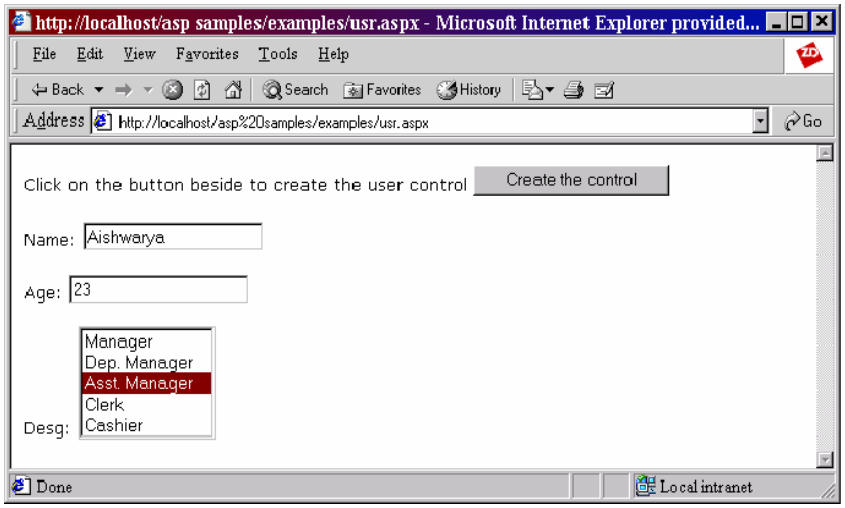
```
<script language="C#" runat="server">

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

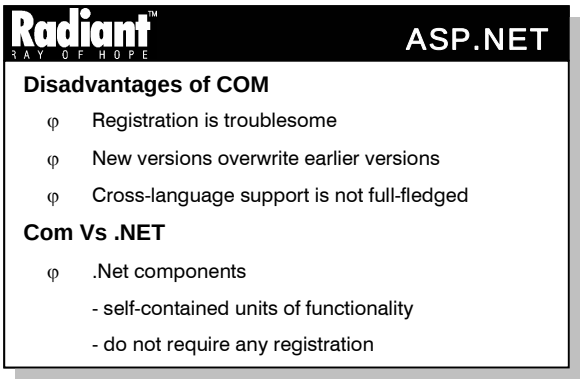
16.7 Brain Storm

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



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++

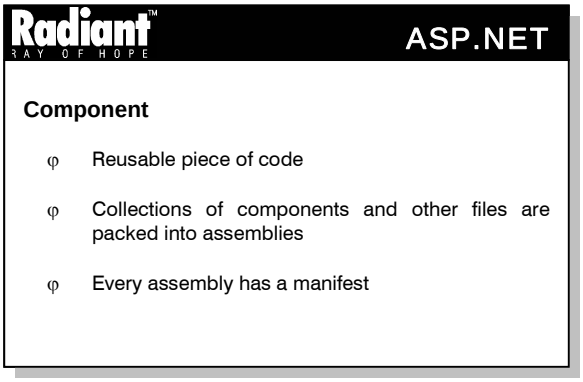
Difference between COM and .NET Components

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**

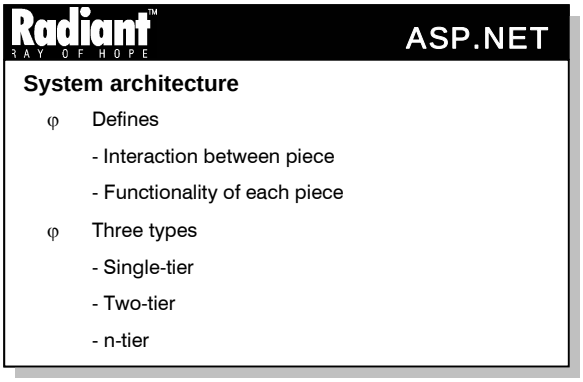
  Each assembly loaded from /bin is limited in scope to the application in which it is running. This means that peer applications could potentially use different assemblies with the same class or namespace names, without conflicting.

- **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

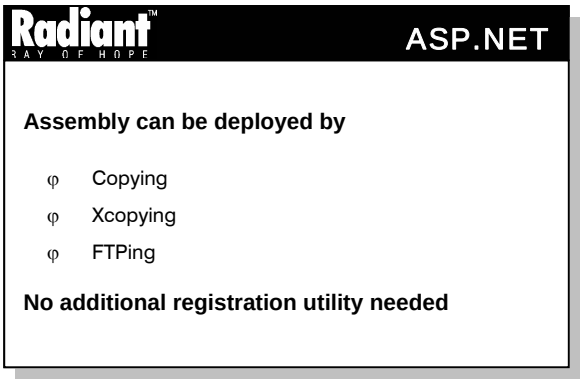
In the first type, the business rules and user interface remain as part of the client application. The data retrieval and manipulation is performed by another separate application which is usually found on a physically separate system. This separate application could be SQL Server or Oracle, which functions 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

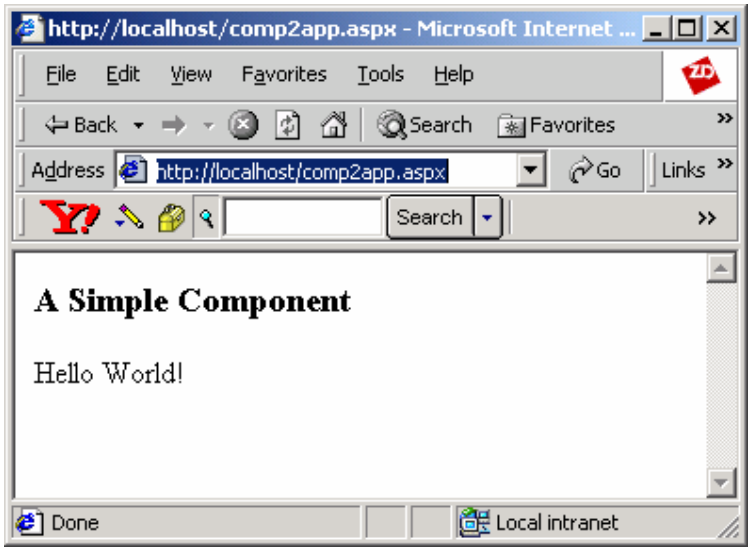
The following application demonstrates a simple component.

```
//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.

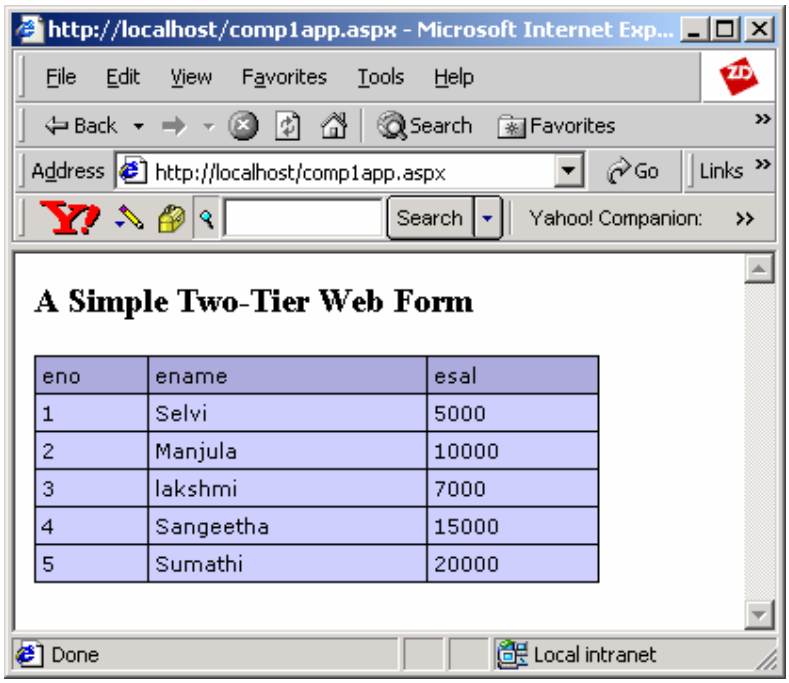
```
//comp1.cs

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
```

```
//comp1app.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 **conmethod()** and **getrecords()** in the **SQLRecords** class. The **conmethod()** creates and returns a **SQLConnection** object using the **SQLConnectionString** property. The **getrecords()** method uses the **conmethod()** 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?