Unit-V

XML:

- XML stands for Extensible markup language. By using XML we can easily retrieve and store data and display the data without using databases in our applications.
- If we need to display dynamic data in our application it will take time to connect databases and retrieve data from databases but if we use XML to store data we can do operations with xml files directly without using databases.
- If we store data in database that is incompatible to some of the computer applications but if we store data in XML format it will support all applications.
- It is independent for all software applications and it is accessible with all applications.
- XML is used to store the record in text format. User can store records in XML file in the form of node. As like SQL tables, XML file can be bind with Gridview, Datalist and other asp.net data control. The main benefit of XML is that you can define your own tag, with your own comfort.

The structure of XML is like below:

```
<Root Element>
<Parent1>
<Child1>...</Child1>
<Child2>...</Child2>
</Parent1>

<Parent2>
<Child1>...</Child1>
<Child2>...</Child2>
</Parent2>


</Root Element>
```

see the below example:

```
<Employees>
<employee>
<username>karan</username>
<id>001</id>
```

```
<salary>12000</salary>
</employee>
</employees>
```

What is an XML Element?

An XML element is everything from (including) the element's start tag to (including) the element's end tag.

```
<price>29.99</price>
```

An element can contain:

- text
- attributes
- other elements
- or a mix of the above

Empty XML Elements

- An element with no content is said to be empty.
- In XML, you can indicate an empty element like this:
- `<element></element>`
- You can also use a so called self-closing tag:
- `<element />`

XML Naming Rules

XML elements must follow these naming rules:

- Element names are case-sensitive
- Element names must start with a letter or underscore
- Element names cannot start with the letters xml (or XML, or Xml, etc)
- Element names can contain letters, digits, hyphens, underscores, and periods
- Element names cannot contain spaces.
- Any name can be used, no words are reserved (except xml).
- XML elements can have attributes, just like HTML.
- Attributes are designed to contain data related to a specific element.

XML Attributes Must be Quoted

- Attribute values must always be quoted. Either single or double quotes can be used.
- For a person's gender, the <person> element can be written like this:
- <person gender="female">
- or like this:
- <person gender='female'>

XML CLASS:

- The System.Xml namespace contains major XML classes.
- This namespace contains many classes to read and write XML documents.
- Here we are going to concentrate on reader and write class.
- These reader and writer classes are used to read and write XML documents.
- These classes are - XmlReader, XmlTextReader, XmlValidatingReader, XmlWriter, and XmlTextWriter.

1. XmlReader
2. XmlTextReader
3. XmlWriter
4. XmlTextWriter

Writing/storing data in xml file

1. Add XML File From Solution Explorer
2. Write following code

```
<?xml version="1.0" encoding="utf-8" ?>
<student>
    <record>
        <roll>10</roll>
        <name>deepa</name>
        <course>BCA</course>
    </record>

    <record>
        <roll>1</roll>
```

```
            <name>priya</name>
            <course>MCA</course>
        </record>
</student>
```

3. Add a web form take a gridview and display these 2 records in that.

Write following code in web form

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Data;
namespace xml_programs
{
   public partial class WebForm1 : System.Web.UI.Page
   {
      protected void Page_Load(object sender, EventArgs e)
      {
         DataSet ds = new DataSet();
         ds.ReadXml(Server.MapPath("XMLFile1.xml"));
         GridView1.DataSource = ds;
         GridView1.DataBind();
}

}
```

Output:

roll	name	course
10	deepa	BCA
1	priya	MCA

1. XMLWRITER CLASS:

- Xmlwriter is a class available in system.xml namespace.
- It is used to write an xml document

Example:

```
XmlWriter w= XmlWriter.Create
("C:\\Users\\Deepa\\Source\\Repos\\xml_programs\\xml_programs\\demo2.xml");
```

The XmlWriter class allows you to write XML to a file. This class contains a number of methods and properties that will do a lot of the work for you. To use this class, you create a new XmlTextWriter object.

Methods of XmlWriter:

METHOD	DESCRIPTION
WriteStartDocument	Writes the XML declaration with the version "1.0".
WriteEndDocument	Closes any open elements or attributes.
Close	Closes the stream.
WriteDocType	Writes the DOCTYPE declaration with the specified name and optional attributes.
WriteStartElement	Writes the specified start tag.
WriteEndElement	Closes one element.
WriteFullEndElement	Closes one element.
WriteElementString	Writes an element containing a string value.

Program

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Xml;
using System.Data;
namespace xml_programs
{
   public partial class XmlWriterClass : System.Web.UI.Page
   {
      protected void Page_Load(object sender, EventArgs e)
      {
      }

      protected void save_Click(object sender, EventArgs e)
      {
XmlWriter w= XmlWriter.Create
("C:\\Users\\Deepa\\Source\\Repos\\xml_programs\\xml_programs\\demo2.xml");
          w.WriteStartDocument();//body
          w.WriteStartElement("Students");//root

          w.WriteStartElement("Record");//subnode
          w.WriteElementString("roll", "10");
          w.WriteElementString("name", "neha");
          w.WriteElementString("course", "BSCIT");
          w.WriteEndElement();


          w.WriteStartElement("Record");//subnode
          w.WriteElementString("roll", "11");
          w.WriteElementString("name", "PRIYA");
          w.WriteElementString("course", "BCA");
          w.WriteEndElement();

          w.WriteEndElement();
          w.WriteEndDocument();
```

```
            w.Close();
            Response.Write("data saved");

        }
          protected void display_Click(object sender, EventArgs e)
        {
            GridView1.Visible = true;
             DataSet ds = new DataSet();
ds.ReadXml("C:\\Users\\Deepa\\Source\\Repos\\xml_programs\\xml_programs\\demo2.
xml");
            GridView1.DataSource = ds;
            GridView1.DataBind();
        }
    }
}
```

2. Xmlreader class:

- The XmlReader is a faster and less memory-consuming alternative. It lets you run through the XML content one element at a time, while allowing you to look at the value, and then moves on to the next element.

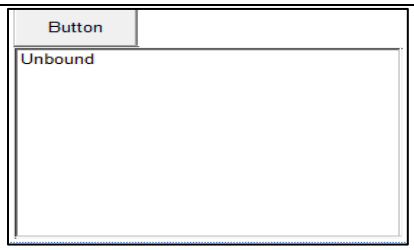
XMLFILE1.XML

```
<?xml version="1.0" encoding="utf-8" ?>
<student>
      <record>
            <roll>10</roll>
            <name>deepa</name>
            <course>BCA</course>
      </record>

      <record>
            <roll>1</roll>
            <name>priya</name>
      </record>
</student>
```

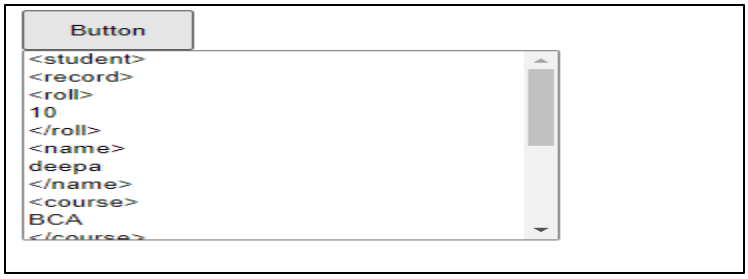
WEBFORM2.ASPX:

```
using System;
using System.Collections.Generic;
```

```
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Xml;
namespace xml_programs
{
    public partial class WebForm2 : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {

        }

      protected void Button1_Click(object sender, EventArgs e)
      {
       XmlTextReader xr = new
XmlTextReader(Server.MapPath("XMLFile1.xml"));
      while(xr.Read())
       {
       switch(xr.NodeType)
       {
       case XmlNodeType.Element:
                       ListBox1.Items.Add("<" + xr.Name+ ">");break;
       case XmlNodeType.Text:
                       ListBox1.Items.Add(xr.Value); break;
       case XmlNodeType.EndElement:
                       ListBox1.Items.Add("</" + xr.Name + ">");break;
         }
       }

        }
    }
}
```



OUTPUT:



XML VALIDATION:

Validation is a process by which an XML document is validated. An XML document is said to be valid if its contents match with the elements, attributes and associated document type declaration (DTD), and if the document complies with the constraints expressed in it. Validation is dealt in two ways. They are –

- Well-formed XML document
- Valid XML document

Well Formed XML Documents

An XML document with correct syntax is called "Well Formed".
The syntax rules were described in the previous chapters:

- XML documents must have a root element
- XML elements must have a closing tag
- XML tags are case sensitive
- XML elements must be properly nested
- XML attribute values must be quoted

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

Valid XML Documents

A "well formed" XML document is not the same as a "valid" XML document.
A "valid" XML document must be well formed. In addition, it must conform to a document type definition.
There are two different document type definitions that can be used with XML:

- DTD - The original Document Type Definition

- XML Schema - An XML-based alternative to DTD

A document type definition defines the rules and the legal elements and attributes for an XML document.

ASP.NET – Security

In ASP.NET, there are two closely interlinked concepts for the security of applications, one is authentication and another is Authorization. In the authentication process, some sort of identification is obtained from the users and using that identification to verify the user's identity. In authorization process is to allow an authenticated user access to resources.

1. Form authentication

```
<system.web>
<authentication mode="Forms">
<forms name="demo_cookie" loginUrl="form1.aspx" defaultUrl="welcome.aspx"
timeout="1">
            <credentials passwordFormat="Clear">
          <user name="deepa" password="deepa123"/>
           <user name="priya" password="125"/>
            </credentials>
</forms>
</authentication>
<authorization>
      <deny users="?"/>
</authorization>
 </system.web>
```

Explanation:

- `<forms name="demo_cookie" loginUrl="form1.aspx" defaultUrl="welcome.aspx" timeout="1">`
- `<deny users="?"/>`
- `<credentials passwordFormat="Clear">`

- **Name:** It is the name of the cookies.by default name is .aspxauth
- **loginUrl:** It is the page where user should re-directed if no valid authentication cookie is found .The deafult value is login.aspx.
- **timeout:** The expiry time of cookie by value is 30.

- **?** indicates anonymous user. If someone tries to access a page without login they are said to be anonymous user.
- **PasswordFormat="Clear":** Password will be in text format.

In out project there are 3 files:

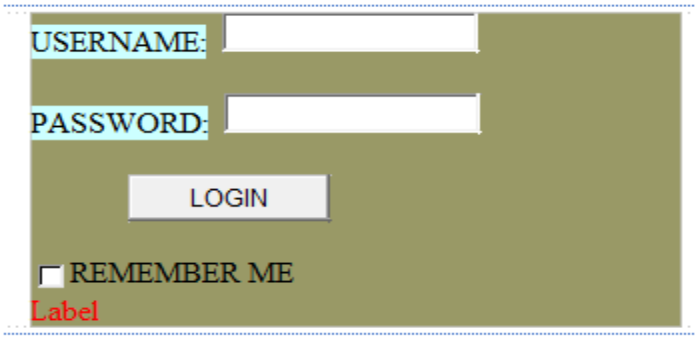
1. Form1.aspx [login page]
2. Welcome.aspx
3. Default.aspx

If we remove authorization setting from web.config file.

```
<authorization>
    <deny users="?"/>
</authorization>
```

Then anyone can access any page without login.But if you want to restrict someone from accessing the default or welcome page without login then we have to use authorization setting in web.config.

Form1.aspx:



On button click write down following code:

```
using System.Web.Security;
namespace Authentication
{
   public partial class form1 : System.Web.UI.Page
   {
      protected void Button1_Click(object sender, EventArgs e)
      {
         if(FormsAuthentication.Authenticate(TextBox1.Text,TextBox2.Text))
         {
            FormsAuthentication.RedirectFromLoginPage(TextBox1.Text, true);
         }
```

```
        else
        {
            Label3.Text = "invalid username or password";
        }
    }
  }
}
```

- All the file related to security class are available in System.web.security class.
- FormsAuthentication is a class and Authenticate is a method which accepts 2 parameter username and password enterd by user.
- Authenticate method verifies the passed username and password against the credential stored in web.config file.It returns true if username and password matches the credential stored in web.config.

```
<credentials passwordFormat="Clear">
            <user name="deepa" password="deepa123"/>
             <user name="priya" password="125"/>
</credentials>
```

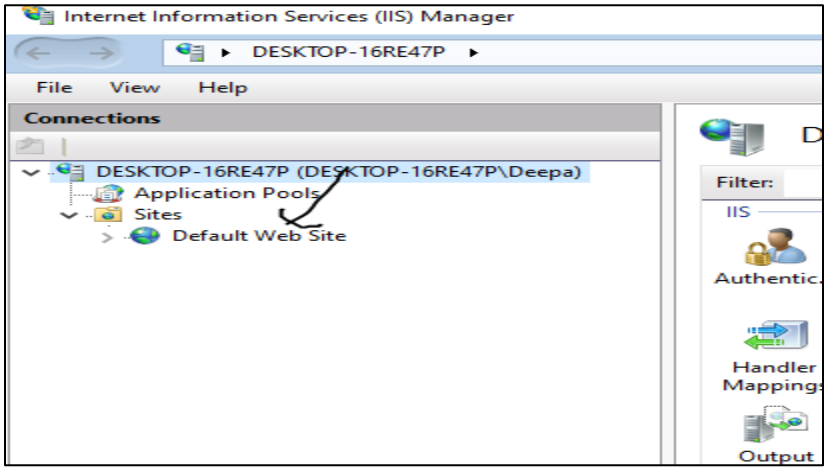
- FormsAuthentication.RedirectFromLoginPage(TextBox1.Text, true): On successful login to redirect on the page this method is used. we will redirect to the page that we have specified in default url.
- It also accepts 2 parameter username and true/false
- True indicates a cookie will be created for a longer period [persistent cookie]
- False indicates a cookie will be active till the browser is open [non-persistent cookie]
- Once our data is stored in the cookie we need to login again.

Windows Authentication:

Add IIS MANAGER

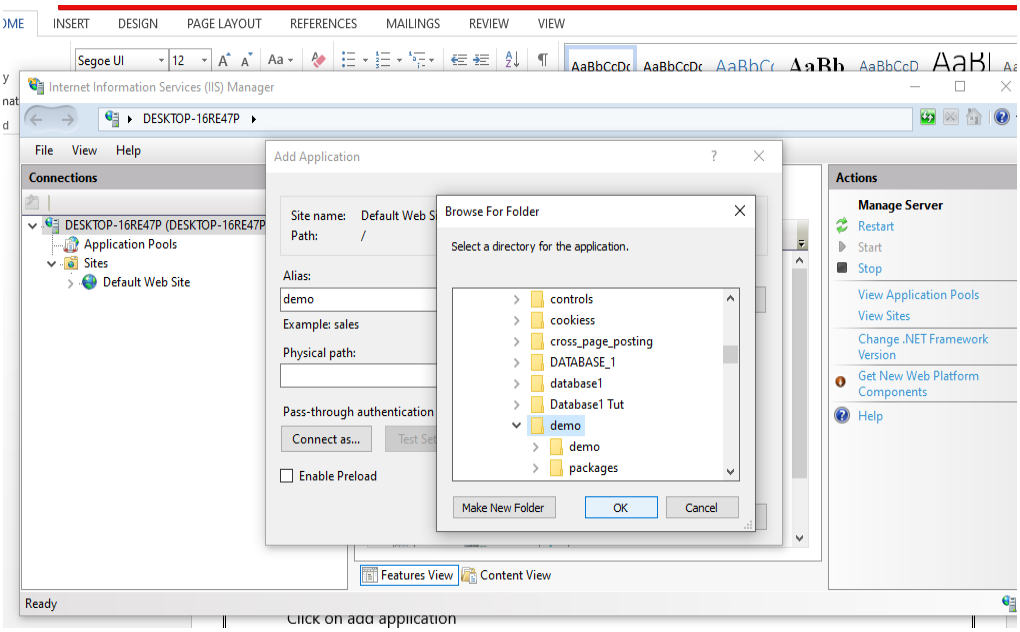
1. Click on windows features on/off
2. Select internet information service
3. Select world wide web services
4. Click on security and select windows authentication

Open iis manager



Right click on default web site

Click on add application
->select the application on which you want to apply windows authentication
->set an alias name



Click on add
After adding double click on demo , click on authentication

- ➔ Disable anonymous
- ➔ Enable windows

Now on web.config write following code

```
<system.web>
        <authentication mode="Windows"></authentication>
        <authorization>

              <allow users="*" />
              <deny users="?" />

        </authorization>
```

AJAX :

- With AJAX, web applications can retrieve data from the server asynchronously, in the background, without reloading the entire browser page. The use of AJAX has led to an increase in interactive animation on web pages.
  - AJAX stands for Asynchronous JavaScript and XML. This is a cross platform technology which speeds up response time. The AJAX server controls add script to the page which is executed and processed by the browser.
  - However like other ASP.NET server controls, these AJAX server controls also can have methods and event handlers associated with them, which are processed on the server side.
  - The control toolbox in the Visual Studio IDE contains a group of controls called the 'AJAX Extensions'

The ScriptManager Control

The ScriptManager control is the most important control and must be present on the page for other controls to work.

It has the basic syntax:

```
<asp:ScriptManager ID="ScriptManager1" runat="server">
</asp:ScriptManager>
```

If you create an 'Ajax Enabled site' or add an 'AJAX Web Form' from the 'Add Item' dialog box, the web form automatically contains the script manager control. The ScriptManager control takes care of the client-side script for all the server side controls.

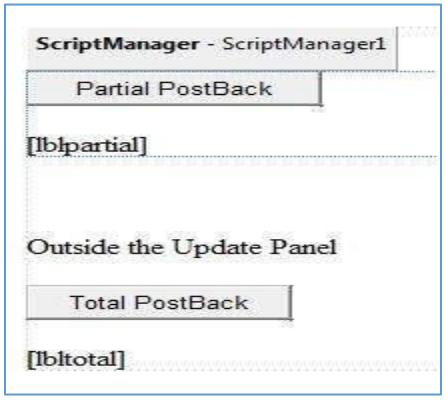
The UpdatePanel Control

- The UpdatePanel control is a container control and derives from the Control class. It acts as a container for the child controls within it and does not have its own interface. When a control inside it triggers a post back, the UpdatePanel intervenes to initiate the post asynchronously and update just that portion of the page.
- For example, if a button control is inside the update panel and it is clicked, only the controls within the update panel will be affected, the controls on the other parts of the page will not be affected. This is called the partial post back or the asynchronous post back.

Example

Add an AJAX web form in your application. It contains the script manager control by default. Insert an update panel. Place a button control along with a label control within the update panel control. Place another set of button and label outside the panel.

The design view looks as follows:



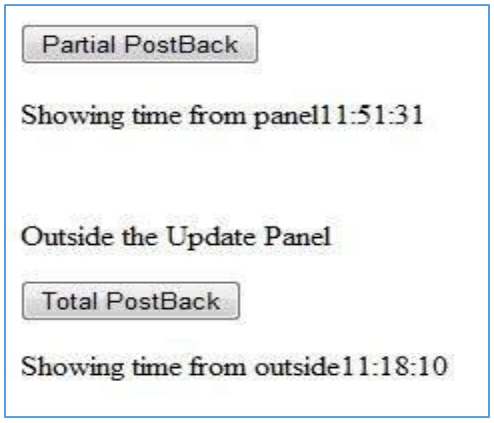

```
<form id="form1" runat="server">
  <div>
    <asp:ScriptManager ID="ScriptManager1" runat="server" />
  </div>

  <asp:UpdatePanel ID="UpdatePanel1" runat="server">
```

```
    <ContentTemplate>
      <asp:Button ID="btnpartial" runat="server" onclick="btnpartial_Click"
Text="Partial PostBack"/>
      <br />
      <br />
      <asp:Label ID="lblpartial" runat="server"></asp:Label>
    </ContentTemplate>
  </asp:UpdatePanel>

  <p> </p>
  <p>Outside the Update Panel</p>
  <p>
    <asp:Button ID="btntotal" runat="server" onclick="btntotal_Click" Text="Total
PostBack" />
  </p>
  <asp:Label ID="lbltotal" runat="server"></asp:Label>
</form>
```

Both the button controls have same code for the event handler:

```
string time = DateTime.Now.ToLongTimeString();
lblpartial.Text = "Showing time from panel" + time;
lbltotal.Text = "Showing time from outside" + time;
```

Observe that when the page is executed, if the total post back button is clicked, it updates time in both the labels but if the partial post back button is clicked, it only updates the label within the update panel.



A page can contain multiple update panels with each panel containing other controls like a grid and displaying different part of data.

When a total post back occurs, the update panel content is updated by default. This default mode could be changed by changing the UpdateMode property of the control. Let us look at other properties of the update panel.

The UpdateProgress Control

The UpdateProgress control provides a sort of feedback on the browser while one or more update panel controls are being updated. For example, while a user logs in or waits for server response while performing some database oriented job.

It provides a visual acknowledgement like "Loading page...", indicating the work is in progress.

The syntax for the UpdateProgress control is:

```
<asp:UpdateProgress ID="UpdateProgress1" runat="server"
DynamicLayout="true" AssociatedUpdatePanelID="UpdatePanel1" >

  <ProgressTemplate>
    Loading...
  </ProgressTemplate>

</asp:UpdateProgress>
```

The above snippet shows a simple message within the ProgressTemplate tag. However, it could be an image or other relevant controls. The UpdateProgress control displays for every asynchronous postback unless it is assigned to a single update panel using the AssociatedUpdatePanelID property.

The Timer Control

The timer control is used to initiate the post back automatically. This could be done in two ways:

(1) Setting the Triggers property of the UpdatePanel control:

```
<Triggers>
  <asp:AsyncPostBackTrigger ControlID="btnpanel2" EventName="Click" />
</Triggers>
```

(2) Placing a timer control directly inside the UpdatePanel to act as a child control trigger. A single timer can be the trigger for multiple UpdatePanels.

```
<asp:UpdatePanel ID="UpdatePanel1" runat="server"
UpdateMode="Always">

  <ContentTemplate>
    <asp:Timer ID="Timer1" runat="server" Interval="1000">
      </asp:Timer>

    <asp:Label ID="Label1" runat="server" Height="101px"
style="width:304px" >
      </asp:Label>
  </ContentTemplate>

</asp:UpdatePanel>
```

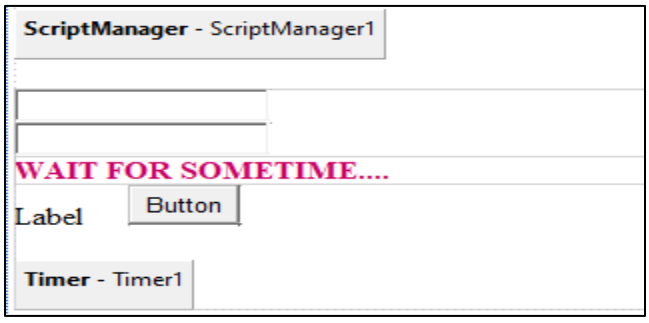
Timer:

```
Label1.Text=DateTime.Now.ToString("hh:mm:ss");
```

UPDATEPROGESS:

```
<body>
  <form id="form1" runat="server">
    <div>

      <asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>

    </div>
    <p>
      <asp:Label ID="Label3" runat="server" Text="Label"></asp:Label>
      <asp:ScriptManager ID="ScriptManager1" runat="server">
```

```
        </asp:ScriptManager>
    </p>
    <asp:UpdatePanel ID="UpdatePanel1" runat="server">

        <ContentTemplate>
            <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
            <br />
            <asp:TextBox ID="TextBox2" runat="server"></asp:TextBox>
            <br />
             <asp:UpdateProgress ID="UpdateProgress1" runat="server">
                <ProgressTemplate>
                    <asp:Label ID="Label4" runat="server" Text="WAIT FOR SOMETIME...." Font-
Bold="True" ForeColor="#CC0066"></asp:Label>
                </ProgressTemplate>
            </asp:UpdateProgress>
            <asp:Label ID="Label2" runat="server" Text="Label"></asp:Label>
<asp:Button ID="Button1" runat="server" CssClass="auto-style1" OnClick="Button1_Click1"
Text="Button" />
            <br />
              <br />
            <asp:Timer ID="Timer1" runat="server" Interval="1000"
OnTick="Timer1_Tick"></asp:Timer>
        </ContentTemplate>
    </asp:UpdatePanel>
  </form>
</body>
</html>
```




```
protected void Button1_Click1(object sender, EventArgs e)
    {
        System.Threading.Thread.Sleep(2000);
        Label2.Text = TextBox2.Text + TextBox1.Text;
```

```
    }
```

We can also image or gif in progess bar

```
<asp:UpdateProgress ID="UpdateProgress1" runat="server">
                    <ProgressTemplate>

asp:Image ID="Image1" runat="server" Height="55px"
ImageUrl="~/Youtube_loading_symbol_1_(wobbly).gif" Width="193px" />

                    </ProgressTemplate>
                </asp:UpdateProgress>
```