UNIT-III

Error

- An error is something that you have done that is considered to be wrong or incorrect, or that should not be done.
- Developers can make mistake whenever they try to create anything useful. As a result, We need some method and tools for managing error. Fortunately asp.net provides several way of finding, logging or trying to fix error when they occur.

What are the type of Errors

In .NET, there are primarily three types of errors:

1. Compilation Errors

- Compilation errors occur during the compilation phase of the code, indicating that the code violates the rules of the programming language or contains syntax errors.
- These errors prevent the code from being successfully compiled into an executable or assembly.
- Common compilation errors include missing or misplaced parentheses, semicolons, or incorrect variable declarations. Compilation errors must be fixed before the code can be executed.

2. Runtime Errors

- Runtime errors, also known as exceptions, occur during the execution of a program. They are typically caused by unexpected or exceptional conditions that the program encounters during runtime.
- These errors can be due to various factors such as invalid input, division by zero, accessing null objects, or attempting to perform an operation that is not supported.
- When a runtime error occurs and is not properly handled, it can lead to program termination or undesired behavior.
- To handle runtime errors, exception handling mechanisms are used, such as try-catch blocks, to gracefully recover from exceptional conditions and prevent the application from crashing.

3. Logical Errors

- Logical errors, also known as bugs, occur when the program's logic or algorithm is incorrect, resulting in unintended or incorrect behavior.
- These errors do not cause the program to crash or generate exceptions, but they lead to incorrect outputs or unexpected results.
- Logical errors can be challenging to identify and fix since they require careful examination of the code's logic and understanding of the expected behavior. Debugging techniques like stepping through the code, inspecting variables, and using logging can help in identifying and resolving logical errors.

Exception:

- An execution is an unexpected or unwanted event, which occurs during program execution i.e. during execution or runtime, which interrupts the normal flow of program instructions.
- At some point during the process, the user may face the possibility that the system may crash during the execution.This unwanted event is known as exception.

Exception Handling:

- Exception Handling is *a process to handle runtime errors*. We perform exception handling so that normal flow of the application can be maintained even after runtime errors.
- Exception is an event or object which is thrown at runtime. All exceptions the derived from *System.Exception* class.
- It is a runtime error which can be handled. If we don't handle the exception, it prints exception message and terminates the program.

C# Exception Classes

- All the exception classes in C# are derived from System.Exception class. Let's see the list of C# common exception classes.

Exception	Description
System.DivideByZeroException	handles the error generated by dividing a number with zero.
System.NullReferenceException	handles the error generated by referencing the null object.

System.InvalidCastException	handles the error generated by invalid typecasting.
System.IO.IOException	handles the Input Output errors.
System.FieldAccessException	handles the error generated by invalid private or protected field access.

Exception Handling Keywords

In C#, we use 4 keywords to perform exception handling:

- try
- catch
- finally, and
- throw

- **try** − A try block identifies a block of code for which particular exceptions is activated. It is followed by one or more catch blocks.
- **catch** − A program catches an exception with an exception handler at the place in a program where you want to handle the problem. The catch keyword indicates the catching of an exception.
- **finally** − The finally block is used to execute a given set of statements, whether an exception is thrown or not thrown. For example, if you open a file, it must be closed whether an exception is raised or not.
- **throw** − A program throws an exception when a problem shows up. This is done using a throw keyword.

Syntax

Assuming a block raises an exception, a method catches an exception using a combination of the try and catch keywords. A try/catch block is placed around the code that might generate an exception. Code within a try/catch block is referred to as protected code, and the syntax for using try/catch looks like the following −

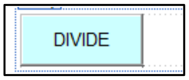
```
Try
 {
// statements causing exception
} catch( ExceptionName e1 )
{
// error handling code
} catch( ExceptionName e2 )
```

```
 {
// error handling code
} catch( ExceptionName eN )
{
  // error handling code
}
finally
 {
  // statements to be executed
}
```

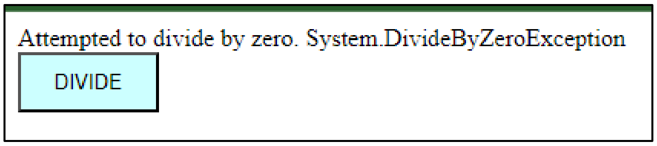
Handling Exceptions

- C# provides a structured solution to the exception handling in the form of try and catch blocks. Using these blocks the core program statements are separated from the error-handling statements.
- These error handling blocks are implemented using the **try**, **catch**, and **finally** keywords. Following is an example of throwing an exception when dividing by zero condition occurs −

```
protected void Button1_Click(object sender, EventArgs e)
{
   int A = 10, B = 0, C;
   try
   {
      C = A / B;
      Response.Write(C);
   }
   catch (Exception e1)
   {
      Response.Write(e1.Message + " " + e1.GetType());
   }
}
```



Output:

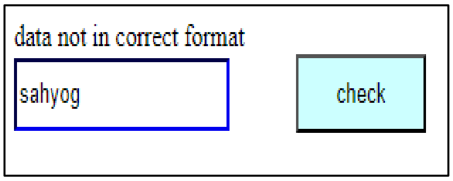


```
Attempted to divide by zero. System.DivideByZeroException
```

Multiple Catch Block

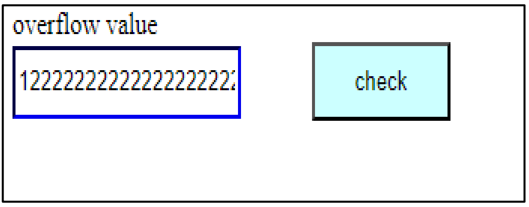
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

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

        }
        protected void Button1_Click(object sender, EventArgs e)
        {
            try
            {
                int a =Convert.ToInt32(TextBox1.Text);
                Response.Write(a);
            }
            catch(IndexOutOfRangeException r)
            {
                Response.Write("array index not in range");
            }
            catch(OverflowException o)
            {
                Response.Write("overflow value");
            }
            catch(FormatException a)
            {
                Response.Write("data not in correct format");
            }
            catch(Exception e1)
            {
                Response.Write(e1.Message);
            }

        }
    }
}
```

Output:



data not in correct format

sahyog



overflow value

1222222222222222222

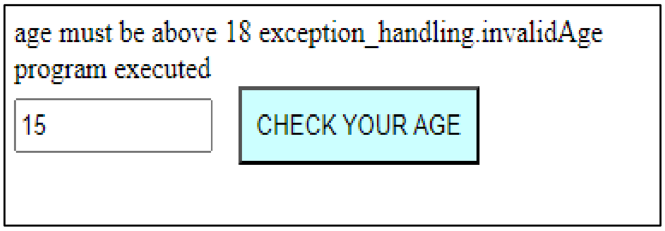
Throwing Your Own Exceptions/User Defined Exception

C# allows us to create user-defined or custom exception. It is used to make the meaningful exception. To do this, we need to inherit Exception class. –

Program:

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

namespace exception_handling
{
    public class invalidAge:Exception
    {
        public invalidAge(string m):base(m)
        {

        }
    }
    public partial class WebForm2 : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {

        }

        protected void Button1_Click(object sender, EventArgs e)
        {
            int a;
            try
            {
                 a = Convert.ToInt32(TextBox1.Text);
                if (a < 18)
                    throw new invalidAge("age must be above 18");
                else
                    Response.Write("valid age");
            }

            catch (Exception e1)
            {
                Response.Write(e1.Message + " " + e1.GetType());
            }
            finally
            {
                Response.Write("<br> program executed");
            }
        }
    }
}
```

Output:



age must be above 18 exception_handling.invalidAge
program executed

State Management:

State Management can be defined as the technique or the way by which we can maintain / store the state of the page or application until the User's Session ends.

What is the need of State Management?

- Let us assume that someone is trying to access a banking website and he has to fill in a form.
- So the person fills in the form and submits it. After submission of the form, the person realizes he has made a mistake. So he goes back to the form page and he sees that the information he submitted is lost. So he again fills in the entire form and submits it again. This is quite annoying for any user. So to avoid such problems "STATE MANAGEMENT" acts as a savior.

ASP.Net State Management Techniques

ASP.NET provides us with 2 ways to manage the state of an application. It is basically divided into the 2 categories:

1. Client Side State Management.
2. Server Side State Management.

1. Client Side State Management

- It is a way in which the information which is being added by the user or the information about the interaction happened between the user and the server is stored on the client's machine or in the page itself.
- The server resources (e.g. server's memory) is not at all utilized during the process.

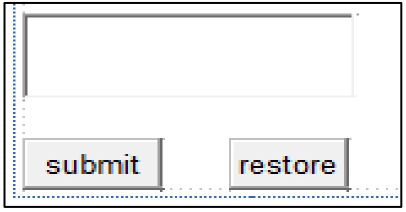
This management technique basically makes use of the following:

a. View State
b. Cookies
c. Hidden Fields
d. Query String

a. View State

- View State can be used to maintain the State at a page level.
- The term "Page Level" means that the information is being stored for a specific page and until that specific page is active (i.e. the page which is being currently viewed by the user).
- Once the user is re-directed or goes to some other page, the information stored in the View State gets lost.
- Using a View State is quite simple. In fact, it is the default way for storing the page or the control's information.
- Typically the View State is used, when we want a user to be re-directed to the same page and the information being added by the user remains persistent until the user is on the same page.

Program:



```
protected void btn_submit_Click(object sender, EventArgs e)
{
          ViewState["a"] = TextBox1.Text;
          TextBox1.Text = " ";
}

protected void btn_reset_Click(object sender, EventArgs e)
{
         TextBox1.Text = ViewState["a"].ToString();
}
```

b. COOKIES:

- A *cookie* is a small text file used to identify users uniquely. It is located on the client's computer; it is not stored in the server's memory.
- You can say a cookie is a small file generated and kept on the client's machine that holds information about the user. When a user requests a new page, the server generates a cookie and delivers it to the client, along with the requested page. The client then receives and keeps that cookie either permanently, or temporarily, in the browser.

There are two types of Cookies:

- Persistence Cookies

- Non-Persistence Cookies

Persistence Cookies

Cookies that have a specific expiry date and time are called *Persistence cookies*. They maintain the data on the user’s machine (user’s hard disk folder), such as sign-on information or settings. They expire based on a date assigned by the webserver. Typically, these are seen as permanent cookies that do not expire.

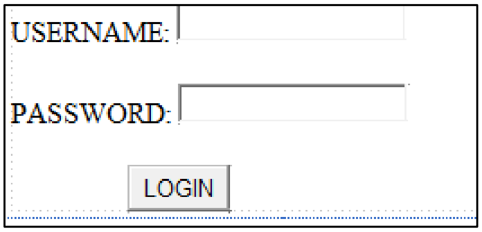
Non-Persistence Cookie

Non-Persistence cookies are not stored permanently on the user’s machine. They maintain information as long as the user accesses the same browser. Once the browser is closed, the cookie gets discarded.

To create a non-persistence cookie, simply do not add an expiration date.

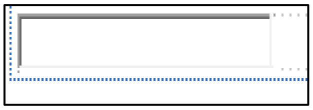
Program to create a cookie and access its value in next page.

FIRST.ASPX:



```
HttpCookie hp = new HttpCookie("Login Details");
        hp["username"] = TextBox1.Text;
        hp["password"] = TextBox2.Text;

        Response.Cookies.Add(hp);
        hp.Expires = DateTime.Now.AddSeconds(100);
        Response.Write("cookies added");
        Response.Redirect("second.aspx");
```

SECOND.ASPX:



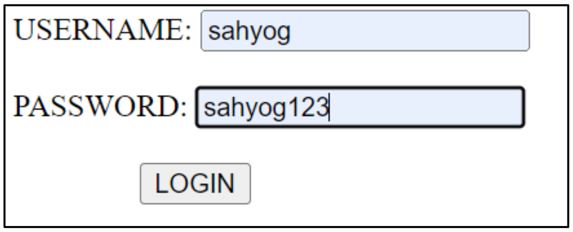
```
protected void Page_Load(object sender, EventArgs e)
{
        HttpCookie c = Request.Cookies["Login Details"];
        TextBox1.Text = c["username"].ToString();
 }
```

- Login details is name of cookie.

- Response.Cookies.Add(hp); -> it adds the cookie
- hp.Expires = DateTime.Now.AddSeconds(100); -> its sets expiry time of cookie.
- Request.cookies is used is used the fetch the cookie value from browser.

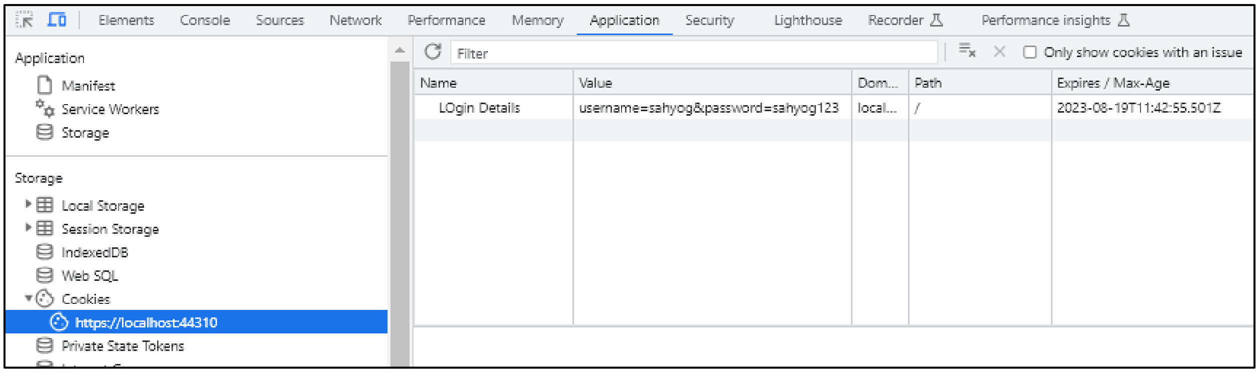
This is how cookies are stored in browser

Rightclick on browser->inspect->application->cookies->localhost



USERNAME: sahyog

PASSWORD: sahyog123

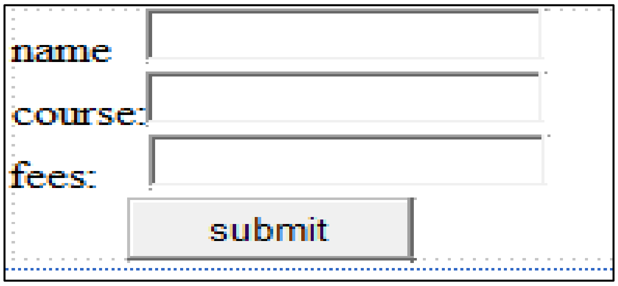


Name	Value	Dom...	Path	Expires / Max-Age
LOgin Details	username=sahyog&password=sahyog123	local...	/	2023-08-19T11:42:55.501Z

c. **Query String**

- In query string data is passed in url in key value pair.
- Query string is mostly used to fetch data from one page to another
- And while redirecting to another page we pass values.
- You can use **QueryString** to store values in the URL. The value of the query string can be seen in the URL and is visible to all users.
- Request.QueryString() is used to fetch the query string value.

First.aspx



```
protected void Button1_Click(object sender, EventArgs e)
{
   Response.Redirect("next.aspx?name=" + TextBox1.Text + "&course="   + TextBox2.Text +
"&fees=" + TextBox3.Text);
}
```

Next.aspx

```
protected void Page_Load(object sender, EventArgs e)
{
         string n = Request.QueryString["name"];
         TextBox1.Text = n;
         TextBox2.Text = Request.QueryString["course"];
 }
```

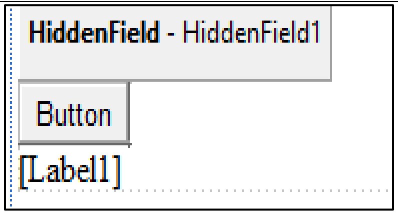
d. HiddenField:

- HiddenField, as name implies, is hidden. This is non visual control in ASP.NET where you can save the value. This is one of the types of client-side state management tools. It stores the value between the roundtrip. Anyone can see HiddenField details by simply viewing the source of document.
- HiddenFields are not encrypted or protected and can be changed by anyone. However, from a security point of view, this is not suggested. ASP.NET uses HiddenField control for managing the ViewState. So, don’t store any important or confidential data like password and credit card details with this control.

```
<asp:HiddenFieldID="HiddenField1" runat="server" />
```

Example:

```
<form id="form1" runat="server">
<div>
<asp:HiddenField ID="HiddenField1" runat="server" Value="0" />
<asp:Button ID="Button1" runat="server" Text="Button" OnClick="Button1_Click" /><br/>
 <asp:Label ID="Label1" runat="server" Text=""></asp:Label>
</div>
</form>
```



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

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

        }

        protected void Button1_Click(object sender, EventArgs e)
        {
            HiddenField1.Value =(Convert.ToInt32(HiddenField1.Value )+ 1).ToString();
            Label1.Text = HiddenField1.Value;
        }
    }
}
```

2. Server Side State Management

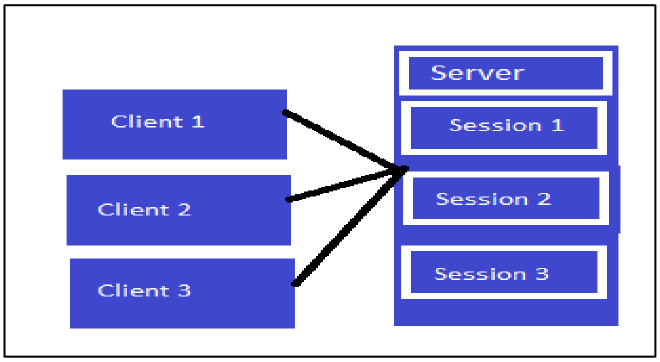
It is another way which ASP.NET provides to store the user's specific information or the state of the application on the server machine. It completely makes use of server resources (the server's memory) to store information.

This management technique basically makes use of the following,

a. Session State
b. Application State

a. Session

- Session is a State Management Technique. A Session can store the value on the Server.
- Session management is a powerful technique used for preserving data over sessions.
- **Session** is used to store user information and to uniquely identify a user (or a browser).
- ASP.NET uses a **Session ID**, which is generated by the server, to keep track of the status of the current user's information.
- When a new user submits a request to the server, ASP.NET automatically generates a **Session ID** and that **Session ID** is transmitted with every request and response made by that particular user.
- A session is one of the best techniques for State Management because it stores the data as client-based, in other words, the data is stored for every user separately and the data is secured also because it is on the server.



Program to understand session.

In this program 4 pages are created login, enroll, third and logout page. To access enroll and third page, user has to login otherwise he wont be able to access that page, so to track whether user is logged in or not we will use session.

LOGIN PAGE

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

        }

        protected void Button1_Click(object sender, EventArgs e)
        {

            if(TextBox1.Text=="sahyog" && TextBox2.Text=="sahyog123")
            {
                Session["user"] = TextBox1.Text;
                Session["pswd"] = TextBox2.Text;
                Response.Redirect("enroll.aspx");
            }
            else
            {
                TextBox1.Focus();
            }
        }
    }
}
```

ENROLL PAGE

```
namespace sessions
{
    public partial class enroll : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            if(Session["user"] ==null)
            {
                Response.Redirect("Login.aspx");
            }
            else
            {
                Label1.Text = "HELLO " + Session["user"];
            }
        }
        protected void Button1_Click(object sender, EventArgs e)
        {
            Session["course"] = DropDownList1.SelectedItem;
            Response.Redirect("third.aspx");
        }
    }
}
```

THIRD.ASPX

```
protected void PREV_Click(object sender, EventArgs e)
        {
            if(Session["user"]==null || Session["pswd"]==null)
            {
                Response.Redirect("LOGIN.ASPX");
            }
            else
            {
              Response.Redirect("enroll.ASPX");
            }
        }
        protected void LOGOUT_Click(object sender, EventArgs e)
        {
            Response.Redirect("logout.aspx");
        }
```

LOGOUT.ASPX

```
namespace sessions
{
    public partial class logout : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
             if(Session["user"]!=null)
            {
                Session.Abandon();
                Response.Redirect("login.aspx");
            }

        }
    }
}
```

Setting Session Timeout:

- By default asp.net web application manages session for 20 minutes and after 20 minutes session gets destroyed.

- But if user wants to manually set the session timeout,then following line must be written in web.config file.

```
<system.web>
    <sessionState timeout="1"> </sessionState>
</Sytem.web>
```

- **Here 1 means 1 minute**

b. Application State

- ASP.Net Application state is a server side state management technique.
- Application state is a global storage mechanism that used to stored data on the server and shared for all users, means data stored in Application state is common for all user. Data from Application state can be accessible anywhere in the application. Application state is based on the System.Web.HttpApplicationState class.
- The application state used same way as session state, but session state is specific for a single user session, where as application state common for all users of asp.net application.

- **Syntax of Application State**

Store information in application state

Application[“name”] = “sahyog college”;

Retrieve information from application state

string str = Application[“name”].ToString();

Example of Application State in ASP.Net

- Generally we use application state for calculate how many times a given page has been visited by various clients.
- Run following code in different browsers to get the output

Set Following code in global.asax file

Global.asax

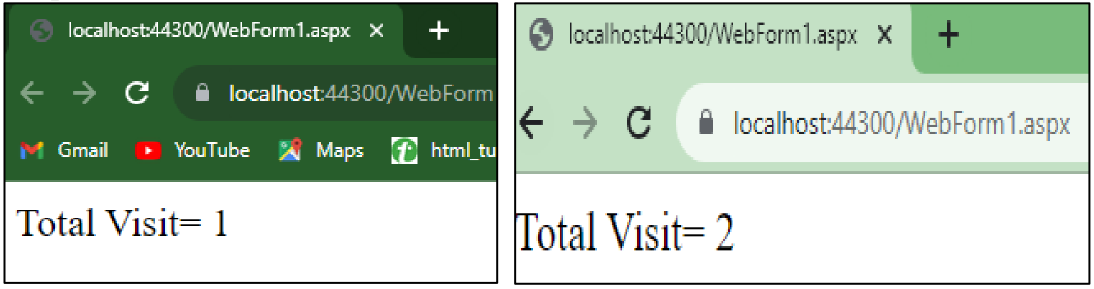
```
protected void Session_Start(object sender, EventArgs e)
 {
    if(Application["c"] != null)
    {
       Application["c"] =  Convert.ToInt32(Application["c"].ToString()) + 1;
    }
    else
    {
       Application["c"] = 1;
```

```
          }
        }
```

Aspx File:

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

namespace applicationss
{
    public partial class WebForm1 : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            Label1.Text = “Total Visit = ”  + Application["c"].ToString();
        }
    }
}
```

Output:


localhost:44300/WebForm1.aspx
localhost:44300/WebForm
Gmail
YouTube
Maps
html_tu
localhost:44300/WebForm1.aspx
localhost:44300/WebForm1.aspx

```
Total Visit= 1
```

```
Total Visit= 2
```

Styles:

- The styles are actually the part of the cascading stylesheet (CSS).
- CSS is simply the code intended to simplify the process of making web page attractive.
- CSS manages the look and feel part of a web page. The CSS can be used to control the styles of font, color of text, the spacing etc.
- CSS is quite simple to learn and understand but still it is a powerful controlling the presentation of an HTML document.

Types of CSS:

1. INTERNAL CSS
2. INLINE CSS
3. EXTERNAL CSS

1. INTERNAL CSS

- Internal CSS is one of the most widely used CSS forms for changing, styling, and modifying the unique styles of a single web page. You can use the internal CSS by integrating the <style> element in the <head> section of a HTML web page.
- Internal CSS can be applied to the whole web page but not on multiple web pages and you can style several web pages by using the same code on every page.

```
<head>
<style>
body {background-color: aliceblue;}
h1   {color: yellow;}
p    {color: black;}
</style>
</head>
```

2. INLINE CSS

- Inline CSS is the technique to define the single element with the insert style sheets in an HTML document. We can add CSS in three approaches: Inline, Internal, and External.
- It has the interactive and unique style to create a single HTML element; we can define the inline CSS on the style attribute.

```
<html>
<body>
<h1 style="color:blue;text-align:center;">This is a placeholder of heading</h1>
<p style="color:red;">This is a placeholder for a paragraph.</p>
</body>
</html>
```

3. EXTERNAL CSS

- To apply a rule to multiple pages, an external style sheet is used. An external style sheet is a separate CSS file that can be accessed by creating a link within the head section of the webpage. Multiple webpages can use the same link to access the stylesheet.
- The link to an external style sheet is placed within the head section of the page.

```
<head>
<link rel="stylesheet"  href="mystyle.css">
</head>
```

- From solution explorer add “stylesheet.css” file.

- The actual style sheet file will contain CSS rules that are then applied across the entire page.
- **For example:**

  ```
  Body
   {
  background-color: ghostwhite;
  }
  h1
  {
  color: blue; font-size: 20px; font-family: verdana; font-style:italic;
  }
  ```

Types of Selectors

1. Universal Selector

- The universal selector is denoted by * and it applies to all element in the web page
- It can be used to set global settings like a font family.

```
* {
    background-color: yellow;
}
```

2. Element Selector

- The element selector selects HTML elements based on the element name.
- It is used to apply formatting to specific element or tag.
- Here, all <p> elements on the page will be center-aligned, with a red text color:

```
p {
  text-align: center;
  color: red;
}
```

3. Id Selector

- The id selector uses the id attribute of an HTML element to select a specific element.
- The id of an element is unique within a page, so the id selector is used to select one unique element!
- To select an element with a specific id, write a hash (#) character, followed by the id of the element.

Example

The CSS rule below will be applied to the element with id="para1":

```
#para1
{
  text-align: center;
  color: red;
}
```

4. Class selector

- The class selector selects elements with a specific class attribute.
- To select elements with a specific class, write a period (.) character, followed by the class name.

Example

In this example all HTML elements with class="center" will be red and center-aligned:

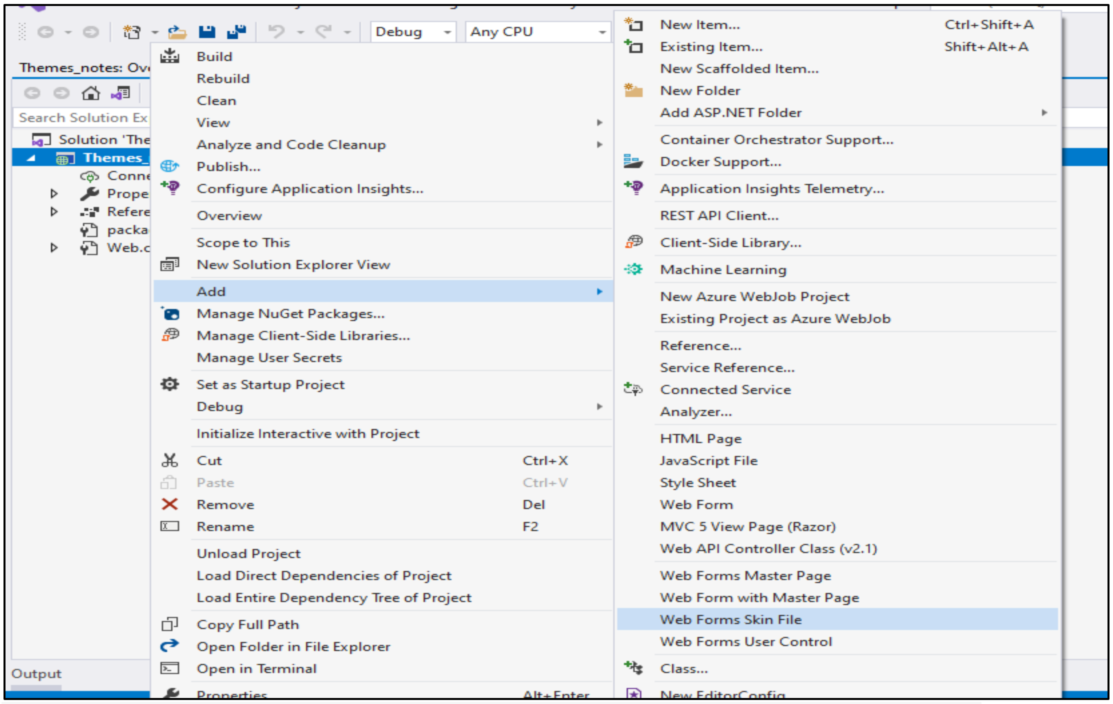
```
.center {
  text-align: center;
  color: red;
}
```

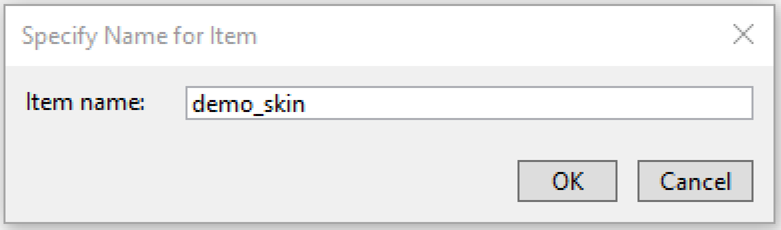
Theme

- A theme is a collection of CSS property settings that allow you to define the look of the pages and controls, and apply the same look across pages in a web application or across all web applications on a server.
- Themes are the way to define the formatting for various controls and can be reused in multiple pages. Later, by applying minor changes on the themes, the complete appearance of website can be changed.
- Themes are made up of a set of elements: skins, cascading style sheets (CSS), images, and other resources. At a minimum, a theme will contain skins.

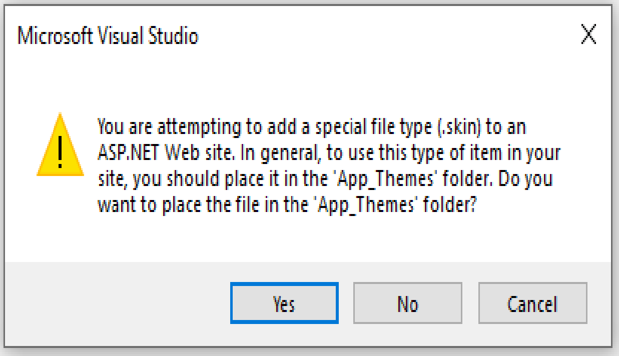
Steps to create Themes or Dynamic Style:

- Create a App_Themes Folder.
- Create the Theme sub folders.

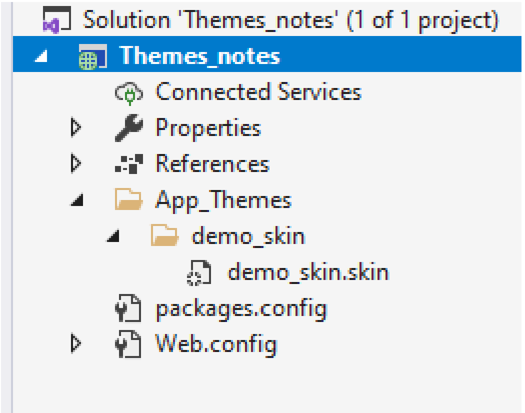




Item name: demo_skin



You are attempting to add a special file type (.skin) to an ASP.NET Web site. In general, to use this type of item in your site, you should place it in the 'App_Themes' folder. Do you want to place the file in the 'App_Themes' folder?



Skins

- A skin file has the file name extension .skin and contains property settings for individual controls such as Button, Label, TextBox, or Calendar controls.
- Control skin settings are like the control markup itself, but contain only the properties you want to set as part of the theme. For example, the following is a control skin for a Button control:

```
<asp:button runat=”server” BackColor=”lightblue” ForeColor=”black” />
```

- You create .skin files in the Theme folder. A .skin file can contain one or more control skins for one or more control types. You can define skins in a separate file for each control or define all the skins for a theme in a single file.

There are two types of control skins,

1. default skins:

- A default skin automatically applies to all controls of the same type when a theme is applied to a page.
- A control skin is a default skin if it does not have a SkinID attribute. For example, if you create a default skin for a Calendar control, the control skin applies to all Calendar controls on pages that use the theme. (Default skins are matched exactly by control type, so that a Button control skin applies to all Button controls, but not to LinkButton controls or to controls that derive from the Button object.)

```
• <asp:Button runat="server" Font-Bold="true" ForeColor="#0000FF" />
• <asp:Label runat="server" Font-Size="Large" ForeColor="#FF0000" />
```

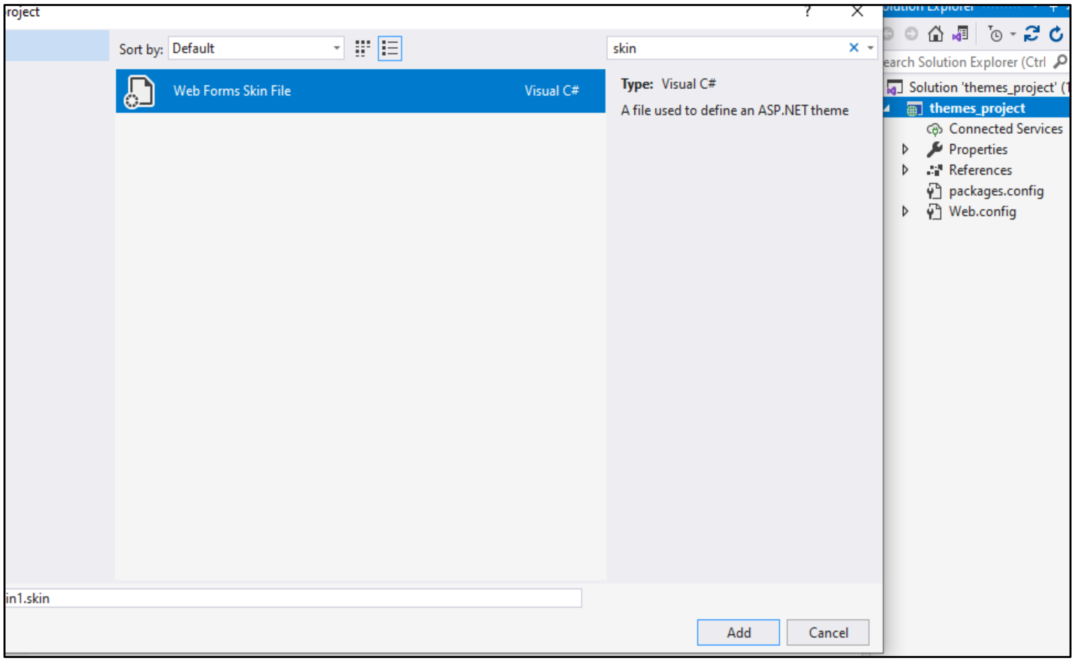
2. named skins:

- A named skin is a control skin with a SkinID property set. Named skins do not automatically apply to controls by type. Instead, you explicitly apply a named skin to a control by setting the control’s SkinID property. Creating named skins allows you to set different skins for different instances of the same control in an application.

```
• <asp:Button runat="server" SkinId=”one” Font-Bold="true" ForeColor="#0000FF" />
•  <asp:Label runat="server” SkinId=”two”  Font-Size="Large" ForeColor="#FF0000" />
```

<u>Default Skin Example</u>

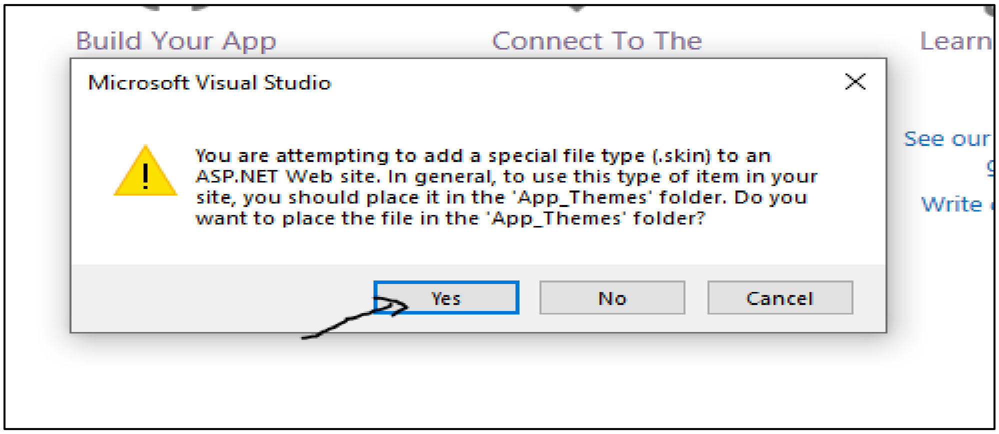
1. Add a skin a file and write following code



Type: Visual C#

A file used to define an ASP.NET theme

in1.skin



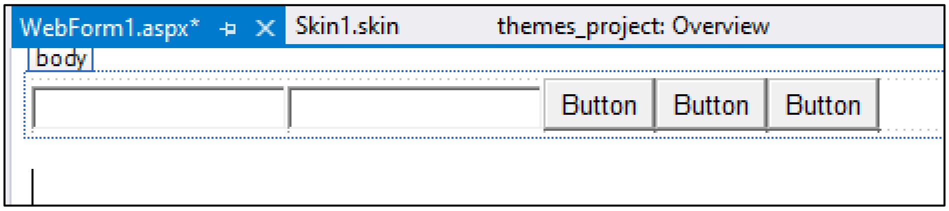
Microsoft Visual Studio

You are attempting to add a special file type (.skin) to an ASP.NET Web site. In general, to use this type of item in your site, you should place it in the 'App_Themes' folder. Do you want to place the file in the 'App_Themes' folder?

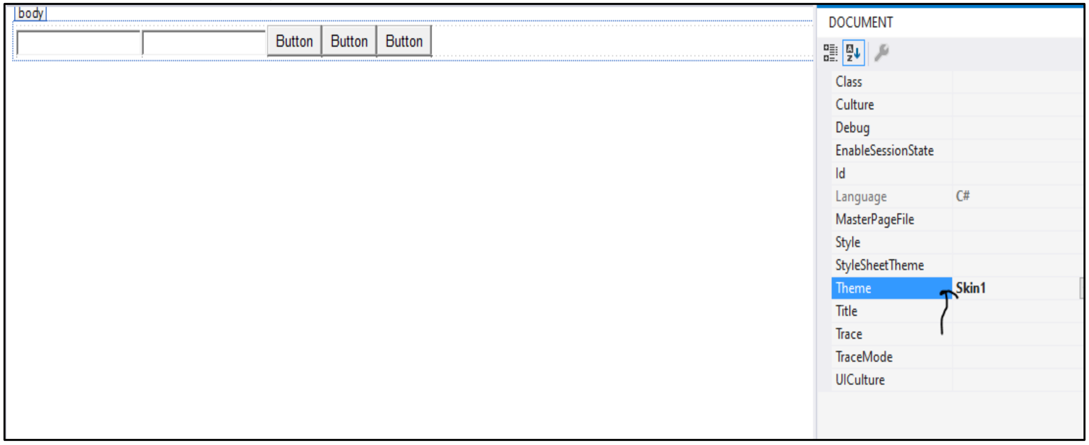
Skin1.skinFile

```
<asp:TextBox runat="server" BackColor="Red" ForeColor="Blue"
BorderColor="yellow"></asp:TextBox>

<asp:Button runat="server" BackColor="blue" ForeColor="green"
BorderColor="pink"></asp:Button>
```

Add a web Form:

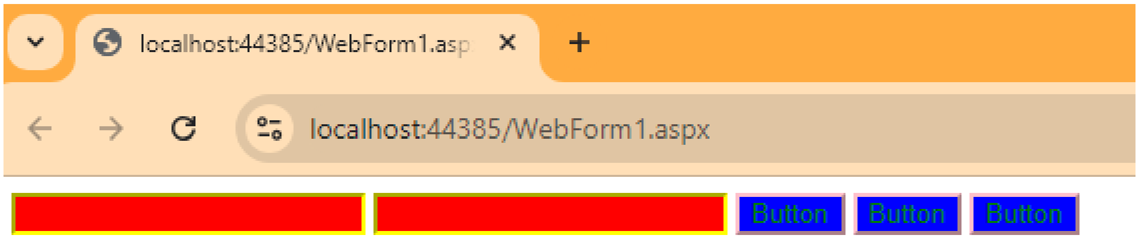


Set theme property of the web form



Class
Culture
Debug
EnableSessionState
Id
Language C#
MasterPageFile
Style
StyleSheetTheme
Theme Skin1
Title
Trace
TraceMode
UICulture

Output:



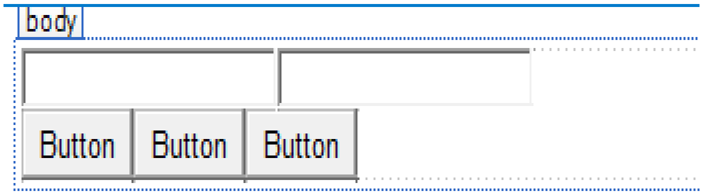
localhost:44385/WebForm1.aspx

Named Skin Example

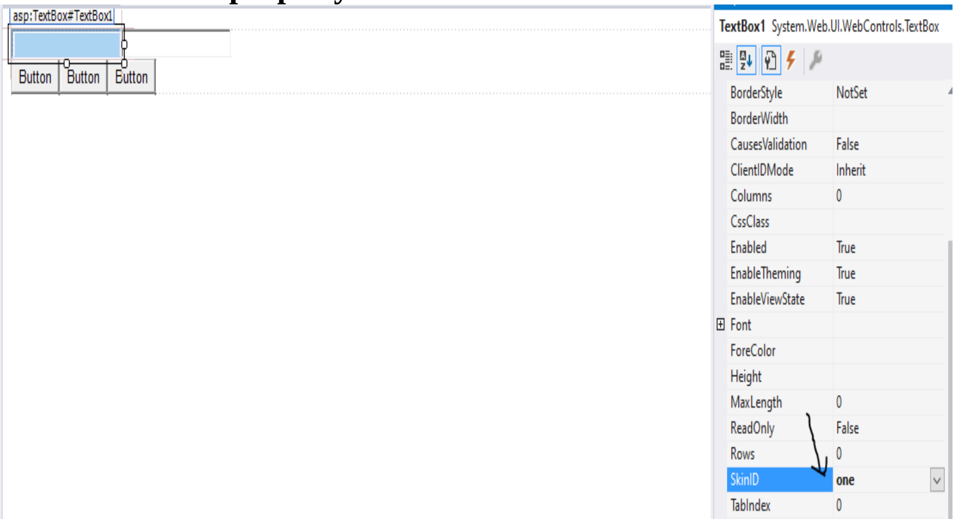
1. **Add a skin file a write following code**

```
<asp:TextBox SkinId="one" runat="server" BackColor="Red" ForeColor="Blue"
BorderColor="yellow"></asp:TextBox>
<asp:TextBox SkinId="two" runat="server" BackColor="blue" ForeColor="red"
BorderColor="yellow"></asp:TextBox>
<asp:Button SkinId="b1" runat="server" BackColor="yellow" ForeColor="red"
BorderColor="pink"></asp:Button>
<asp:Button SkinId="b2" runat="server" BackColor="black" ForeColor="white"
BorderColor="pink"></asp:Button>
<asp:Button SkinId="b3" runat="server" BackColor="blue" ForeColor="pink"
BorderColor="pink"></asp:Button>
```

2. **Add a web form and set its theme property**



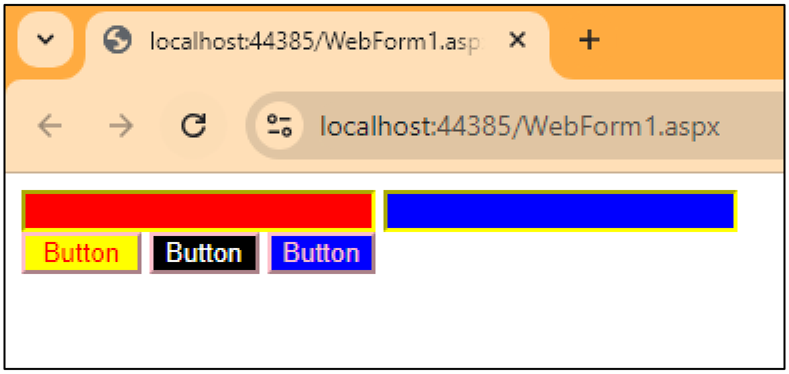
3. **Now Add skinid property of the controls.**



TextBox1 System.Web.UI.WebControls.TextBox

```
BorderStyle       NotSet
BorderWidth
CausesValidation  False
ClientIDMode      Inherit
Columns           0
CssClass
Enabled           True
EnableTheming     True
EnableViewState   True
Font
ForeColor
Height
MaxLength         0
ReadOnly          False
Rows              0
SkinID            one
TabIndex          0
```

Likewise add skinid property of all the controls



Master Page

****