TPCT's
College of Engineering, Osmanabad

Laboratory Manual

SOFTWARE DEVELOPMENT LAB

For

Third Year Students

Manual Prepared by

Mr. T.K.Takbhate

Author COE, Osmanabad

TPCT's

College of Engineering
Solapur Road, Osmanabad
Department of Computer Science and Engineering

Vision of the Department:

To develop computer engineers with necessary analytical ability and human values who can creatively design, implement a wide spectrum of computer systems for welfare of the society.

Mission of the Department:

1. Preparing graduates to work on multidisciplinary platforms associated with their professional position both independently and in a team environment.
2. Preparing graduates for higher education and research in computer science and engineering enabling them to develop systems for society development.

College of Engineering

Technical Document

This technical document is a series of Laboratory manuals of Computer Science and Engineering Department and is a certified document of College of Engineering, Osmanabad. The care has been taken to make the document error-free. But still if any error is found. Kindly bring it to the notice of subject teacher and HOD.

Recommended by,

HOD

Approved by,

Principal

Copies:

1. Departmental Library
2. Laboratory
3. HOD
4. Principal

FOREWORD

It is my great pleasure to present this laboratory manual for Third year engineering students for the subject of Programming in C# .NET
As a student, many of you may be wondering with some of the questions in your mind regarding the subject and exactly what has been tried is to answer through this manual. As you may be aware that MGM has already been awarded with ISO 9001:2000certification and it is our endure to technically equip our students taking the advantage of the procedural aspects of ISO 9001:2000 Certification.
Faculty members are also advised that covering these aspects in initial stage itself, will greatly relived them in future as much of the load will be taken care by the enthusiasm energies of the students once they are conceptually clear.

H.O.D.

CSE Dept.

LABORATORY MANUAL CONTENTS

This manual is intended for the Third year students of Computer Science and Engineering in the subject of Programming in C# .NET. This manual typically contains practical/Lab Sessions related Programming In C# .NET covering various aspects related the subject to enhanced understanding. Students are advised to thoroughly go through this manual rather than only topics mentioned in the syllabus as practical aspects are the key to understanding and conceptual visualization of theoretical aspects covered in the books.
Good Luck for your Enjoyable Laboratory Session

Mr. T.K.Takbhate

SUBJECT INDEX

Lab No.	Index	Week Involved
1	Design & develop ASP.net web application using validation controls.	1
2	Design & develop ASP.net web application use master & content page.	2
3	Design a web form to insert , update, delete & show student information through database connectivity with (SQl/Oracle).	3
4	Create a web service and use it in web site.	4
5	Design & develop ASP.net web application using session.	5
6	Call an external web service through sharepoint workflow.	6
7	Create custom action using sharepoint workflow.	7
8	Create a simple calculator using sharepoint workflow.	8
9	Create a web forms application that integrated with office 365.	9
10	Create a web form application for building resume	10
11	Mini Project	11
	Conduction of Viva-Voce Examinations	

Appendix - A

Appendix – B

1. Do’s and Don’ts in the laboratory

- Make entry in the Log Book as soon as you enter the Laboratory.
- All the students should sit according to their roll numbers starting from their left to right.
- All the students are supposed to enter the terminal number in the log book.
- Do not change the terminal on which you are working.
- All the students are expected to get at least the algorithm of the program/concept to be implemented.
- Strictly observe the instructions given by the teacher/Lab Instructor.

2. Pre-lab (Introduction to .NET Framework)

The Microsoft .NET Framework is a software framework that can be installed on computers running Microsoft Windows operating systems. It includes a large library of coded solutions to common programming problems and a virtual machine that manages the execution of programs written specifically for the framework.

The framework's Base Class Library provides a large range of features including user interface, data and data acces, database connectivity, cryptography, web application development, numeric algorithms, and network communications. The class library is used by programmers, who combine it with their own code to produce applications.Programs written for the .NET Framework execute in a software environment that manages the program's runtime requirements. Also part of the .NET Framework, this runtime environment is known as the Common Language Runtime (CLR). The CLR provides the appearance of an application virtual machine so that programmers need not consider the capabilities of the specific CPU that will execute the program. The CLR also provides other important services such as security, memory management, and exception handling. The class library and the CLR together constitute the .NET Framework.

3. Lab Experiments:

Experiment No:-1

Aim:- Design & develop ASP.net web application using validation controls.

Objective:-

To understand the concept of validation.
To understand the validation of input
Ablity to write the regular expression.

Theory:-

ASP.NET validation controls validate the user input data to ensure that useless, unauthenticated, or contradictory data don't get stored.

ASP.NET provides the following validation controls:

- RequiredFieldValidator
- RangeValidator
- CompareValidator
- RegularExpressionValidator
- CustomValidator
- ValidationSummary

BaseValidator Class

The validation control classes are inherited from the BaseValidator class hence they inherit its properties and methods. Therefore, it would help to take a look at the properties and the methods of this base class, which are common for all the validation controls:

Members	**Description**
ControlToValidate	Indicates the input control to validate.
Display	Indicates how the error message is shown.
EnableClientScript	Indicates whether client side validation will take.

Enabled	Enables or disables the validator.
ErrorMessage	Indicates error string.
Text	Error text to be shown if validation fails.
IsValid	Indicates whether the value of the control is valid.
SetFocusOnError	It indicates whether in case of an invalid control, the focus should switch to the related input control.
ValidationGroup	The logical group of multiple validators, where this control belongs.
Validate()	This method revalidates the control and updates the IsValid property.

RequiredFieldValidator Control

The RequiredFieldValidator control ensures that the required field is not empty. It is generally tied to a text box to force input into the text box.

The syntax of the control is as given:

```
<asp:RequiredFieldValidator ID="rfvcandidate"
runat="server" ControlToValidate ="ddlcandidate"
ErrorMessage="Please choose a candidate"
InitialValue="Please choose a candidate">


</asp:RequiredFieldValidator>
```

RangeValidator Control

The RangeValidator control verifies that the input value falls within a predetermined range.

It has three specific properties:

Properties	**Description**
Type	It defines the type of the data. The available values are: Currency, Date, Double, Integer, and String.
MinimumValue	It specifies the minimum value of the range.
MaximumValue	It specifies the maximum value of the range.

The syntax of the control is as given:

```
<asp:RangeValidator ID="rvclass" runat="server" ControlToValidate="txtclass"

ErrorMessage="Enter your class (6 - 12)" MaximumValue="12"

MinimumValue="6" Type="Integer">


</asp:RangeValidator>
```

CompareValidator Control

The CompareValidator control compares a value in one control with a fixed value or a value in another control.

It has the following specific properties:

Properties	**Description**
Type	It specifies the data type.
ControlToCompare	It specifies the value of the input control to compare with.
ValueToCompare	It specifies the constant value to compare with.
Operator	It specifies the comparison operator, the available values are: Equal, NotEqual, GreaterThan, GreaterThanEqual, LessThan, LessThanEqual, and DataTypeCheck.

The basic syntax of the control is as follows:

```
<asp:CompareValidator ID="CompareValidator1" runat="server"
ErrorMessage="CompareValidator">

</asp:CompareValidator>
```

RegularExpressionValidator

The RegularExpressionValidator allows validating the input text by matching against a pattern of a regular expression. The regular expression is set in the ValidationExpression property.

The following table summarizes the commonly used syntax constructs for regular expressions:

Character Escapes	**Description**
\b	Matches a backspace.
\t	Matches a tab.
\r	Matches a carriage return.
\v	Matches a vertical tab.
\f	Matches a form feed.
\n	Matches a new line.
\	Escape character.

Apart from single character match, a class of characters could be specified that can be matched, called the metacharacters.

Metacharacters	**Description**
.	Matches any character except \n.

[abcd]	Matches any character in the set.
[^abcd]	Excludes any character in the set.
[2-7a-mA-M]	Matches any character specified in the range.
\w	Matches any alphanumeric character and underscore.
\W	Matches any non-word character.
\s	Matches whitespace characters like, space, tab, new line etc.
\S	Matches any non-whitespace character.
\d	Matches any decimal character.
\D	Matches any non-decimal character.

Quantifiers could be added to specify number of times a character could appear.

Quantifier	**Description**
*	Zero or more matches.
+	One or more matches.
?	Zero or one matches.

{N}	N matches.
{N,}	N or more matches.
{N,M}	Between N and M matches.

The syntax of the control is as given:

```
<asp:RegularExpressionValidator ID="string" runat="server" ErrorMessage="string"
ValidationExpression="string" ValidationGroup="string">

</asp:RegularExpressionValidator>
```

CustomValidator

The CustomValidator control allows writing application specific custom validation routines for both the client side and the server side validation.

The client side validation is accomplished through the ClientValidationFunction property. The client side validation routine should be written in a scripting language, such as JavaScript or VBScript, which the browser can understand.

The server side validation routine must be called from the control's ServerValidate event handler. The server side validation routine should be written in any .Net language, like C# or VB.Net.

The basic syntax for the control is as given:

```
<asp:CustomValidator ID="CustomValidator1" runat="server"
ClientValidationFunction=.cvf_func. ErrorMessage="CustomValidator">

</asp:CustomValidator>
```

ValidationSummary

The ValidationSummary control does not perform any validation but shows a summary of all errors in the page. The summary displays the values of the ErrorMessage property of all validation controls that failed validation.

The following two mutually inclusive properties list out the error message:

- **ShowSummary** : shows the error messages in specified format.
- **ShowMessageBox** : shows the error messages in a separate window.

The syntax for the control is as given:

```
<asp:ValidationSummary ID="ValidationSummary1" runat="server"
DisplayMode = "BulletList" ShowSummary = "true" HeaderText="Errors:" />
```

Validation Groups

Complex pages have different groups of information provided in different panels. In such situation, a need might arise for performing validation separately for separate group. This kind of situation is handled using validation groups.

To create a validation group, you should put the input controls and the validation controls into the same logical group by setting their *ValidationGroup* property.

Code with Example:

The following example describes a form to be filled up by all the students of a school, divided into four houses, for electing the school president. Here, we use the validation controls to validate the user input.

This is the form in design view:

Code:-

The content file code is as given:

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

<table style="width: 66%;">

<tr>
<td class="style1" colspan="3" align="center">
<asp:Label ID="lblmsg"
Text="President Election Form : Choose your president"
runat="server" />
</td>
</tr>

<tr>
<td class="style3">
Candidate:
</td>

<td class="style2">
<asp:DropDownList ID="ddlcandidate" runat="server"  style="width:239px">
<asp:ListItem>Please Choose a Candidate</asp:ListItem>
<asp:ListItem>M H Kabir</asp:ListItem>
<asp:ListItem>Steve Taylor</asp:ListItem>
<asp:ListItem>John Abraham</asp:ListItem>
<asp:ListItem>Venus Williams</asp:ListItem>
</asp:DropDownList>
</td>
```

```
<td>
<asp:RequiredFieldValidator ID="rfvcandidate"
runat="server" ControlToValidate ="ddlcandidate"
ErrorMessage="Please choose a candidate"
InitialValue="Please choose a candidate">
</asp:RequiredFieldValidator>
</td>
</tr>

<tr>
<td class="style3">
House:
</td>

<td class="style2">
<asp:RadioButtonList ID="rblhouse" runat="server" RepeatLayout="Flow">
<asp:ListItem>Red</asp:ListItem>
<asp:ListItem>Blue</asp:ListItem>
<asp:ListItem>Yellow</asp:ListItem>
<asp:ListItem>Green</asp:ListItem>
</asp:RadioButtonList>
</td>

<td>
<asp:RequiredFieldValidator ID="rfvhouse" runat="server"
ControlToValidate="rblhouse" ErrorMessage="Enter your house name" >
</asp:RequiredFieldValidator>
<br />
</td>
```

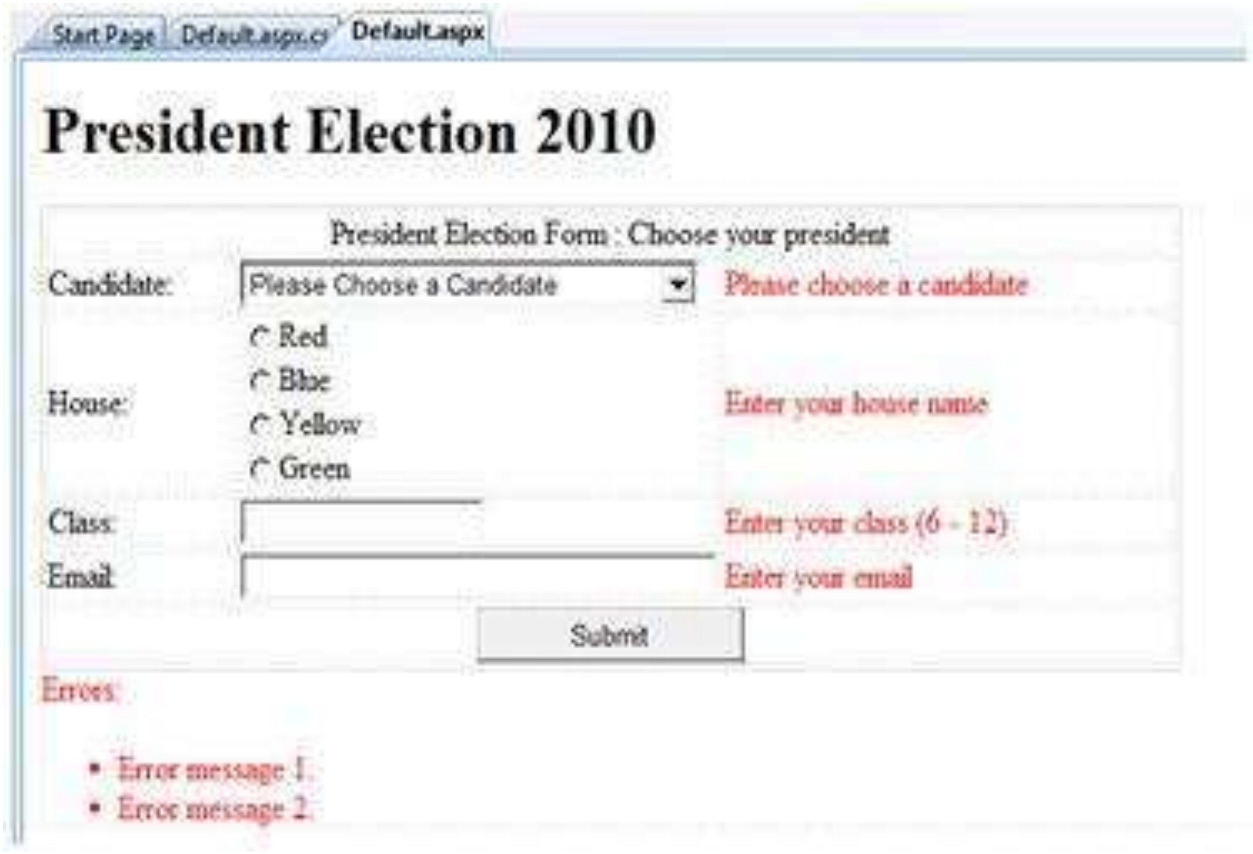
```
</tr>

<tr>
<td class="style3">
Class:
</td>

<td class="style2">
<asp:TextBox ID="txtclass" runat="server"></asp:TextBox>
</td>

<td>
<asp:RangeValidator ID="rvclass"
runat="server" ControlToValidate="txtclass"
ErrorMessage="Enter your class (6 - 12)" MaximumValue="12"
MinimumValue="6" Type="Integer">
</asp:RangeValidator>
</td>
</tr>

<tr>
<td class="style3">
Email:
</td>
<td class="style2">
<asp:TextBox ID="txtemail" runat="server" style="width:250px">
</asp:TextBox>
</td>
```

```
<td>
<asp:RegularExpressionValidator ID="remail" runat="server"
ControlToValidate="txtemail" ErrorMessage="Enter your email"
ValidationExpression="\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*">
</asp:RegularExpressionValidator>
</td>
</tr>
<tr>
<td class="style3" align="center" colspan="3">
<asp:Button ID="btnsubmit" runat="server" onclick="btnsubmit_Click"
style="text-align: center" Text="Submit" style="width:140px" />
</td>
</tr>
</table>
<asp:ValidationSummary ID="ValidationSummary1" runat="server"
DisplayMode ="BulletList" ShowSummary ="true" HeaderText="Errors:" />
</form>
```

The code behind the submit button:

```
protected void btnsubmit_Click(object sender, EventArgs e)
{
if (Page.IsValid)
{
lblmsg.Text = "Thank You";
}
else
{
lblmsg.Text = "Fill up all the fields";
   }
```

Output:-



Start Page | Default.aspx.cs | Default.aspx

President Election 2010

President Election Form : Choose your president

Candidate:	Please Choose a Candidate	Please choose a candidate
House:	○ Red ○ Blue ○ Yellow ○ Green	Enter your house name
Class:		Enter your class (6 - 12)
Email:		Enter your email
	Submit	

Errors:

- Error message 1.
- Error message 2.

Experiment No. 2

AIM*:* Design & develop ASP.net web application use master & content page.

Objective:-

To understand the concept of master page.

Ability to create website using master page

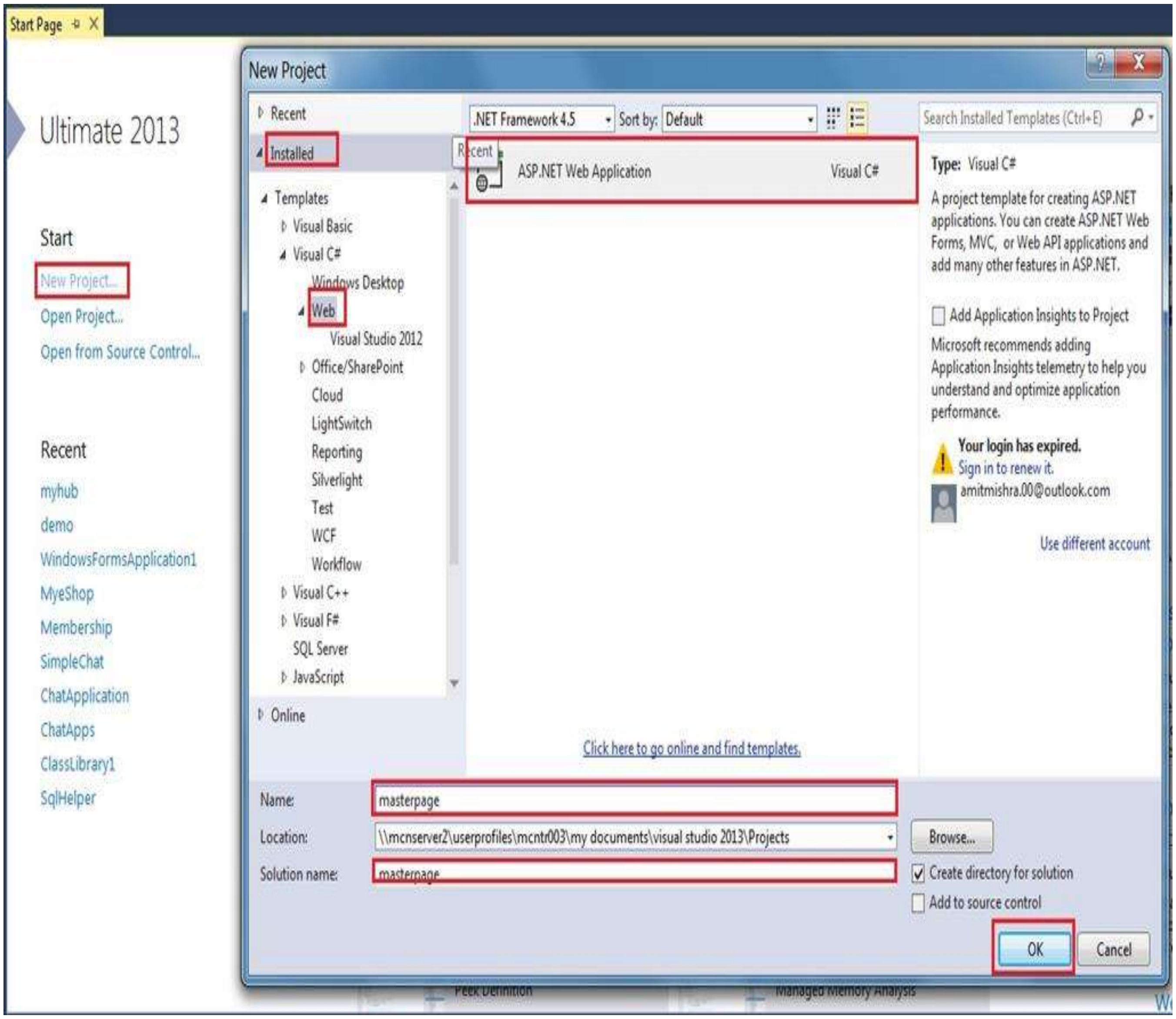
Theory:-

Master page provides the layout and functionality to the other pages. Creating a master page in

ASP.NET is very easy. Let's start creating master page step by step.

Step 1: Open new project in visual studio

New project->Installed->Web->ASP.NET Web Application (shown in the picture),



Type: Visual C#

A project template for creating ASP.NET applications. You can create ASP.NET Web Forms, MVC, or Web API applications and add many other features in ASP.NET.

☐ Add Application Insights to Project

Microsoft recommends adding Application Insights telemetry to help you understand and optimize application performance.

Your login has expired. Sign in to renew it.

amitmishra.00@outlook.com

Name: masterpage

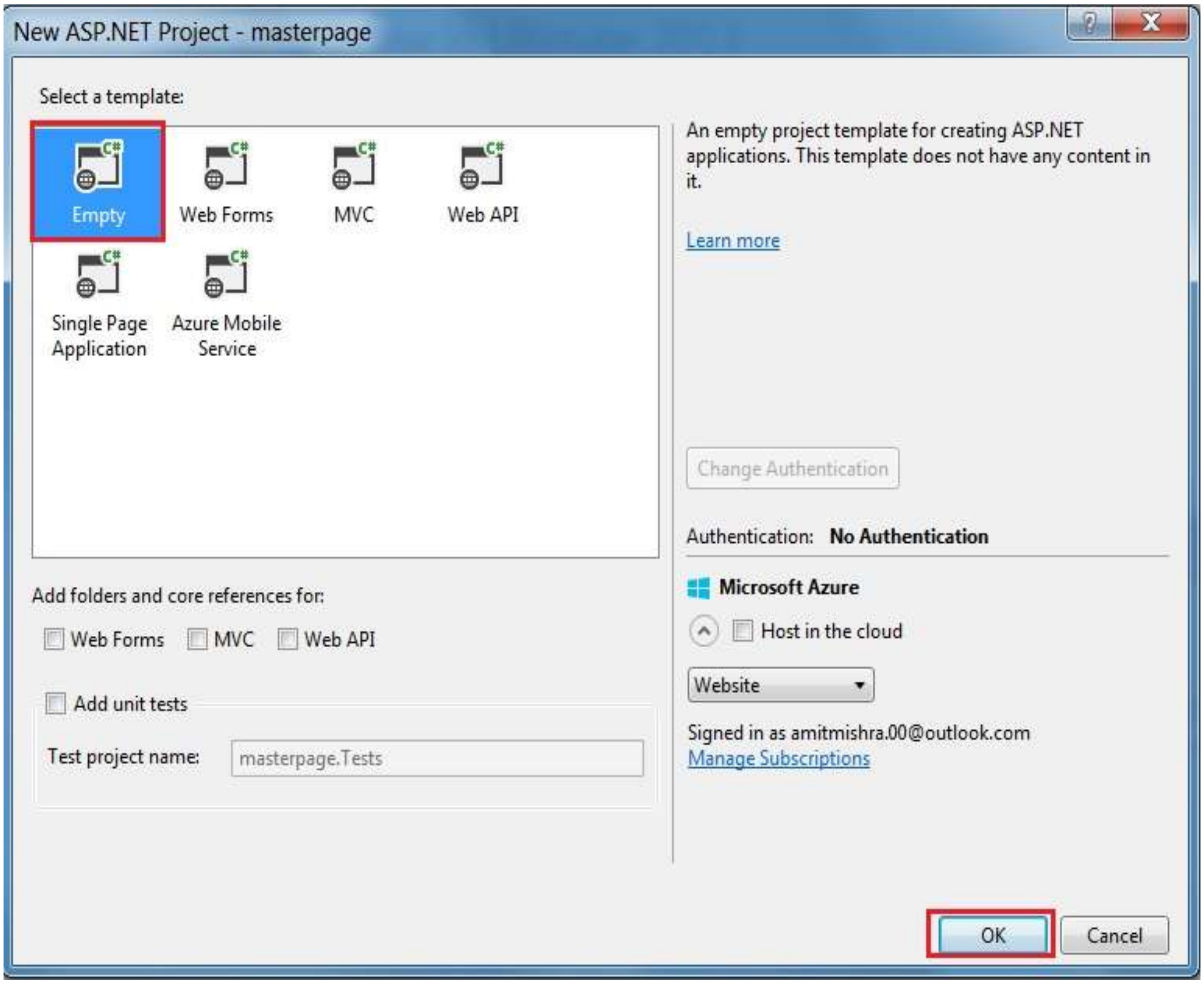
Location: \\mcnserver2\userprofiles\mcntr003\my documents\visual studio 2013\Projects

Solution name: masterpage

☑ Create directory for solution

☐ Add to source control

After clicking OK button in the Window, select Empty (shown in the picture),



An empty project template for creating ASP.NET applications. This template does not have any content in it.

Authentication: **No Authentication**

Add folders and core references for:

☐ Web Forms ☐ MVC ☐ Web API

☐ Add unit tests

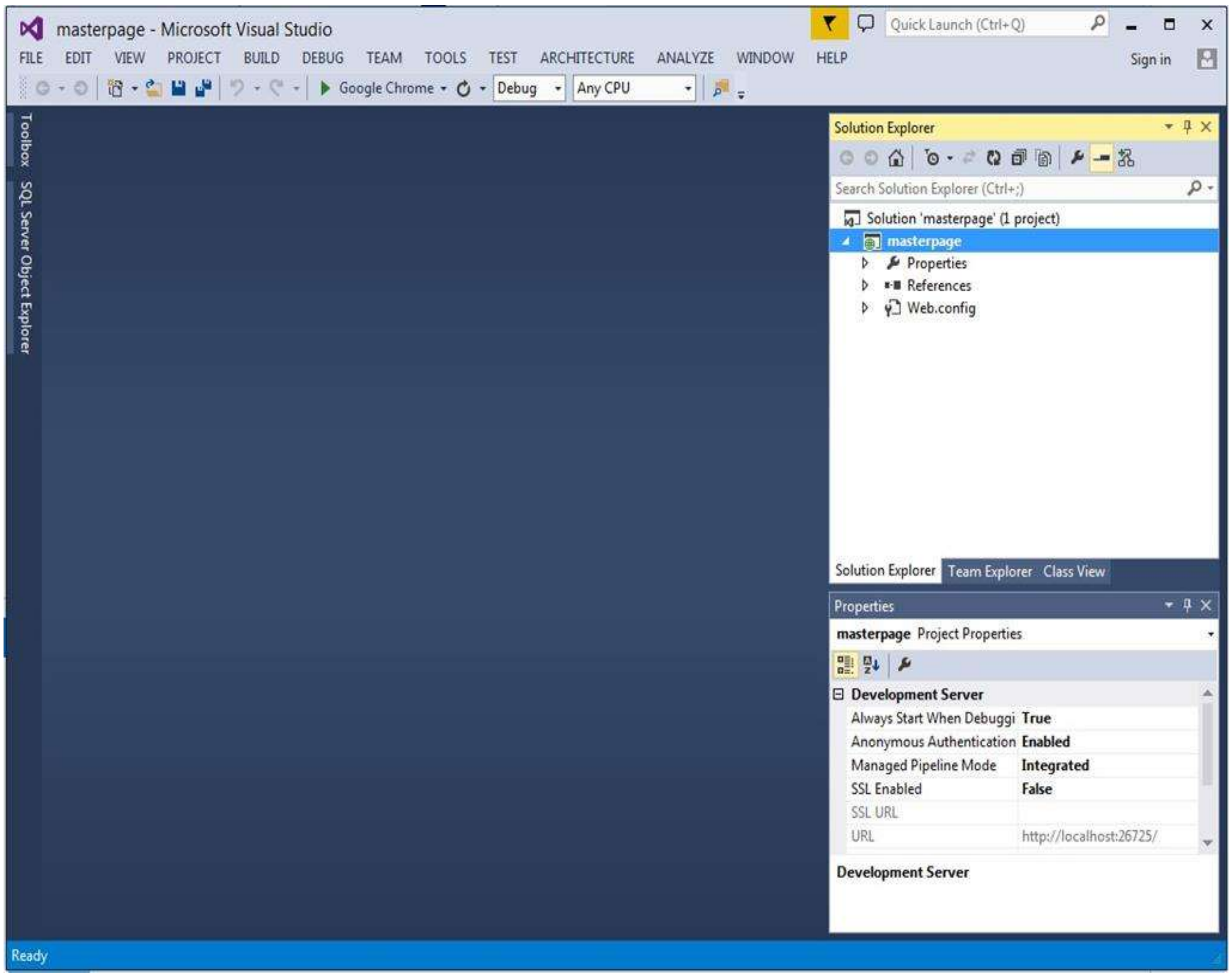
Test project name: masterpage.Tests

☐ Host in the cloud

Website

Signed in as amitmishra.00@outlook.com

After clicking OK button, project "masterpage" opens but no file is there (shown in the picture),


masterpage - Microsoft Visual Studio
Quick Launch (Ctrl+Q)
FILE EDIT VIEW PROJECT BUILD DEBUG TEAM TOOLS TEST ARCHITECTURE ANALYZE WINDOW HELP
Sign in
Google Chrome
Debug
Any CPU
Toolbox
SQL Server Object Explorer
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'masterpage' (1 project)
masterpage
Properties
References
Web.config
Solution Explorer
Team Explorer
Class View
Properties
masterpage Project Properties
Ready

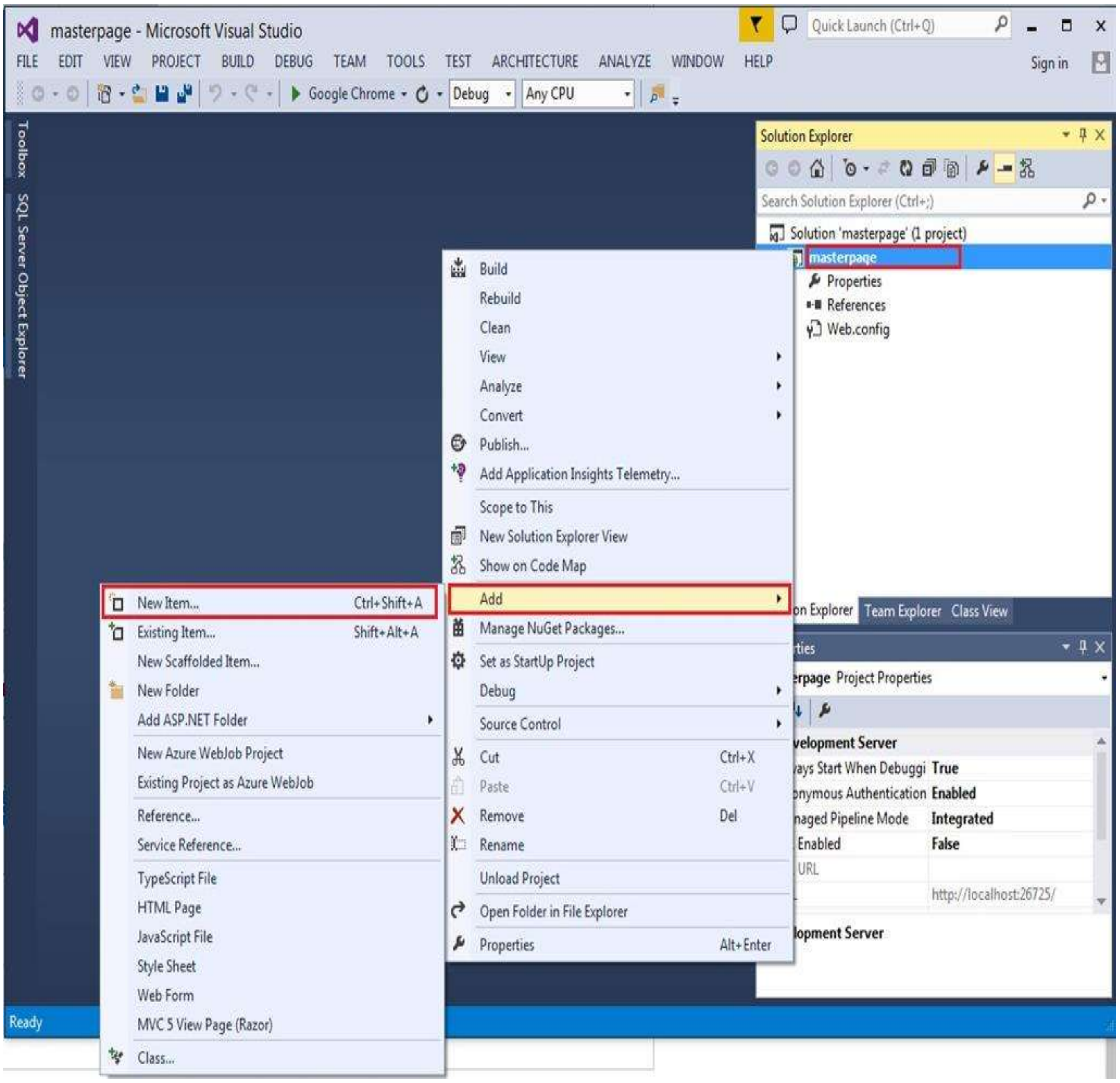
Development Server	
Always Start When Debuggi	True
Anonymous Authentication	Enabled
Managed Pipeline Mode	Integrated
SSL Enabled	False
SSL URL	
URL	http://localhost:26725/

Development Server

Step 2: Add new file in to our project.

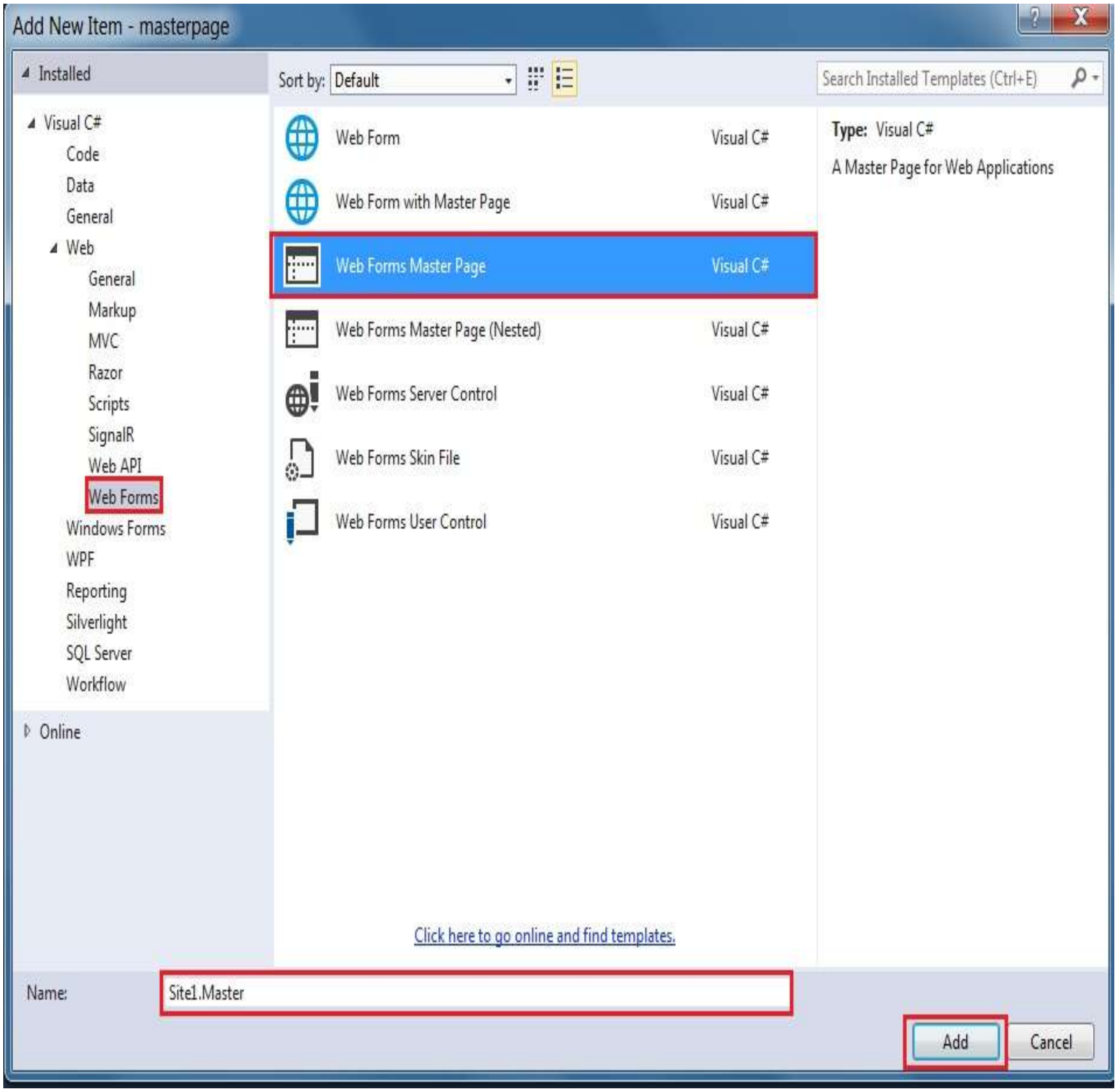
Add the master page into our project.

Right click Project->Add->New item (shown in the picture),



Always Start When Debuggi	True
Anonymous Authentication	Enabled
Managed Pipeline Mode	Integrated
Enabled	False
URL	
	http://localhost:26725/

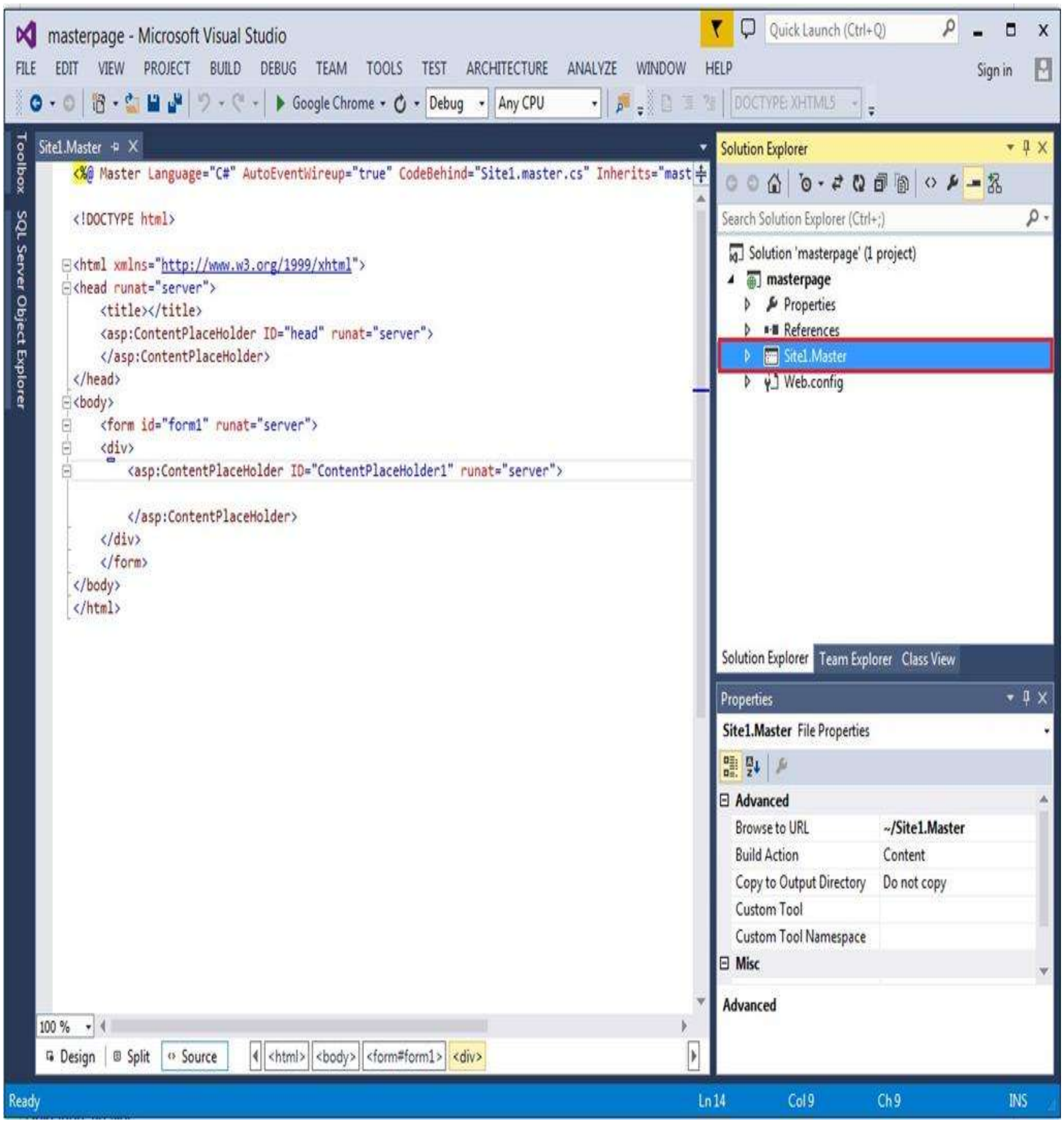
After clicking on new item, Window will open, select Web Form->Web Forms Master Page (shown in the picture),



Name: Site1.Master

After clicking the add button, master page 'site1.master' adds to our project.

Click on site1.master into Solution Explorer (shown in the picture),



```
<%@ Master Language="C#" AutoEventWireup="true" CodeBehind="Site1.master.cs" Inherits="mast

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
    <asp:ContentPlaceHolder ID="head" runat="server">
    </asp:ContentPlaceHolder>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:ContentPlaceHolder ID="ContentPlaceHolder1" runat="server">

        </asp:ContentPlaceHolder>
    </div>
    </form>
</body>
</html>
```

Properties

Site1.Master File Properties

Advanced	
Browse to URL	~/Site1.Master
Build Action	Content
Copy to Output Directory	Do not copy
Custom Tool	
Custom Tool Namespace	
Misc	

Advanced

Code:-

Step 3: Design the master page, using HTML.

HTML code of my master page is,

```
<%@ Master Language="C#" AutoEventWireup="true" CodeBehind="Site1.master.cs" Inherits="masterpage.Site1" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>c# corner</title>
    <link href="css/my.css" rel="stylesheet" />
    <asp:ContentPlaceHolder ID="head" runat="server">
    </asp:ContentPlaceHolder>
</head>
<body>
    <!DOCTYPE html>
<html>
<head>
    <title>my layout</title>
    <link rel="stylesheet" type="text/css" href="my.css">
</head>
<body>
<header id="header">
<h1>c# corner</h1>
</header>
<nav id="nav">
    <ul>
        <li><a href="home.aspx">Home</a></li>
        <li><a href="#">About</a></li>
        <li><a href="#">Article</a></li>
        <li><a href="#">Contact</a></li>
    </ul>
</nav>
<aside id="side">
    <h1>news</h1>
    <a href="#"><p>creating html website</p></a>
    <a href="#"><p>learn css</p></a>
    <a href="#">learn c#</a>
</aside>


    <div id="con">
```

```
        <asp:ContentPlaceHolder ID="ContentPlaceHolder1" runat="server">

        </asp:ContentPlaceHolder>
    </div>


<footer id="footer">
    copyright @c# corner
</footer>
</body>
</html>
    <form id="form1" runat="server">

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

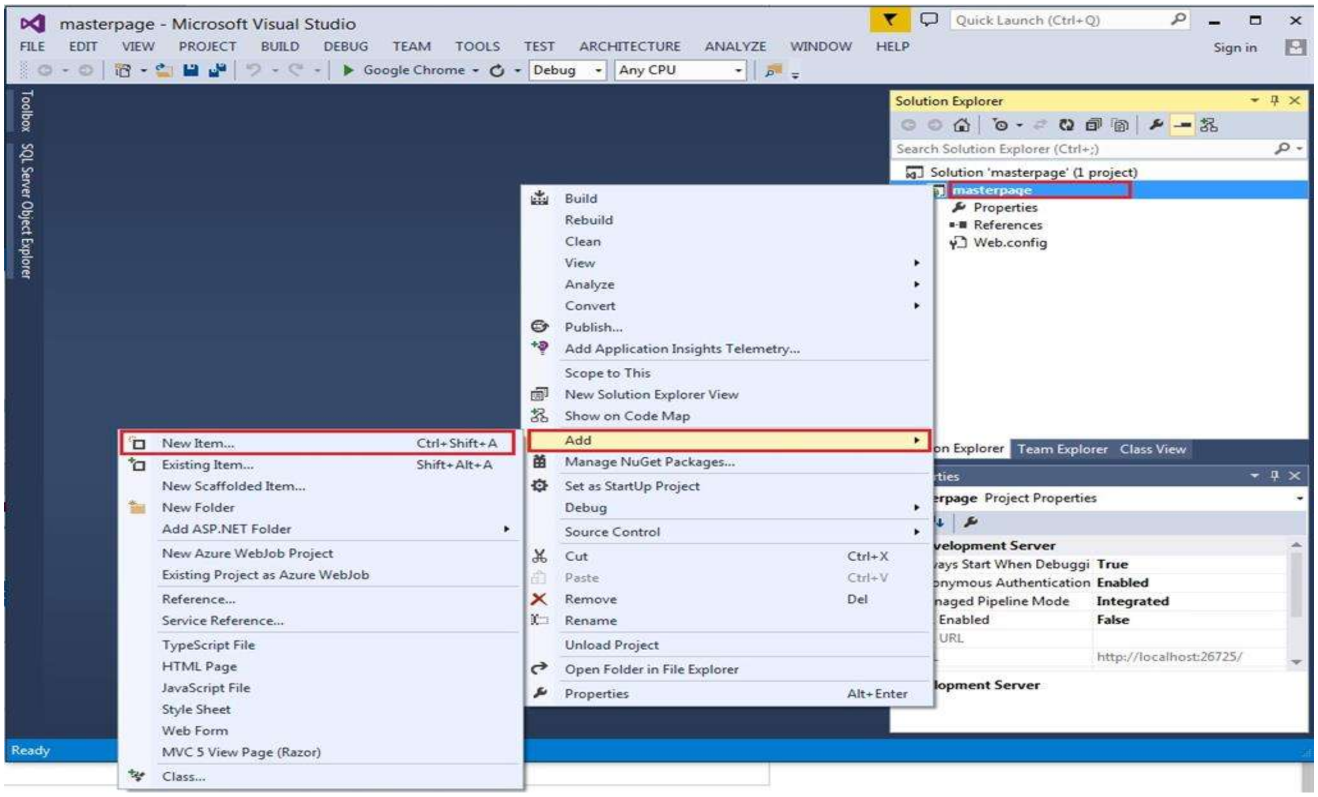
CSS Code

```
#header{
    color: #247BA0;
    text-align: center;
    font-size: 20px;
}
#nav{
    background-color:#FF1654;
    padding: 5px;
}
ul{

    list-style-type: none;
}
li a {
    color: #F1FAEE;
font-size: 30px;
column-width: 5%;
    }
    li
   {
    display: inline;
    padding-left: 2px;
    column-width: 20px;
   }
   a{
 text-decoration: none;
 margin-left:20px
    }
   li a:hover{
    background-color: #F3FFBD;
    color: #FF1654;
    padding:1%;
   }
   #side{
    text-align: center;
    float: right;
    width: 15%;
    padding-bottom: 79%;
    background-color: #F1FAEE;
   }
   #article{
    background-color: #EEF5DB;
    padding: 10px;
    padding-bottom: 75%;
   }
   #footer{
    background-color: #C7EFCF;
    text-align:center;
    padding-bottom: 5%;
    font-size: 20px;
}
   #con{
       border:double;
       border-color:burlywood;
   }
```

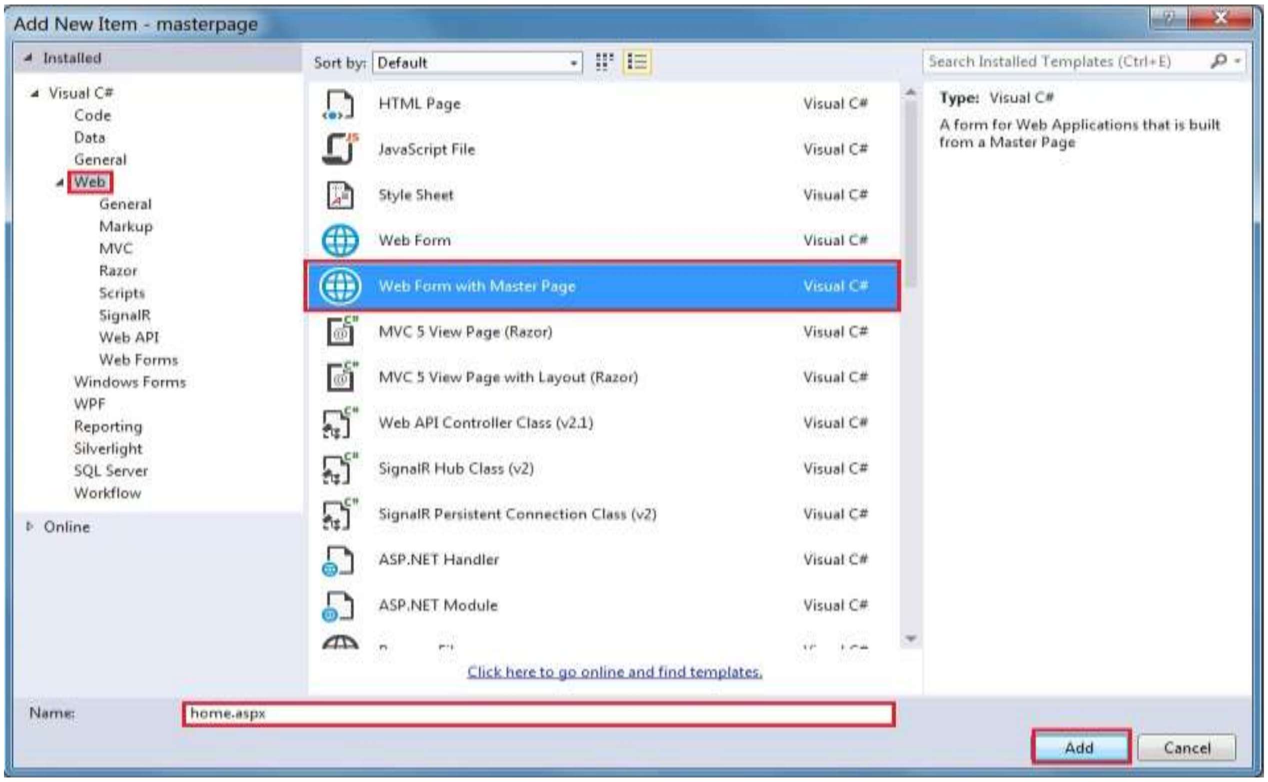
Our master page is designed. Move to the next step.

Step 4: Add web form in to our project.

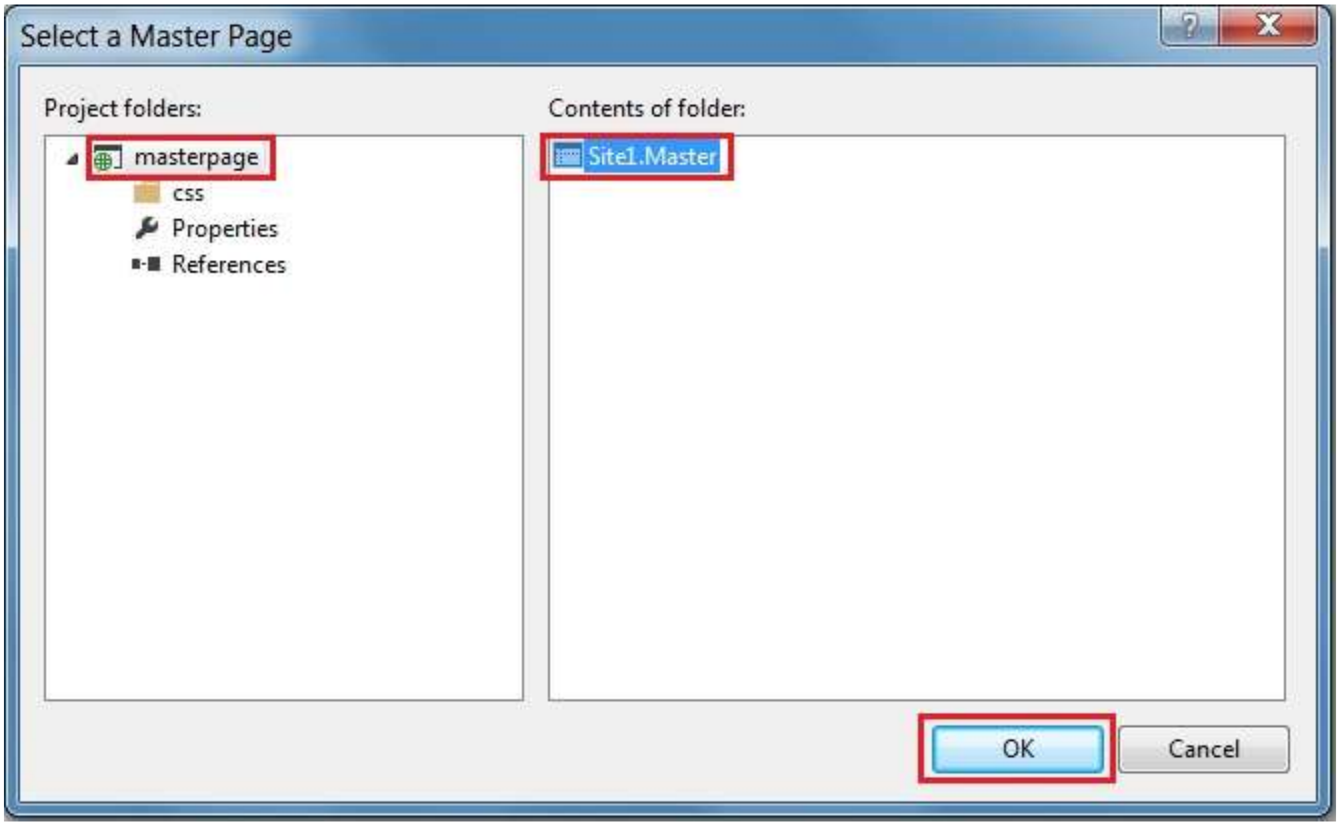
Right click on the project->Add->New item (shown in the picture),



Select Web form with the master page


Add New Item - masterpage
Installed
Visual C#
Web
Sort by: Default
HTML Page Visual C#
JavaScript File Visual C#
Style Sheet Visual C#
Web Form Visual C#
Web Form with Master Page Visual C#
MVC 5 View Page (Razor) Visual C#
MVC 5 View Page with Layout (Razor) Visual C#
Web API Controller Class (v2.1) Visual C#
SignalR Hub Class (v2) Visual C#
SignalR Persistent Connection Class (v2) Visual C#
ASP.NET Handler Visual C#
ASP.NET Module Visual C#
Type: Visual C#
A form for Web Applications that is built from a Master Page
Name: home.aspx
Add
Cancel

After clicking on that, add the button Window, open the selected masterpage->site1.master and click OK.



Now, design our homepage.

Here, we write home page only,

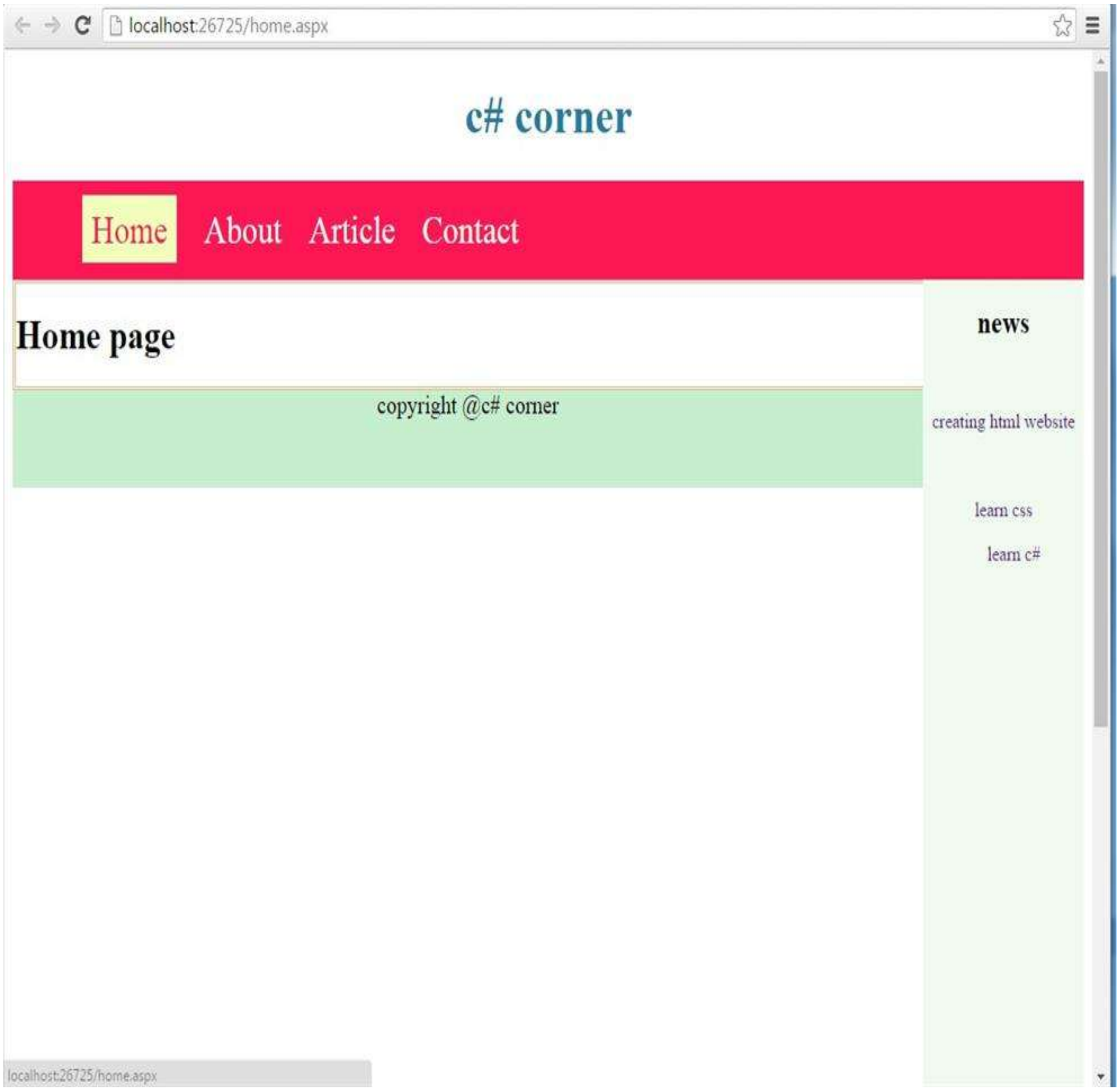
Home.aspx

```
<%@ Page Title="" Language="C#" MasterPageFile="~/Site1.Master" AutoEventWireup="true" CodeBehind="home.aspx.cs" Inherits="masterpage.home" %>
<asp:Content ID="Content1" ContentPlaceHolderID="head" runat="server">
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder1" runat="server">
    <h1>Home page</h1>
</asp:Content>
```

Output:

Finally, our Master page is created; build and run the project.

The master page looks aas shown in the picture:



Home page

copyright @c# corner

EXPERIMENT No. 3

Aim: Design a web form to insert , update, delete & show student information through database connectivity with (SQl/Oracle).

Objectives:

Fundamentals of Database connectivity
ASP.NET Database Connection
ASP.NET Read Database using SqlDataReader
Insert Database Record using InsertCommand
Update Database Record using UpdateCommand
Delete Database Record using DeleteCommand
Connecting Asp.net Controls to Data

Theory:

Fundamentals of Database connectivity
ASP.Net has the ability to work with a majority of databases. The most common being Oracle and Microsoft SQL Server. But with every database, the logic behind working with all of them is mostly the same.

In our examples, we will look at working with the Microsoft SQL Server as our database. For learning purposes, one can download and use the Microsoft SQL Server Express Edition. This is a free database software provided by Microsoft.

While working with databases, the following concepts which are common across all databases.

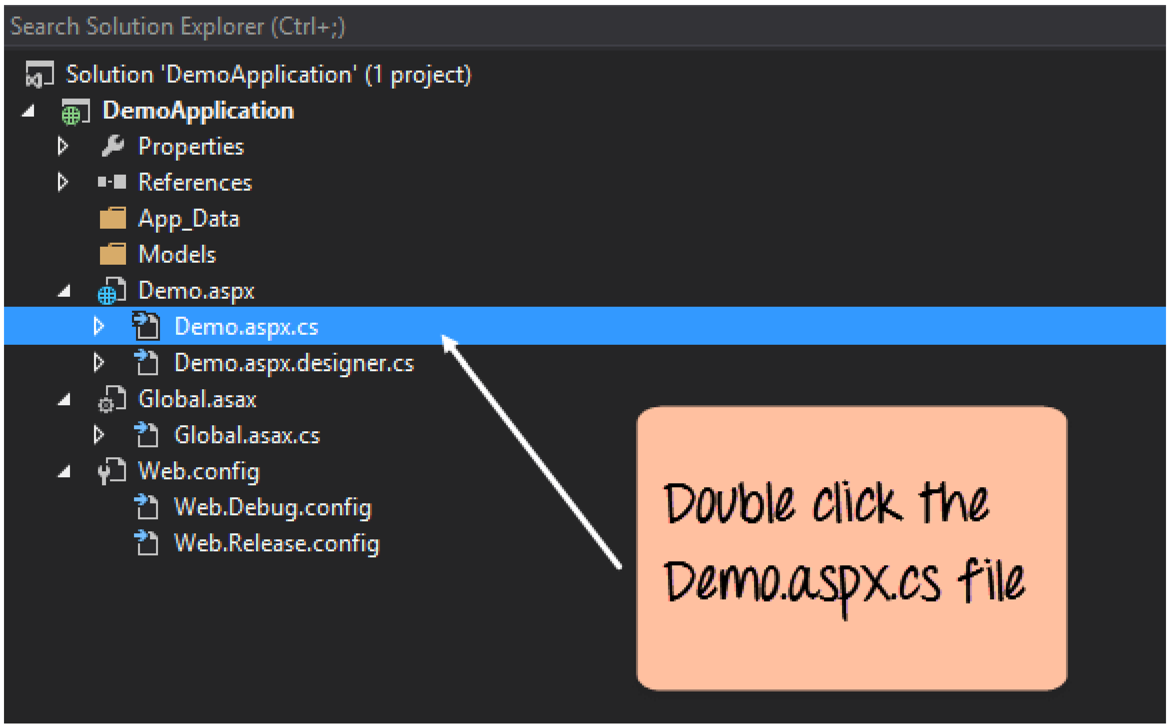
Connection – To work with the data in a database, the first obvious step is the connection. The connection to a database normally consists of the below-mentioned parameters.
Database name or Data Source – The first important parameter is the database name. Each connection can only work with one database at a time.
Credentials – The next important aspect is the 'username' and 'password'. This is used to establish a connection to the database.
Optional parameters - You can specify optional parameters on how .net should handle the connection to the database. For example, one can specify a parameter for how long the connection should stay active.
Selecting data from the database – Once the connection is established, data is fetched from the database. ASP.Net has the ability to execute 'sql' select command against the database. The 'sql' statement can be used to fetch data from a specific table in the database.
Inserting data into the database – ASP.Net is used to insert records into the database. Values for each row that needs to be inserted in the database are specified in ASP.Net.
Updating data into the database – ASP.Net can also be used to update existing records into the database. New values can be specified in ASP.Net for each row that needs to be updated into the database.
Deleting data from a database – ASP.Net can also be used to delete records from the database. The code is written to delete a particular row from the database.
Ok, now that we have seen the theory part of each operation. Now, let's see how to perform database operations in ASP.Net.

ASP.NET Database Connections
Let's now look at the code, which needs to be kept in place to create a connection to a database. In our example, we will connect to a database which has the name of Demodb. The credentials used to connect to the database are given below

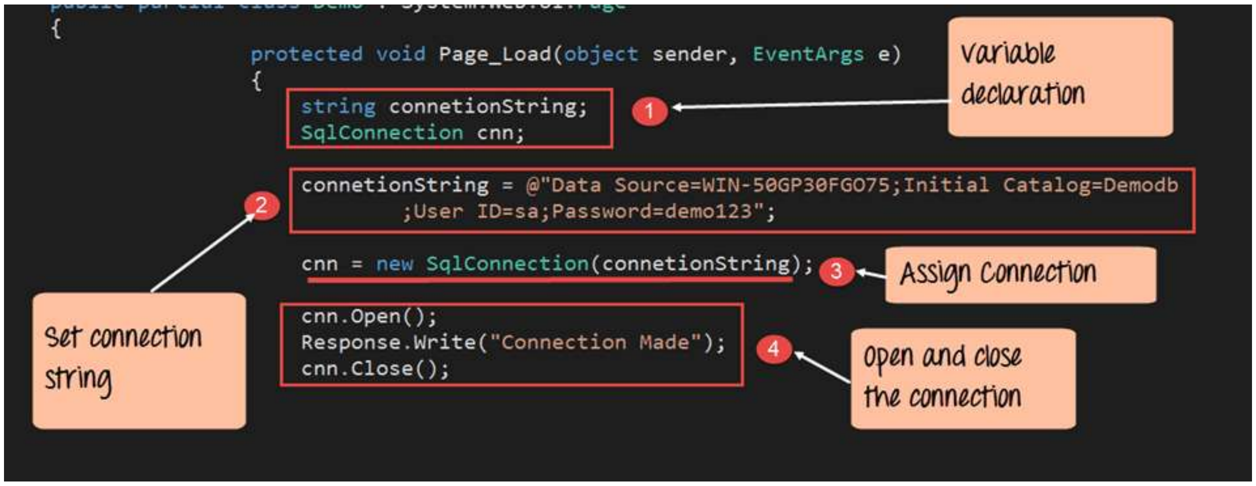
Username – sa
Password – demo123
Let's work with our current web application created in the earlier sections.

Start adding database operations to it.
Our example look's at establishing a simple connection. This connection is made to the Demodb database. This is done when the page is first launched.
When the connection is established, a message will be sent to the user. The message will indicate that the connection has been established.
Let's follow the below-mentioned steps to achieve this.

Step 1) Let's first ensure that you have your web application (DemoApplication) opened in Visual Studio. Double click the 'demo.aspx.cs' file to enter the code for the database connection.



Step 2) Add the below code which will be used to establish a connection to the database.



```
public partial class Demo : System.Web.UI.Page
{
        protected void Page_Load(object sender, EventArgs e)
        {
            string connetionString;
            SqlConnection cnn;

            connetionString = @"Data Source=WIN-50GP30FG075;Initial Catalog=Demodb
                    ;User ID=sa;Password=demo123";

            cnn = new SqlConnection(connetionString);

            cnn.Open();
            Response.Write("Connection Made");
            cnn.Close();
```

Code:-

```
namespace DemoApplication
{
        public partial class Demo  System.Web.UI.Page
    {
          protected void Page_Load(object sender, EventArgs e)
          {
                string connetionString;
                SqlConnection cnn;

                connetionString = @"Data Source=WIN-50GP30FGO75;Initial
Catalog=Demodb ;User ID=sa;Password=demol23";

                cnn = new SqlConnection(connetionString);

                cnn.Open();

                Response.Write("Connection MAde");
                conn.Close();

          }
        }
}
```

Code Explanation:-

1. The first step is to create variables. It will be used to create the connection string and the connection to the SQL Server database.
2. The next step is to actually create the connection string. The connection string consists of the following parts

- Data Source – This is the name of the server on which the database resides. In our case, it resides on a machine called WIN- 50GP30FGO75.
- The Initial Catalog is used to specify the name of the database
- The UserID and Password are the credentials required to connect to the database.

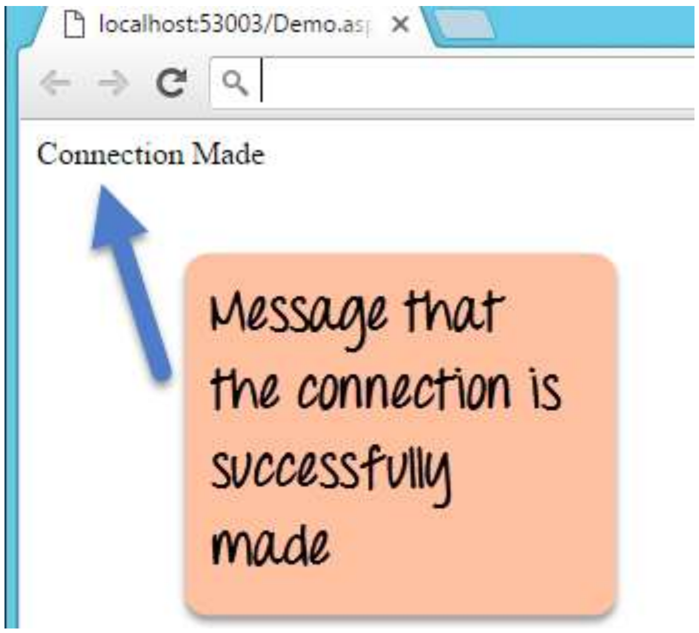
3. Next, we assign the connecting string to the variable 'cnn'.

- The variable cnn is of type SqlConnection. This is used to establish a connection to the database.
- SqlConnection is a class in ASP.Net, which is used to create a connection to a database.
- To use this class, you have to first create an object of this class. Hence, here we create a variable called 'cnn' which is of the type SqlConnection.

4. Next, we use the open method of the cnn variable to open a connection to the database. We display a message to the user that the connection is established. This is done via the 'response.write' method. We then close the connection to the database.

When the above code is set, and the project is executed using Visual Studio. You will get the below output. Once the form is displayed, click the Connect button.

Output:-



Connection Made

The output message displayed in the browser will show that the connection to the database is made.

ASP.NET Read Database using SqlDataReader

To show data accessed using Asp.Net, let us assume the following artifacts in our database.

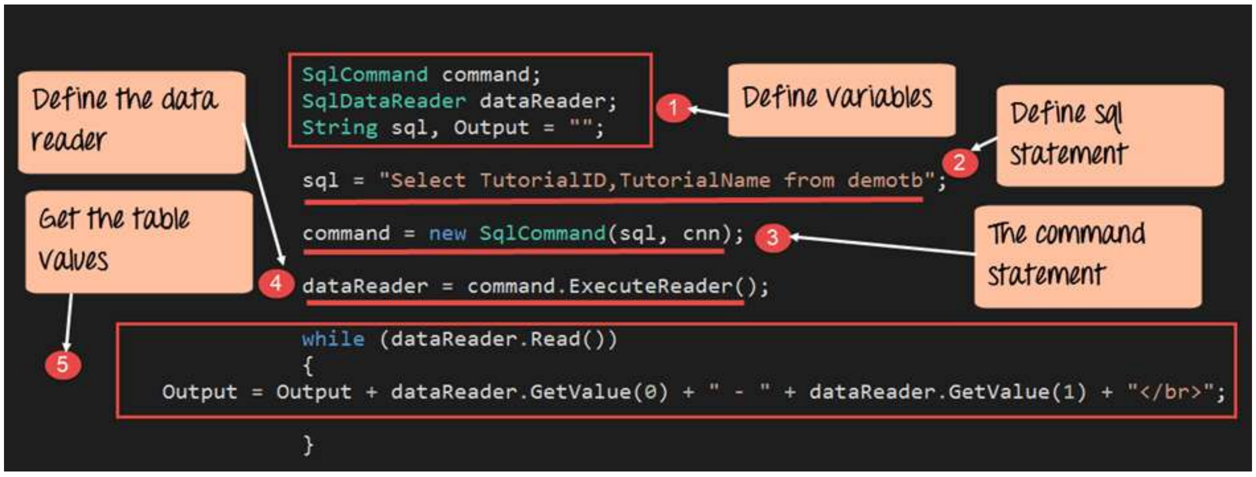
1. A table called demotb. This table will be used to store the ID and names of various Tutorials.
2. The table will have two columns, one called "TutorialID" and the other called "TutorialName."
3. For the moment, the table will have two rows as shown below.

TutorialID	TutorialName
1	C#
2	ASP.Net

Let's change the code so that we can query for this data and display the information on the web page itself. Note that the code entered is in continuation to that written for the data connection module.

Step 1) Let's split the code into two parts,

- The first part will be to construct our "select" statement. It will be used to read the data from the database.
- We will then execute the "select" statement against the database. This will fetch all the table rows accordingly.



```
SqlCommand command;
SqlDataReader dataReader;
String sql, Output = "";

sql = "Select TutorialID,TutorialName from demotb";

command = new SqlCommand(sql, cnn);

dataReader = command.ExecuteReader();

                    while (dataReader.Read())
                    {
    Output = Output + dataReader.GetValue(0) + " - " + dataReader.GetValue(1) + "</br>";

                    }
```

Code Explanation:-

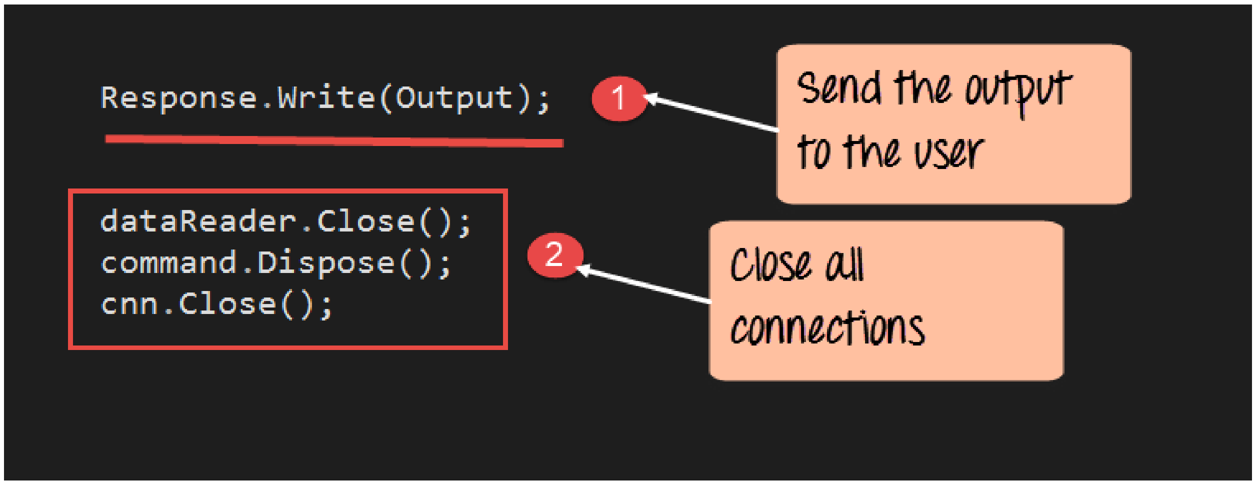
1. The first step is to create the following variables

- SQLCommand – The 'SQLCommand' is a class defined within C#. This class is used to perform operations of reading and writing into the database. Hence, the first step is to make sure that we create a variable type of this class. This variable will then be used in subsequent steps of reading data from our database.
- The DataReader object is used to get all the data specified by the SQL query. We can then read all the table rows one by one using the data reader.
- We then define two string variables. One is "SQL" to hold our SQL command string. The next is the "Output" which will contain all the table values.

2. The next step is to actually define the SQL statement. This will be used against our database. In our case, it is "Select TutorialID, TutorialName from demotb". This will fetch all the rows from the table demotb.
3. Next, we create the command object which is used to execute the SQL statement against the database. In the SQL command, you have to pass the connection object and the SQL string.
4. Next, we will execute the data reader command, which will fetch all the rows from the demotb table.
5. Now that we have all the rows of the table with us, we need a mechanism to access the row one by one.

- For this, we will use the 'while' statement.
- The 'while' statement will be used to access the rows from the data reader one at a time.
- We then use the 'GetValue' method to get the value of TutorialID and TutorialName.

Step 2) In the final step, we will just display the output to the user. Then we will close all the objects related to the database operation.



```
Response.Write(Output);

dataReader.Close();
command.Dispose();
cnn.Close();
```

Code:-

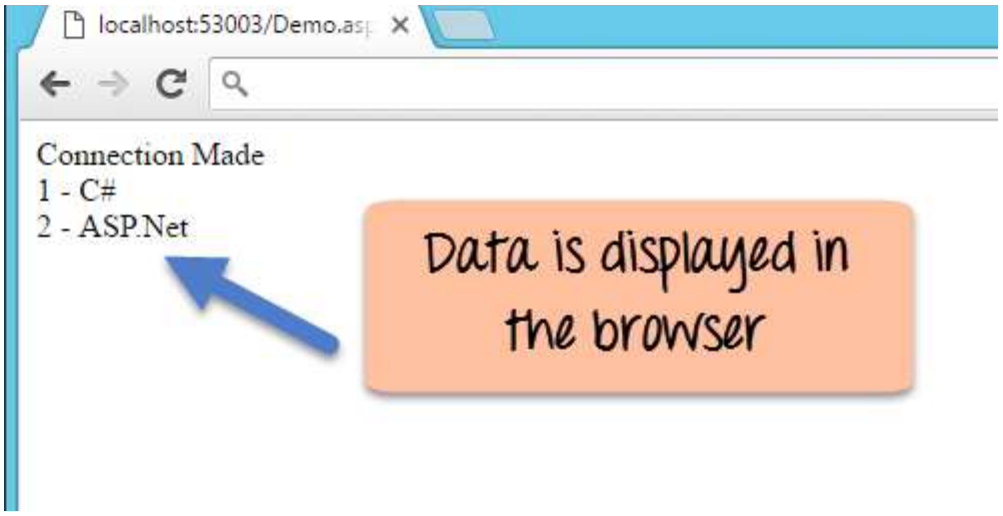
```
namespace DemoApplication
{
        public partial class Demo System.Web.UI.Page
    {
          protected void Page_Load(object sender, EventArgs e)
          {
                SqlCommand command;
                SqlDataReader dataReader;
                String sql, Output =" ";
                sql = "Select TutorialID,TutorialName from demotb";

                command = new SqlCommand(sql, cnn);

                dataReader = command.ExecuteReader();
                while (dataReader.Read())
                {
                  Output = Output + dataReader.GetValue(0) + "-" +
dataReader.GetValue(1) + "</br>";
                }

                Response.Write(Output);
                dataReader.Close();
                command.dispose();
                conn.Close();

          }
        }
}
```

Code Explanation:-

1. We will continue our code by displaying the value of the Output variable. This is done using the Response.Write method.
2. We finally close all the objects related to our database operation. Remember this is always a good practice.

When the above code is set, and the project is run using Visual Studio, you will get the below output.

Output:-



```
Connection Made
1 - C#
2 - ASP.Net
```

From the output, you can clearly see that the program was able to get the values from the database. The data is then displayed in the browser to the user.

Insert Database Record using InsertCommand

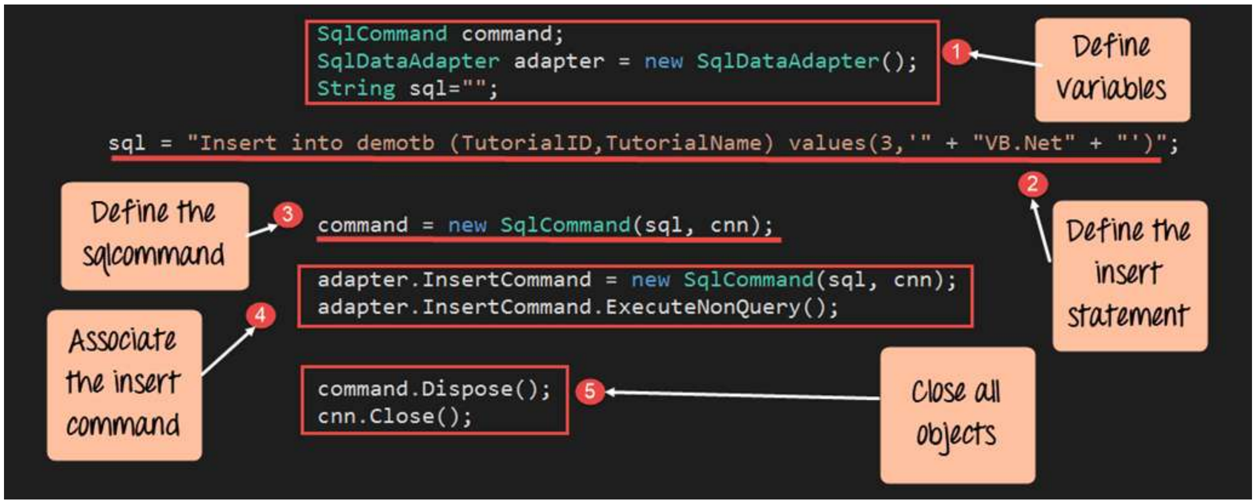
Just like Accessing data, ASP.Net has the ability to insert records into the database as well. Let's take the same table structure used for inserting records.

TutorialID	TutorialName
1	C#
2	ASP.Net

Let's change the code in our form, so that we can insert the following row into the table

TutorialID	TutorialName
3	VB.Net

Step 1) As the first step let's add the following code to our program. The below code snippet will be used to insert an existing record in our database.



```
SqlCommand command;
SqlDataAdapter adapter = new SqlDataAdapter();
String sql="";

sql = "Insert into demotb (TutorialID,TutorialName) values(3,'" + "VB.Net" + "')";

command = new SqlCommand(sql, cnn);

adapter.InsertCommand = new SqlCommand(sql, cnn);
adapter.InsertCommand.ExecuteNonQuery();

command.Dispose();
cnn.Close();
```

Code:-

```
namespace DemoApplication
{
        public partial class Demo System.Web.UI.Page
    {
          protected void Page_Load(object sender, EventArgs e)
          {
                SqlCommand command;
                SqlDataAdapter adapter = new SqlDataAdapter();
                String sql="";

                sql = "Insert into demotb(TutorialID,TutorialName) value(3, '" +
"VB.Net +"')";

                command = new SqlCommand(sql,cnn);
                adapter.InsertCommand = new SqlCommand(sql,cnn);
                adapter.InsertCommand.ExecuteNonQuery();

                command.Dispose():
                cnn.Close();

          }
        }
}
```

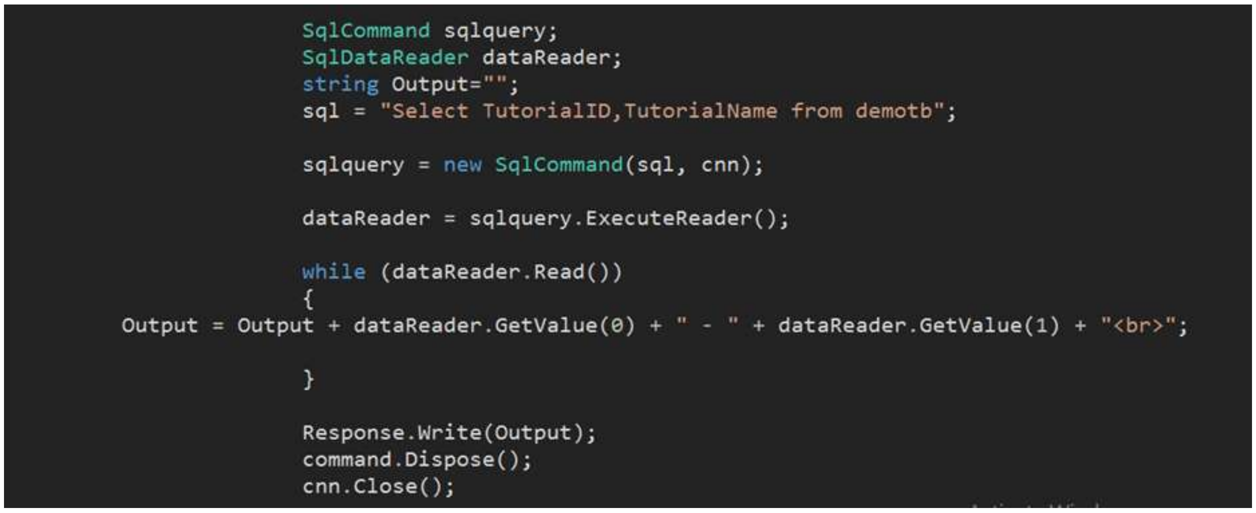
Code Explanation:-

1. The first step is to create the following variables
    1. SQLCommand – This data type is used to define objects. These objects perform SQL operations against a database. This object will hold the SQL command which will run against our SQL Server database.
    2. The DataAdapter object is used to perform insert, delete and update SQL commands
    3. We then define a string variable, which is "SQL" to hold our SQL command string.
2. The next step is actually to define the SQL statement, which will be used against our database. In our case, we are issuing an insert statement. This will insert the record of TutorialID=1 and TutorialName=VB.Net
3. Next, we create the command object which is used to execute the SQL statement against the database. In the SQL command, you have to pass the connection object and the SQL string
4. In our data adapter command,

- Associate the insert SQL command to the adapter.
- Then issue the 'ExecuteNonQuery' method. This is used to execute the Insert statement against our database.
- The 'ExecuteNonQuery' method is used in C# to issue any DML statements (insert, delete and update operation) against the database.
- To issue any table statements in ASP.Net, one need's to use the 'ExecuteNonQuery' method.

5. We finally close all the objects related to our database operation. Remember this is always a good practice.

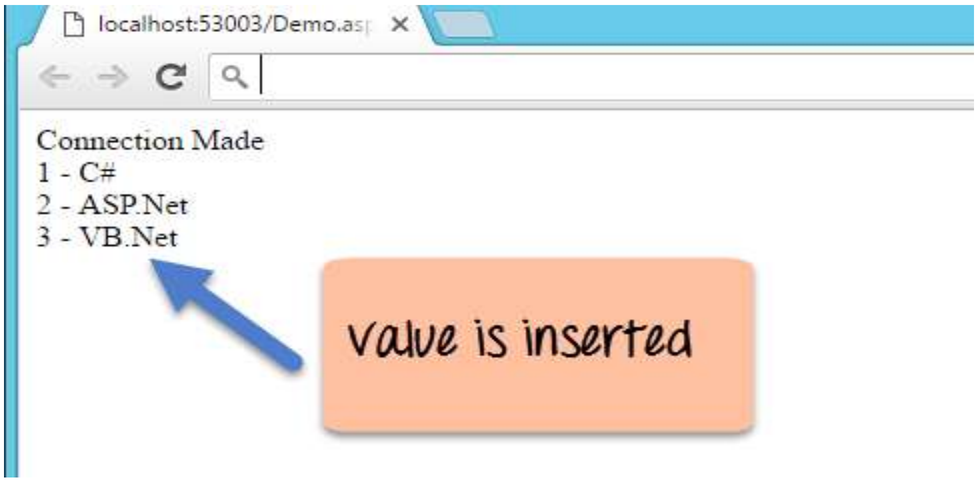
Step 2) As a second step, let's add the same code as in the Accessing data section. This is to display the recent table data in the browser. For that, we will add the below code to the demo.aspx.cs file.


```
                        SqlCommand sqlquery;
                        SqlDataReader dataReader;
                        string Output="";
                        sql = "Select TutorialID,TutorialName from demotb";

                        sqlquery = new SqlCommand(sql, cnn);

                        dataReader = sqlquery.ExecuteReader();

                        while (dataReader.Read())
                        {
         Output = Output + dataReader.GetValue(0) + " - " + dataReader.GetValue(1) + "<br>";

                        }

                        Response.Write(Output);
                        command.Dispose();
                        cnn.Close();
```

Code:-

```
namespace DemoApplication
{
        public partial class Demo System.Web.UI.Page
    {
          protected void Page_Load(object sender, EventArgs e)
          {
                SqlCommand sqlquery;
                SqlDataReader dataReader;
                String Output =" ";
                sql = "Select TutorialID,TutorialName from demotb";

                sqlquery = new SqlCommand(sql, cnn);

                dataReader = command.ExecuteReader();
                while (dataReader.Read())
                {
                  Output = Output + dataReader.GetValue(0) + "-" +
dataReader.GetValue(1) + "</br>";
                }

                Response.Write(Output);
                dataReader.Close();
                command.dispose();
                conn.Close();
          }
        }
}
```

Output:-



```
Connection Made
1 - C#
2 - ASP.Net
3 - VB.Net
```

In the browser window, you will see that the rows was successfully inserted in the database.

Update Database Record using UpdateCommand

ASP.Net has the ability to update existing records from a database. Let's take the same table structure which was used above for the example to insert records.

TutorialID	TutorialName
1	C#
2	ASP.Net
3	VB.Net

Let's change the code in our form, so that we can update the following row. The old row value is TutorialID as "3" and Tutorial Name as "VB.Net". Which we will update it to "VB.Net complete" while the row value for Tutorial ID will remain same.

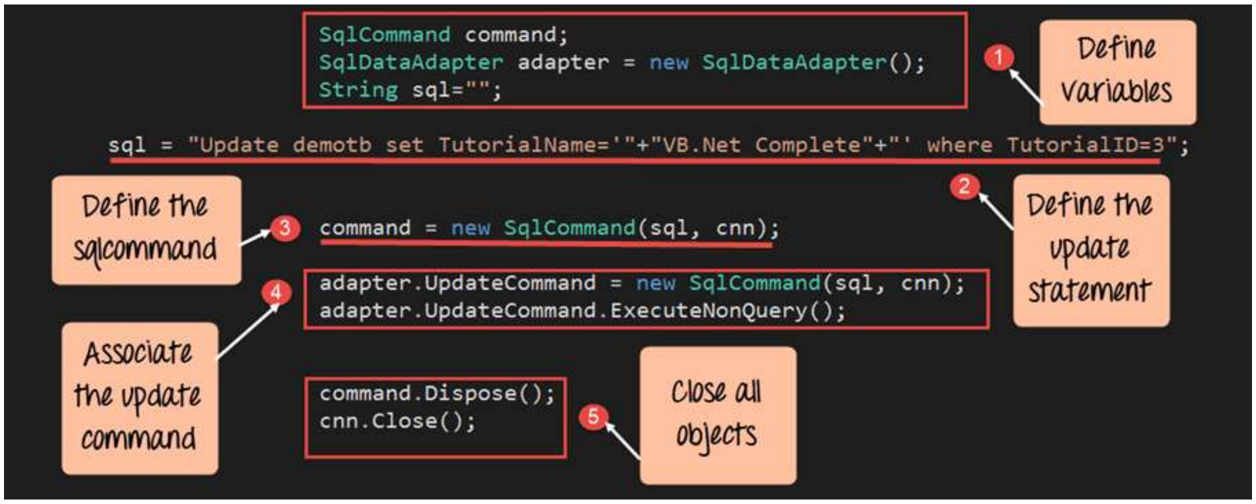
Old row

TutorialID	TutorialName
3	VB.Net

New row

TutorialID	TutorialName
3	VB.Net complete

Step 1) As the first step let's add the following code to our program. The below code snippet will be used to update an existing record in our database.



```
SqlCommand command;
SqlDataAdapter adapter = new SqlDataAdapter();
String sql="";

sql = "Update demotb set TutorialName='"+"VB.Net Complete"+"' where TutorialID=3";

command = new SqlCommand(sql, cnn);

adapter.UpdateCommand = new SqlCommand(sql, cnn);
adapter.UpdateCommand.ExecuteNonQuery();

command.Dispose();
cnn.Close();
```

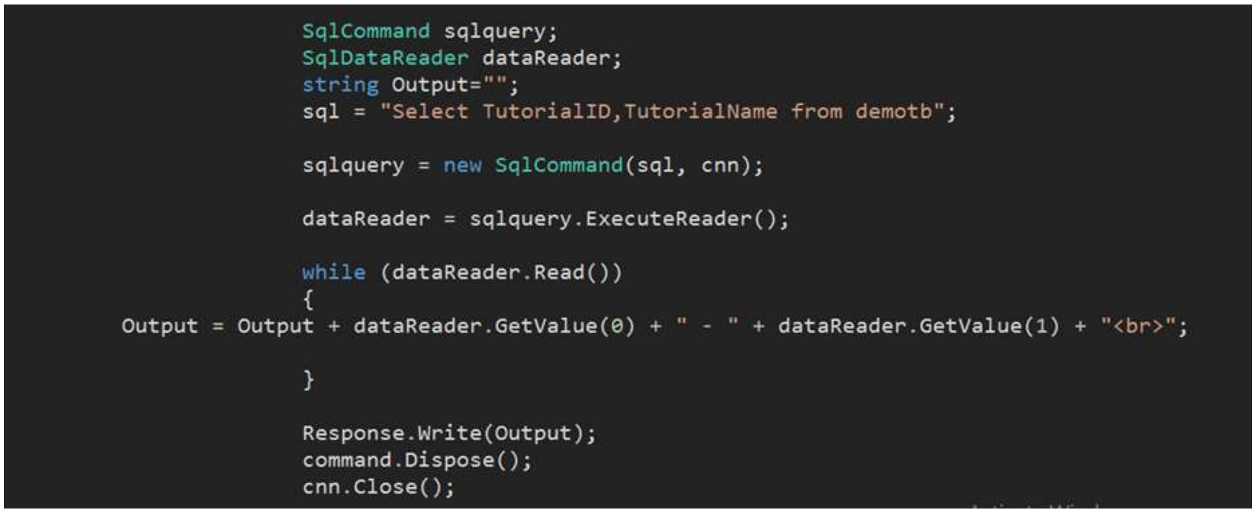
Code:-

```
namespace DemoApplication
{
        public partial class Demo System.Web.UI.Page
    {
          protected void Page_Load(object sender, EventArgs e)
          {
                SqlCommand command;
                SqlDataAdapter adapter = new SqlDataAdapter();
                String sql="";

                sql = "Update demotb set TutorialName='"VB.Net Complete"+"' where 
TutorialID=3";

                command = new SqlCommand(sql,cnn);

                adapter.InsertCommand = new SqlCommand(sql,cnn);
                adapter.InsertCommand.ExecuteNonQuery;

                command.Dispose():
                cnn.Close();

          }
        }
}
```

Code Explanation:-

1. The first step is to create the following variables
    1. SQLCommand – his data type is used to define objects to perform SQL operations against a database. This object will hold the SQL command which will run against our SQL Server database.
    2. The dataadapter object is used to perform insert, delete and update SQL commands
    3. We then define a string variable, which is SQL to hold our SQL command string.
2. The next step is actually to define the SQL statement which will be used against our database. In our case, we are issuing an 'update' statement. This will update the Tutorial name to "VB.Net Complete". The TutorialID will remain unchanged, and the value will be 3.
3. Next, we will create the command object. This is used to execute the SQL statement against the database. In the SQL command, you have passed the connection object and the SQL string
4. In our data adapter command, we now associate the insert SQL command to our adapter. We then issue the ExecuteNonQuery method. This is used to execute the Update statement against our database.
5. We finally close all the objects related to our database operation. Remember this is always a good practice.

Step 2) As a second step, let's add the same code as in the Accessing data section. This is to display the recent table data in the browser. For that, we will add the below code

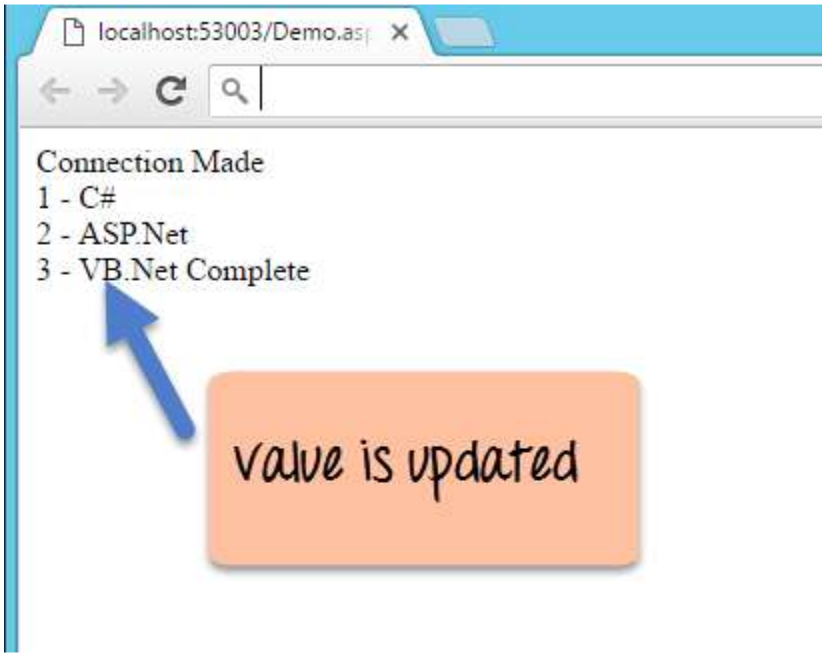


```
                    SqlCommand sqlquery;
                    SqlDataReader dataReader;
                    string Output="";
                    sql = "Select TutorialID,TutorialName from demotb";

                    sqlquery = new SqlCommand(sql, cnn);

                    dataReader = sqlquery.ExecuteReader();

                    while (dataReader.Read())
                    {
        Output = Output + dataReader.GetValue(0) + " - " + dataReader.GetValue(1) + "<br>";

                    }

                    Response.Write(Output);
                    command.Dispose();
                    cnn.Close();
```

```
namespace DemoApplication
{
        public partial class Demo System.Web.UI.Page
    {
          protected void Page_Load(object sender, EventArgs e)
          {
                SqlCommand sqlquery;
                SqlDataReader dataReader;
                String Output =" ";
                sql = "Select TutorialID,TutorialName from demotb";

                sqlquery = new SqlCommand(sql, cnn);

                dataReader = command.ExecuteReader();

                while (dataReader.Read())
                {
                  Output = Output + dataReader.GetValue(0) + "-" +
dataReader.GetValue(1) + "</br>";
                }

                Response.Write(Output);
                dataReader.Close();
                command.dispose();
                conn.Close();
          }
        }
}
```

When the above code is set, and the project is executed using Visual Studio, you will get the below output.

Output:-



```
Connection Made
1 - C#
2 - ASP.Net
3 - VB.Net Complete
```

In the browser window, you will see that the rows were successfully updated in the database.

Delete Database Record using DeleteCommand

ASP.Net can delete existing records from a database. Let's take the same table structure which was used above for the example to delete records.

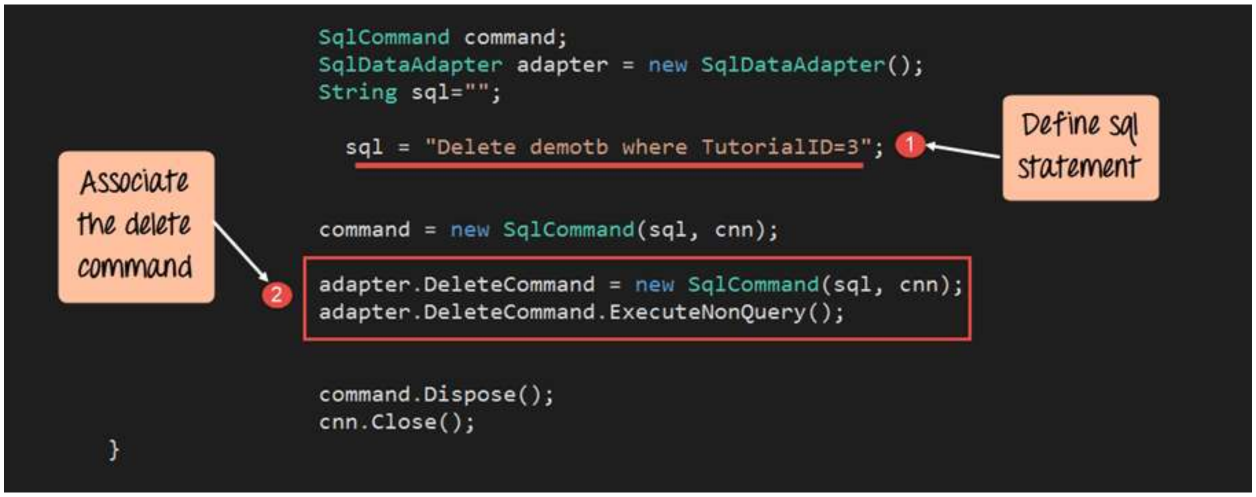
TutorialID	TutorialName
1	C#
2	ASP.Net
3	VB.Net complete

Let's change the code in our form so that we can delete the following row

TutorialID	TutorialName
3	VB.Net complete

So let's add the following code to our program. The below code snippet will be used to delete an existing record in our database.

Step 1) As the first step let's add the following code to our program. The below code snippet will be used to delete an existing record in our database.



```
        SqlCommand command;
        SqlDataAdapter adapter = new SqlDataAdapter();
        String sql="";

          sql = "Delete demotb where TutorialID=3";

        command = new SqlCommand(sql, cnn);

        adapter.DeleteCommand = new SqlCommand(sql, cnn);
        adapter.DeleteCommand.ExecuteNonQuery();

        command.Dispose();
        cnn.Close();
    }
```

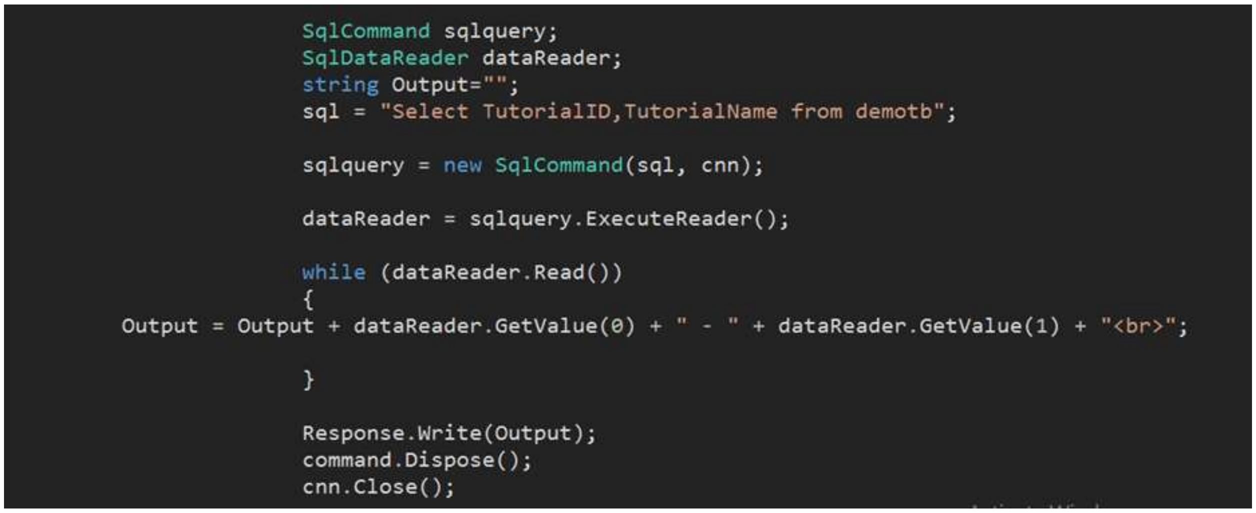
Code:-

```
namespace DemoApplication
{
        public partial class Demo System.Web.UI.Page
    {
          protected void Page_Load(object sender, EventArgs e)
          {
                SqlCommand command;
                SqlDataAdapter adapter = new SqlDataAdapter();
                String sql="";

                sql = "Delete demotb where TutorialID=3";

                command = new SqlCommand(sql,cnn);

                adapter.InsertCommand = new SqlCommand(sql,cnn);
                adapter.InsertCommand.ExecuteNonQuery;

                command.Dispose():
                cnn.Close();

          }
        }
}
```

Code Explanation:-

1. The Key difference in this code is that we are now issuing the delete SQL statement. The delete statement is used to delete the row in the demotb table in which the TutorialID has a value of 3.
2. In our data adapter command, we now associate the insert SQL command to our adapter. We also issue the 'ExecuteNonQuery' method which is used to execute the delete statement against our database.

Step 2) As a second step, let's add the same code as in the Accessing data section. This is to display the recent table data in the browser. For that, we will add the below code.



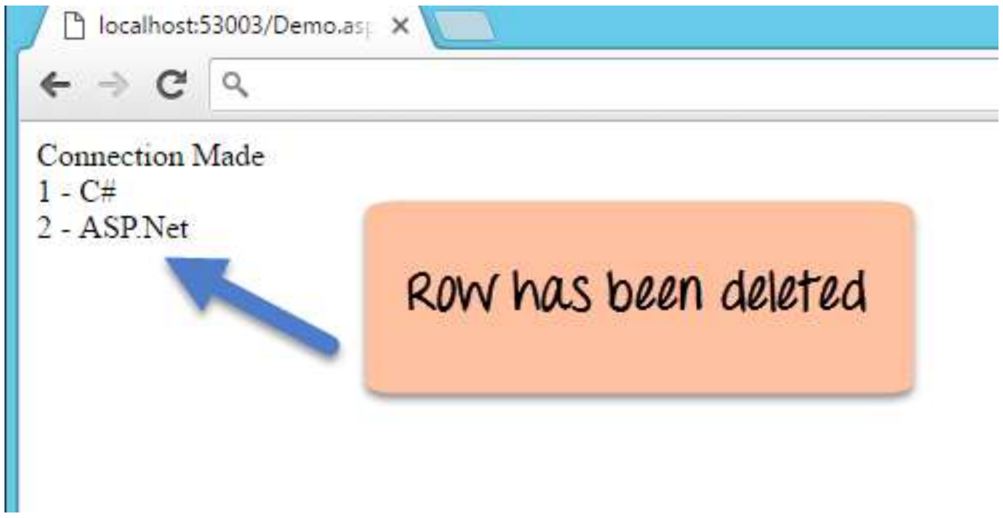
```
                    SqlCommand sqlquery;
                    SqlDataReader dataReader;
                    string Output="";
                    sql = "Select TutorialID,TutorialName from demotb";

                    sqlquery = new SqlCommand(sql, cnn);

                    dataReader = sqlquery.ExecuteReader();

                    while (dataReader.Read())
                    {
        Output = Output + dataReader.GetValue(0) + " - " + dataReader.GetValue(1) + "<br>";

                    }

                    Response.Write(Output);
                    command.Dispose();
                    cnn.Close();
```

Code:-

```
namespace DemoApplication
{
        public partial class Demo System.Web.UI.Page
    {
          protected void Page_Load(object sender, EventArgs e)
          {
                SqlCommand sqlquery;
                SqlDataReader dataReader;
                String Output ="";
                sql = "Select TutorialID,TutorialName from demotb";

                sqlquery = new SqlCommand(sql, cnn);

                dataReader = command.ExecuteReader();

                while(dataReader.Read())
                {
                  Output = Output + dataReader.GetValue(0) + "-" +
dataReader.GetValue(1) + "</br>";
                }

                Response.Write(Output);
                dataReader.Close();
                command.dispose();
                conn.Close();
          }
        }
}
```

When the above code is set, and the project is executed using Visual Studio, you will get the below output.

Output:-



```
Connection Made
1 - C#
2 - ASP.Net
```

Connecting Asp.net Controls to Data

We have seen how we can use ASP.Net commands such as SQLCommand and SQLReader to fetch data from a database. We also saw how we can read each row of the table display it on the web page.

There are methods available to link controls directly to different fields in the table. At the moment, only the below controls can be bound to an ASP.Net application

1. CheckboxList
2. RadioButtonList
3. DropDownlist
4. Listbox

So let's see an example of how we can use control binding in ASP.Net. Here we will take a listbox example. Let's say we have the following data in our database.

TutorialID	TutorialName
1	C#
2	ASP.Net
3	VB.Net complete

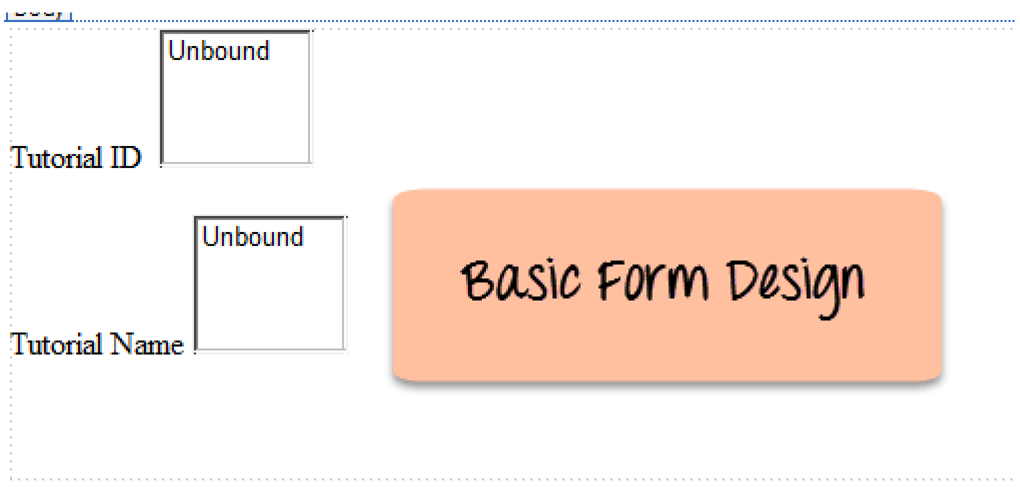
Let's use the Listbox control and see how it can automatically pick up the data from our Demotb table.

Let's follow the below-mentioned steps to achieve this.

Step 1) Construct the basic web form. From the toolbox in Visual Studio, drag and drop 2 components- labels and Listboxes. Then carry out the following substeps;

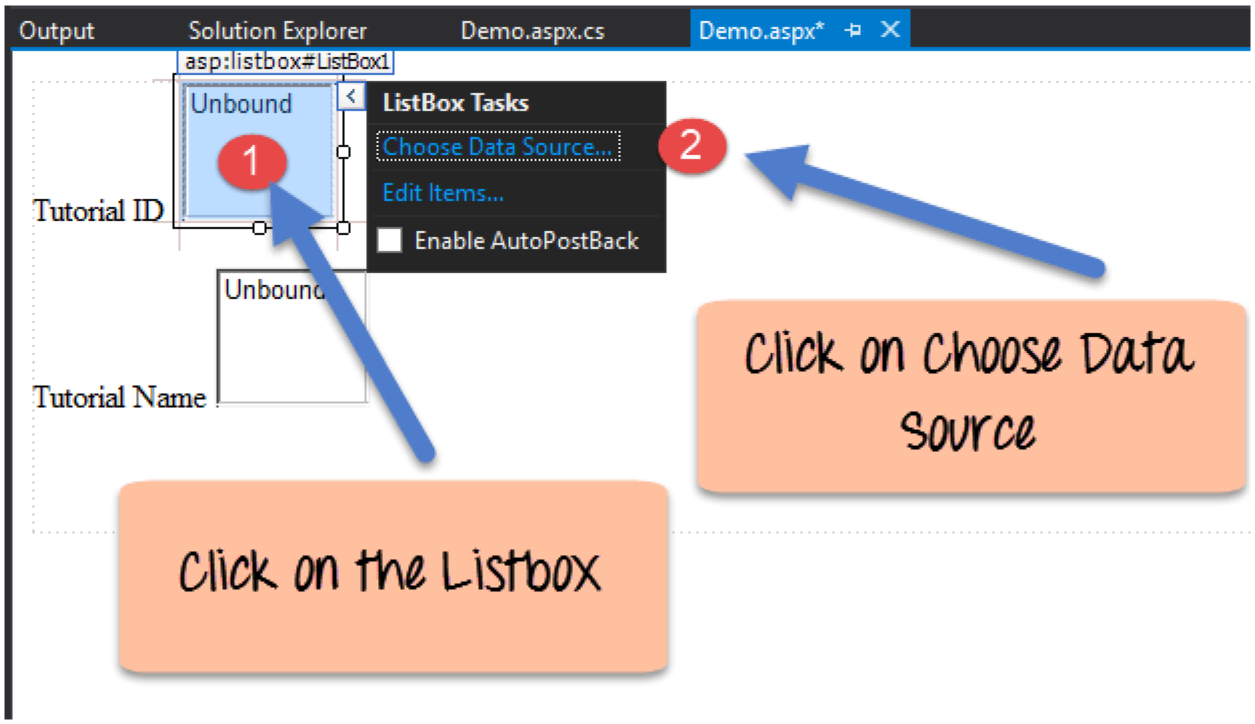
1. Put the text value of the first label as TutorialID
2. Put the text value of the second label as TutorialName

Below is how the form would look like once the above-mentioned steps are performed.

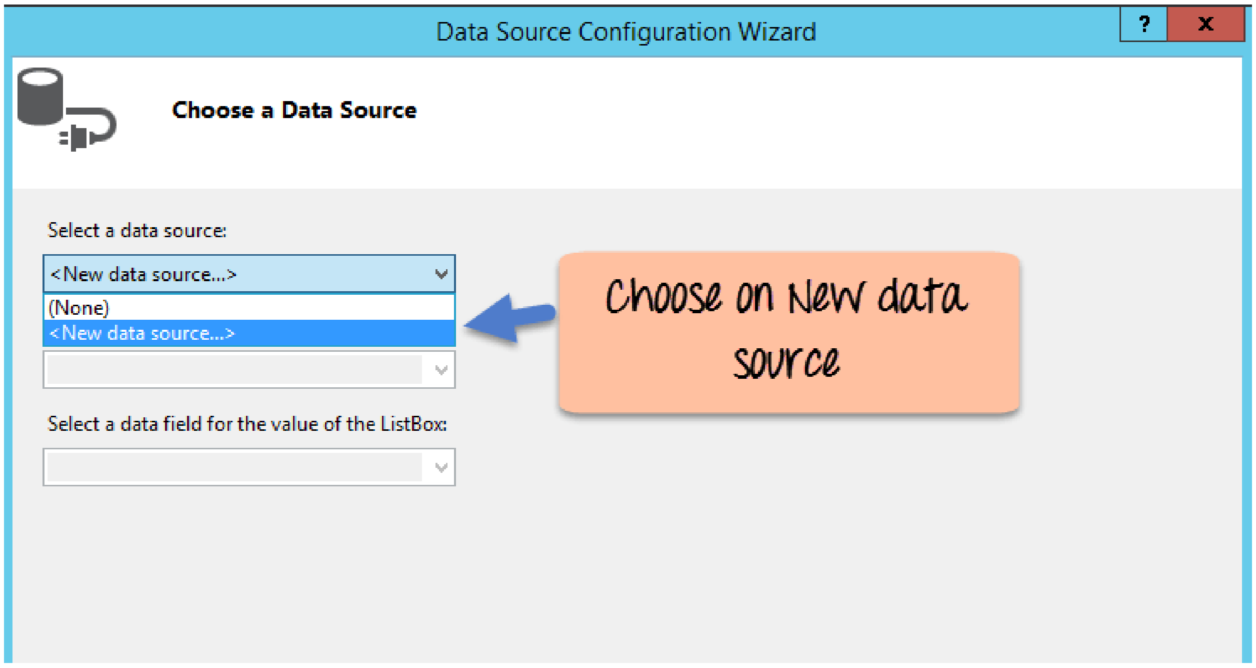


Step 2) The next step is to start connecting each listbox to the database table.

1. First, click on the Listbox for Tutorial ID. This will bring up another dialog box to the side of the control.
2. From the dialog box, we need to click on the option of Choose Data source.

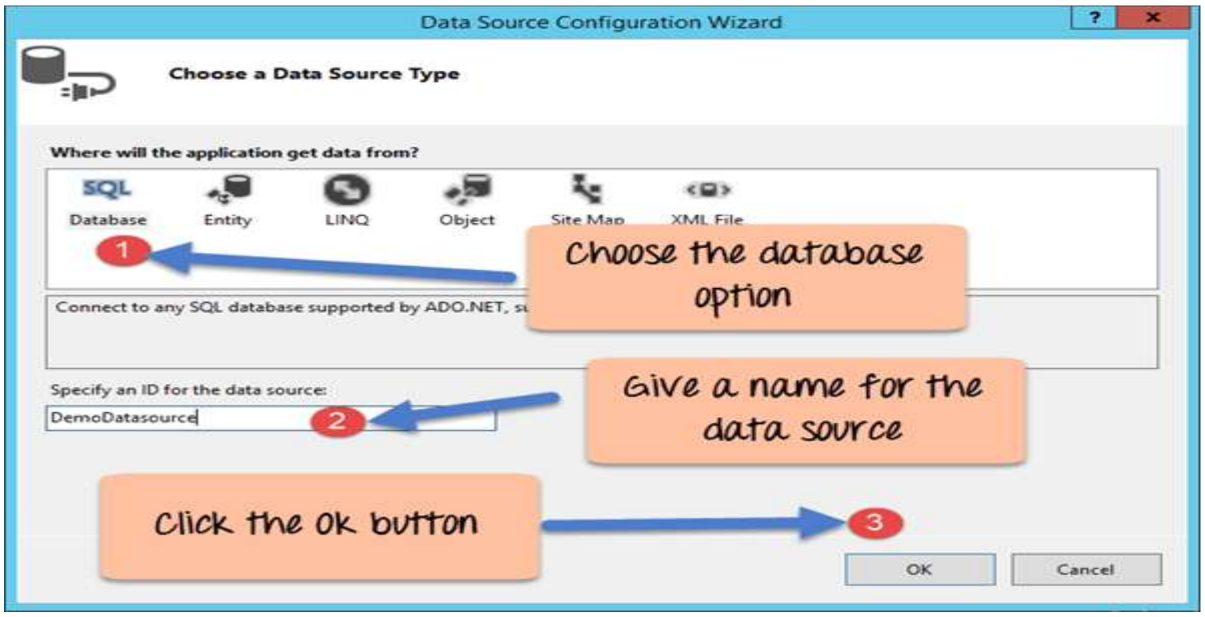


Step 3) You will then be presented with a dialog box. This can be used to create a new data source. The data source will represent a connection to the database. Choose the option of 'New data source'.

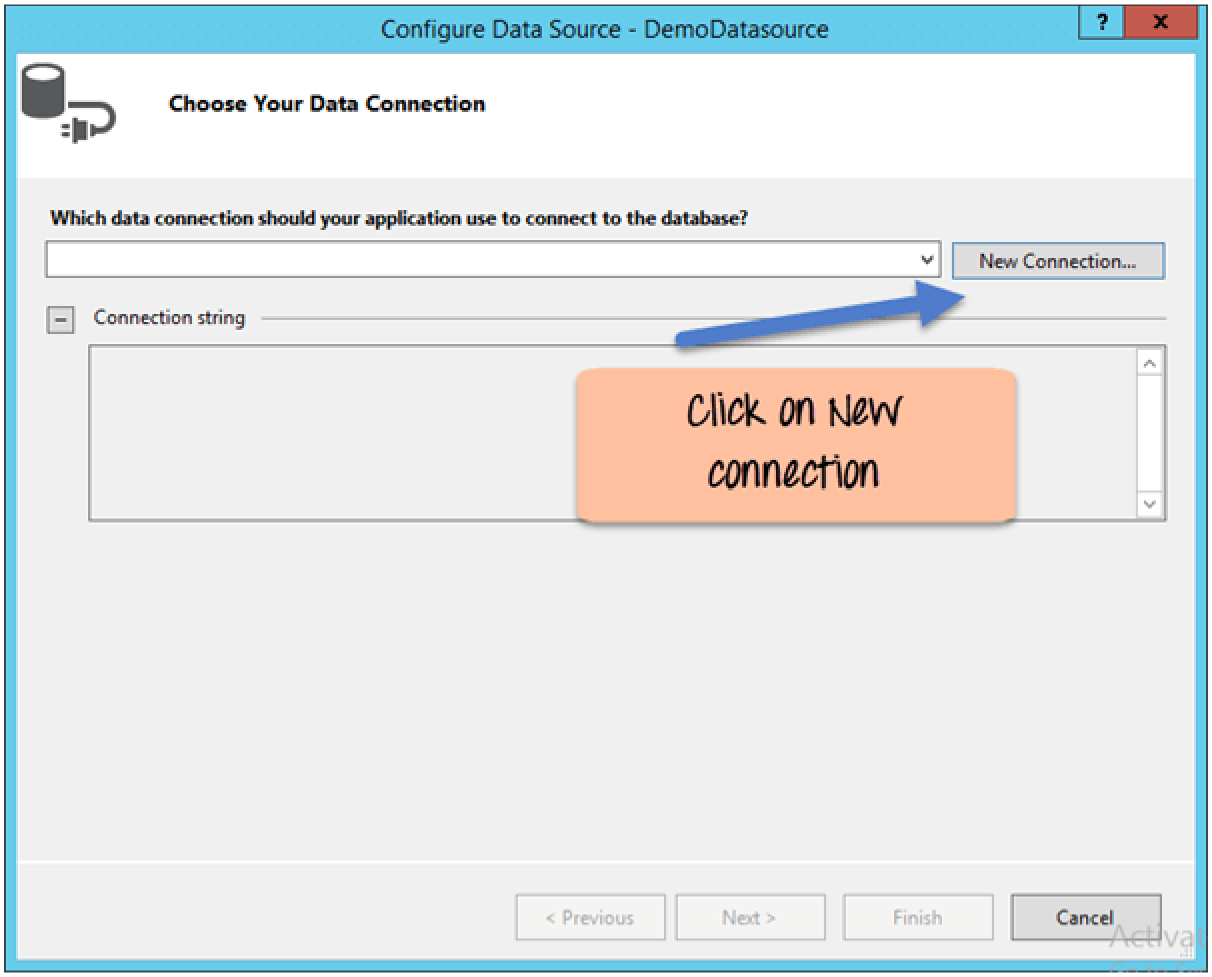

Data Source Configuration Wizard
Choose a Data Source
Select a data source:
<New data source...>
(None)
<New data source...>
Select a data field for the value of the ListBox:
Choose on New data source

Step 4) The below screen will be prompted after choosing the new data source in the last step. Here we need to mention the type of data source we want to create.

1. Choose the database option to work with an SQL Server database.
2. Now we need to give a name to our data source. Here we are giving it a name of DemoDataSource.
3. Finally, we click the 'OK' button to proceed to the next screen.

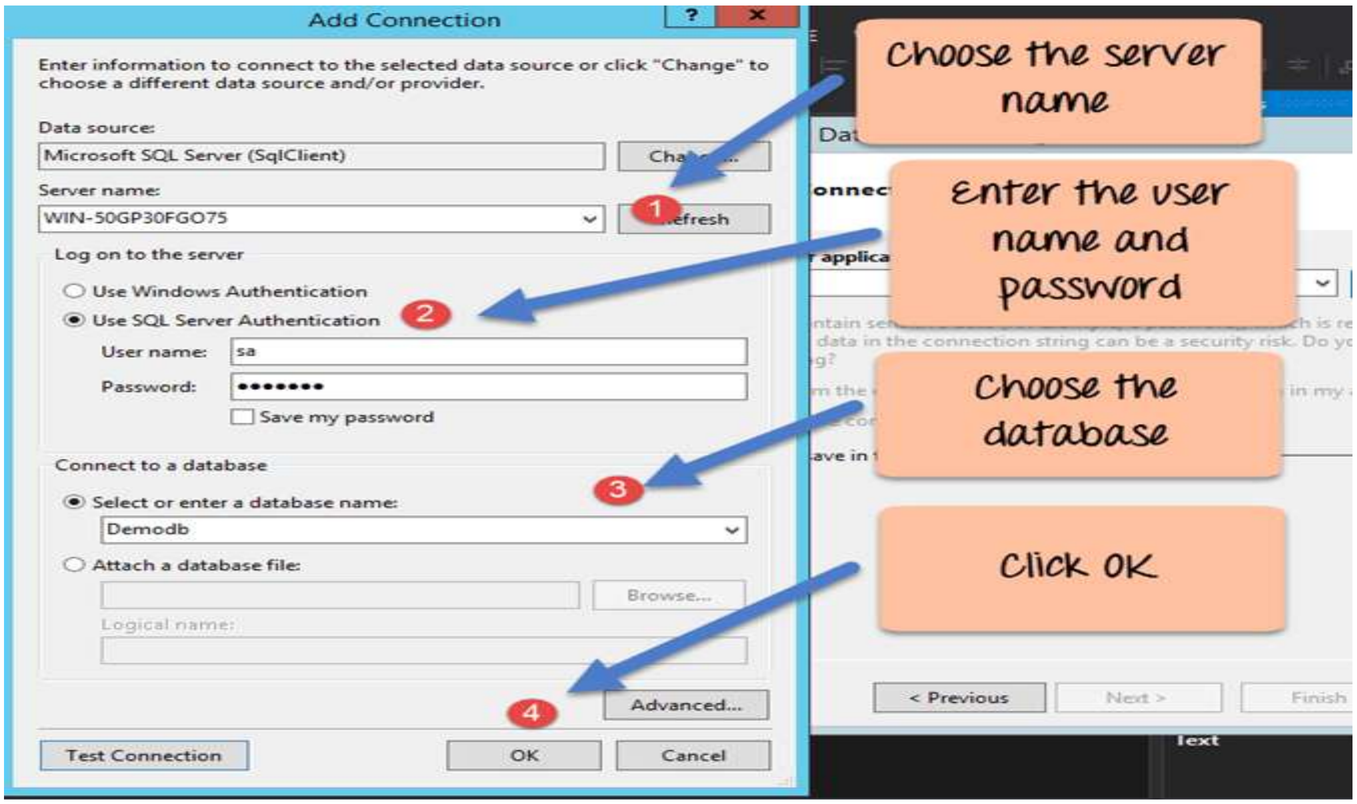

Data Source Configuration Wizard
Choose a Data Source Type
Where will the application get data from?
SQL Database
Entity
LINQ
Object
Site Map
XML File
Choose the database option
Connect to any SQL database supported by ADO.NET, s
Specify an ID for the data source:
DemoDatasource
Give a name for the data source
Click the Ok button
OK
Cancel

Step 5) Now we need to create a connection to our database. In the next screen, click on the New Connection button


Configure Data Source - DemoDatasource
Choose Your Data Connection
Which data connection should your application use to connect to the database?
New Connection...
Connection string
Click on New connection
< Previous
Next >
Finish
Cancel

Step 6) Next you need to add the credentials to connect to the database.

1. Choose the server name on which the SQL Server resides
2. Enter the user id and password to connect to the database
3. Choose the database as 'demotb'
4. Click the 'OK' button.



Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.

Data source: Microsoft SQL Server (SqlClient)

Server name: WIN-50GP30FGO75

Log on to the server

- Use Windows Authentication
- [✓ Use SQL Server Authentication]

User name: sa

Password: •••••••

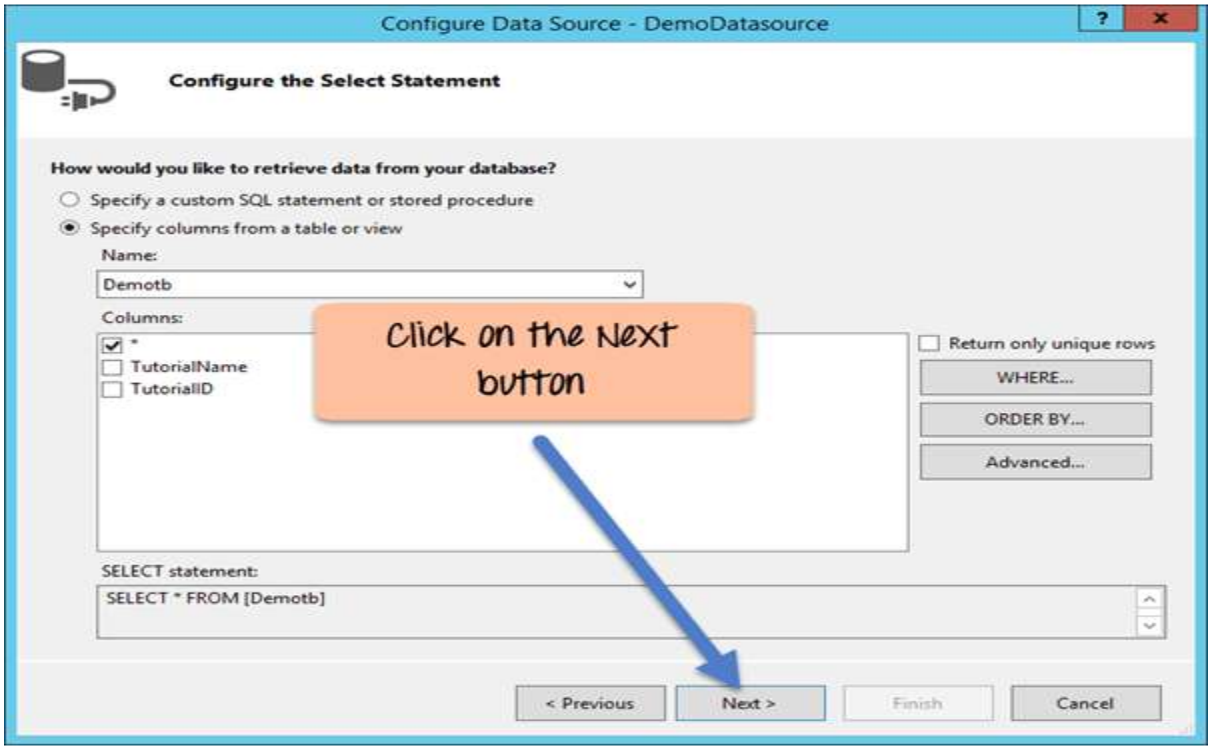
Save my password

Connect to a database

- [✓ Select or enter a database name:] Demodb
- Attach a database file:

Logical name:

Step 7) On the next screen, you will be able to see the Demotb table. Just click on the Next button to accept the default setting.



How would you like to retrieve data from your database?

- Specify a custom SQL statement or stored procedure
- [✓ Specify columns from a table or view]

Name: Demotb

Columns:

- [✓ *]
- TutorialName
- TutorialID

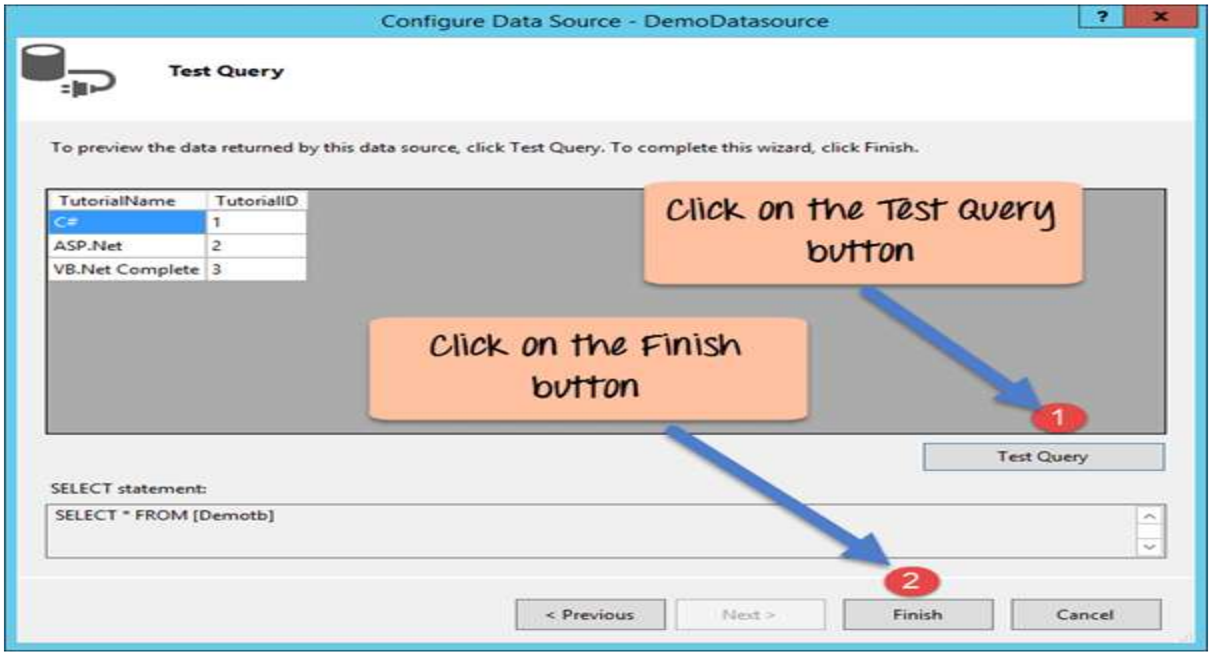
Return only unique rows

SELECT statement:

```
SELECT * FROM [Demotb]
```

Step 8) You will now be able to test the connection on the next screen.

1. Click on the Test Query button to just see if you are able to get the values from the table
2. Click the Finish button to complete the wizard.


Configure Data Source - DemoDatasource
Test Query
Click on the Test Query button
Click on the Finish button
1
2
Test Query
< Previous
Next >
Finish
Cancel

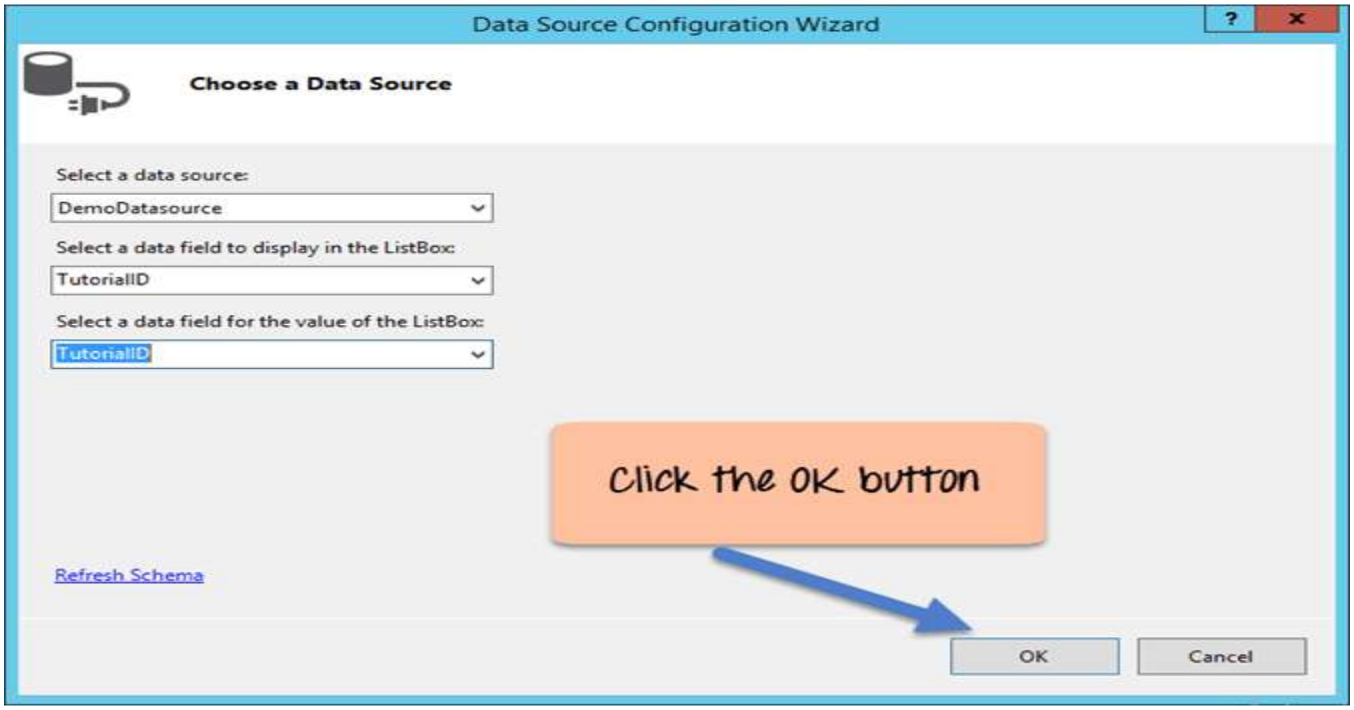
To preview the data returned by this data source, click Test Query. To complete this wizard, click Finish.

TutorialName	TutorialID
C#	1
ASP.Net	2
VB.Net Complete	3

SELECT statement:

```
SELECT * FROM [Demotb]
```

Step 9) Now on the final screen, you can click the 'OK' button. This will now bind the TutorialID listbox to the TutorialID field name in the 'demotb' table.



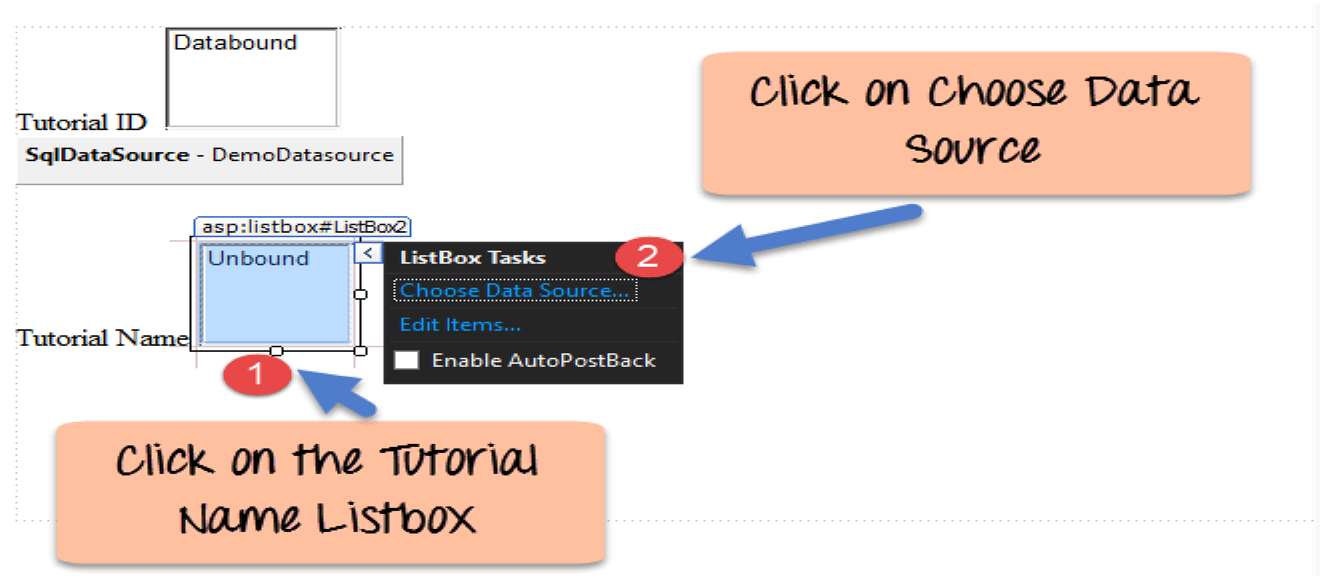
Select a data source: DemoDatasource

Select a data field to display in the ListBox: TutorialID

Select a data field for the value of the ListBox: TutorialID

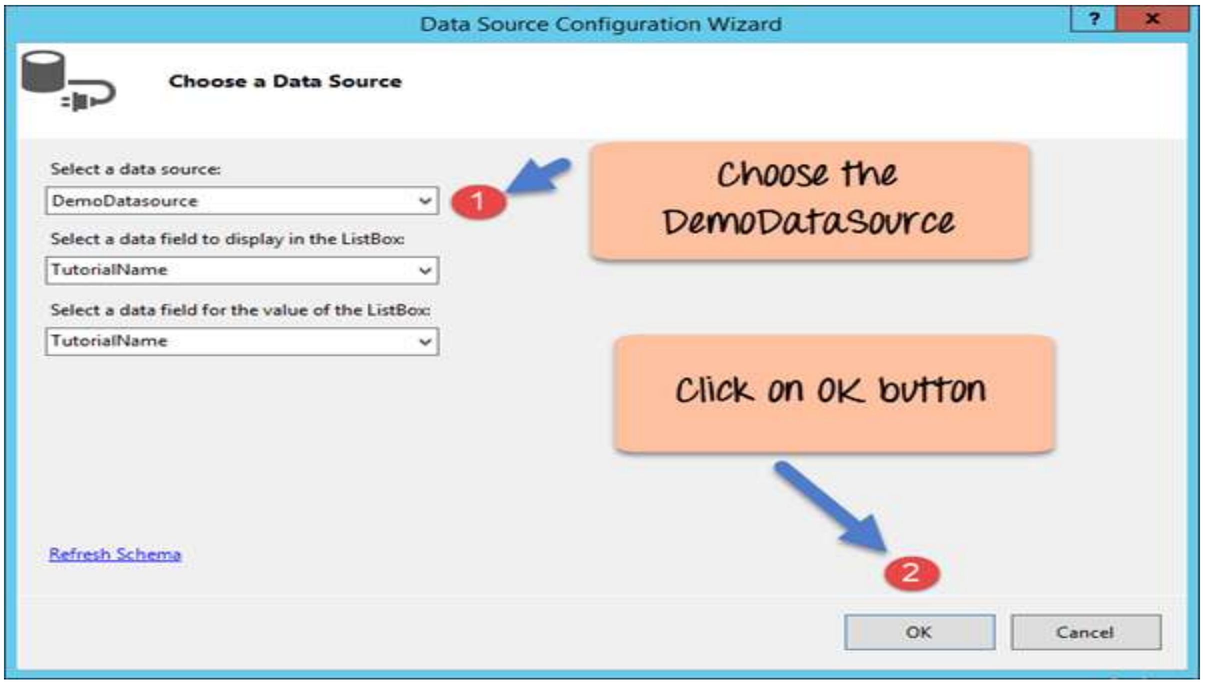
Step 10) Now it's time to bind the Tutorial Name listbox to the Tutorial Name field.

1. First, click on the Tutorial Name Listbox.
2. Next, Choose from Data Source in the dialog box which appears at the side of the Listbox.



Step 11) You will already see the DemoDataSource when choosing the Data Source in the next screen.

1. Choose the DemoData Source
2. Click on the OK button.



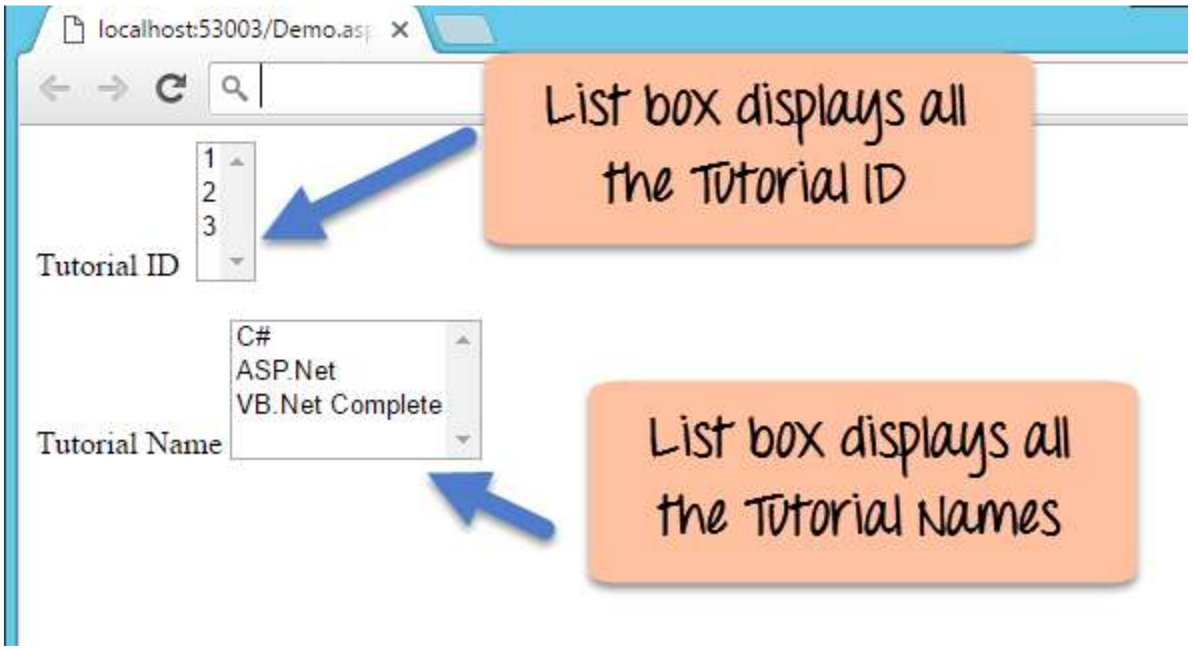
Select a data source: DemoDatasource

Select a data field to display in the ListBox: TutorialName

Select a data field for the value of the ListBox: TutorialName

If all the above steps are executed as shown, you will get the below-mentioned output.

Output:-


localhost:53003/Demo.asp
Tutorial ID
1
2
3
Tutorial Name
C#
ASP.Net
VB.Net Complete
List box displays all the Tutorial ID
List box displays all the Tutorial Names

From the output, you can see that the listboxes display the Tutorial ID and Tutorial Names respectively

Experiment No:-4

Aim:- Create a web service and use it in web site.

Objectives:-
To understand Increase sales.
To understand Becoming an authoritative resource.
Goal: Improve interaction with existing and potential customers.
To understand Build your brand.
Theory:

Create your first Restful web service in ASP.NET

Web services can be created in a variety of languages. Many integrated development environments can be used to create REST-based services.

In this example, we are going to create our REST application in .Net using Visual Studio. In our example, for Restful web services we are going to emulate the following example.

We are going to have a Restful web service which will work on the below set of data.

The below set of data represents an example of having a company which exposes the Tutorial's they have based on the Tutorialid.

Tutorialid	**TutorialName**
0	Arrays
1	Queues
2	Stacks

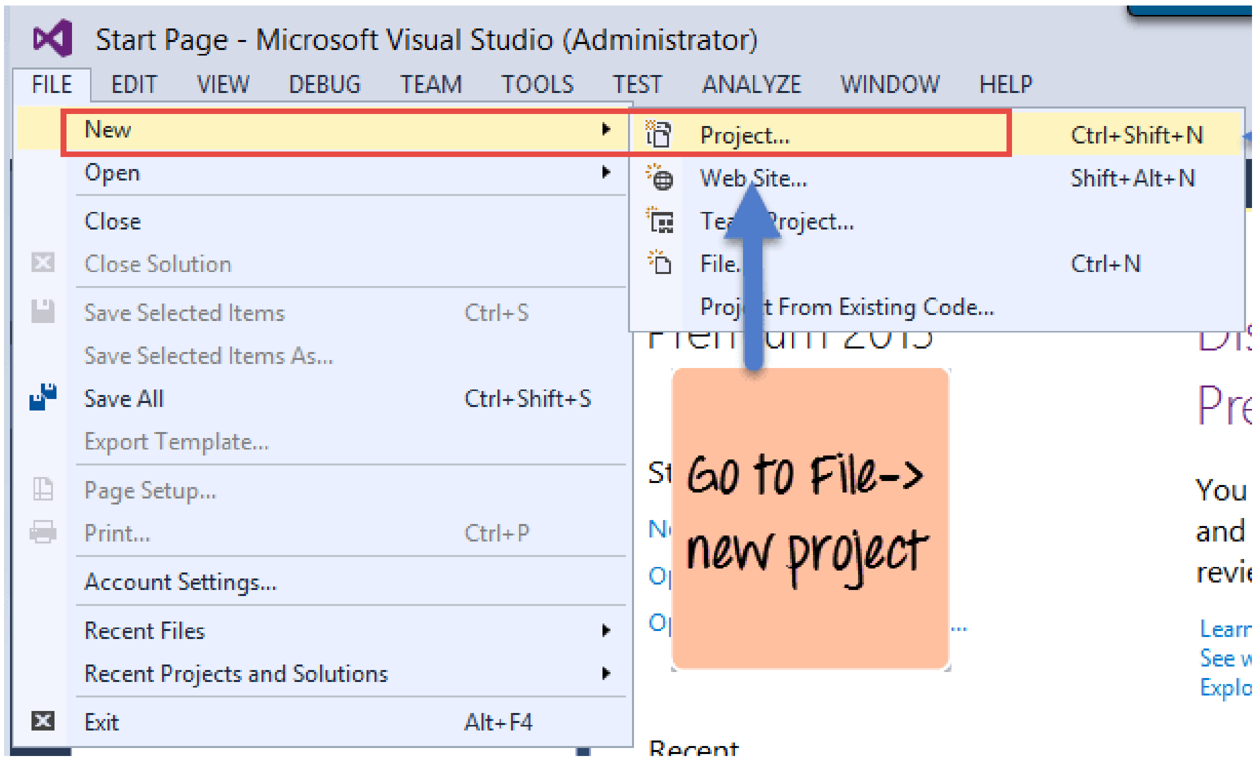
In our example, we are going to implement the below Restful Verbs.

1. GET Tutorial – When a client invokes this Restful API, they will be given the entire set of Tutorials available from the web service.
2. GET Tutorial/Tutorialid - When a client invokes this Restful API, they will be given the Tutorial name based on the Tutorialid sent by the client.
3. POST Tutorial/Tutorialname - When a client invokes this Restful API, the client will submit a request to insert a Tutorialname. The web service will then add the submitted Tutorial name to the collection.

4. DELETE Tutorial/Tutorialid- When a client invokes this Restful API, the client will submit a request to delete a Tutorialname based on the Tutorialid. The web service will then delete the submitted Tutorial name from the collection.

Let's follow the below steps in creating our first RESTful web services, which carries out the above implementation.

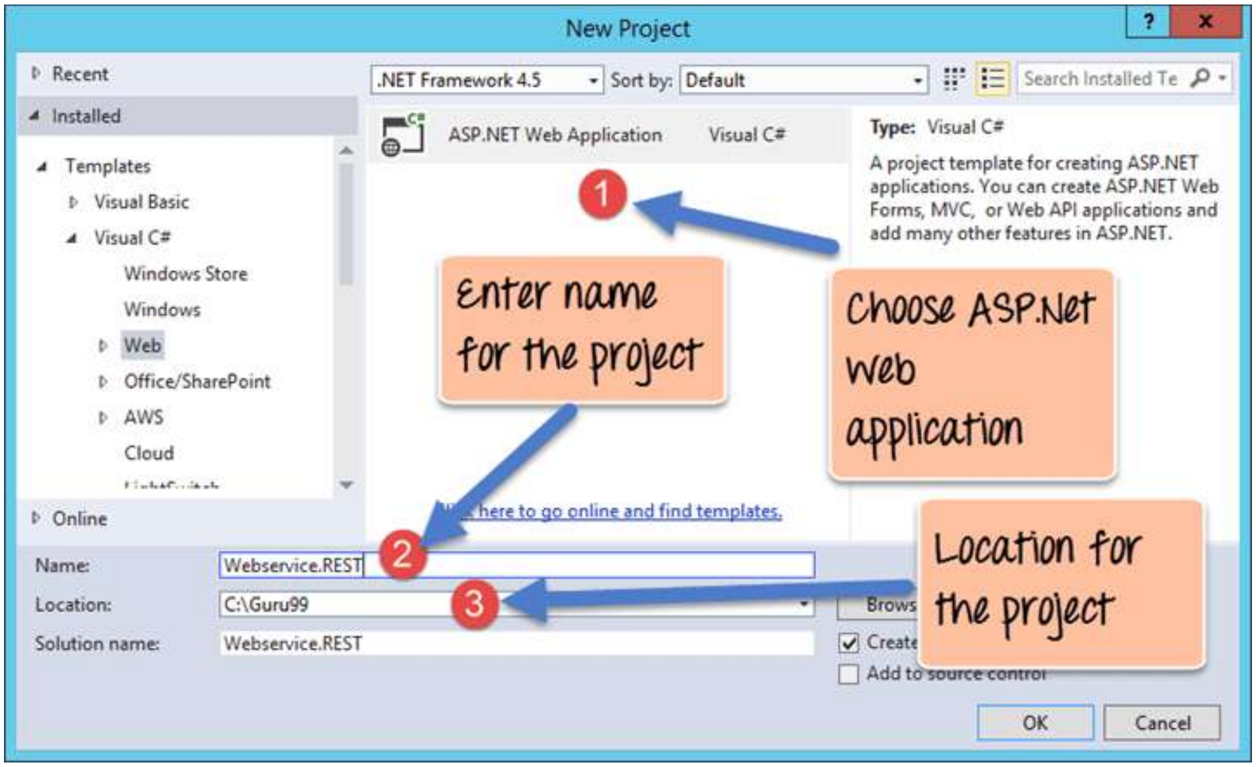
Step 1) The first step is to create an empty Asp.Net Web application. From Visual Studio 2013, click on the menu option File->New project.



Once you click on the New Project option, Visual Studio will then give you another dialog box for choosing the type of project and to give the necessary details of the project. This is explained in the next step

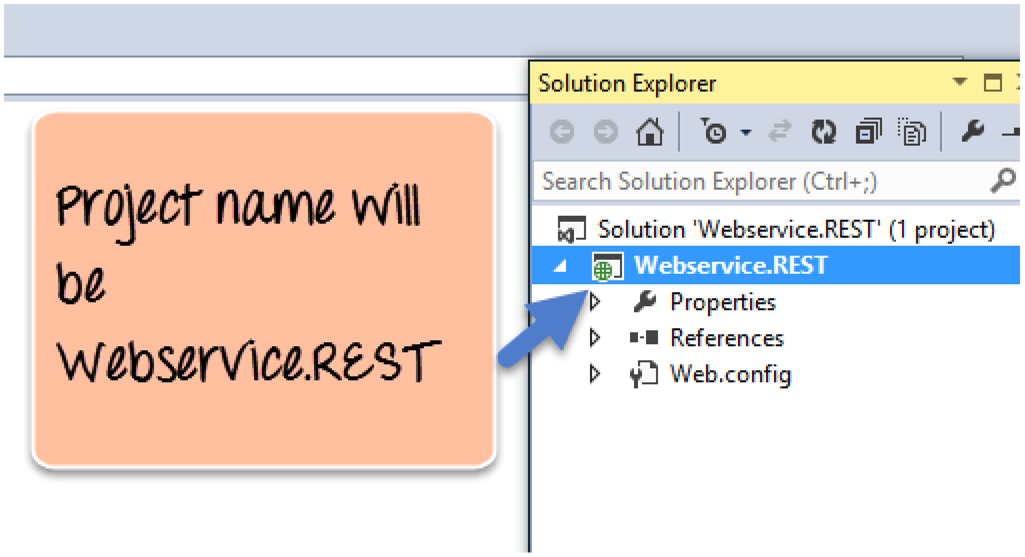
Step 2) In this step,

1. Ensure to first choose the C# web template of ASP.NET Web application. The project has to be of this type in order to create web services project. By choosing this options, Visual Studio will then carry out the necessary steps to add required files which are required by any web-based application.
2. Give a name for your project which in our case has been given as "Webservice.REST".
3. Then ensure to give a location, where the project files will be stored.



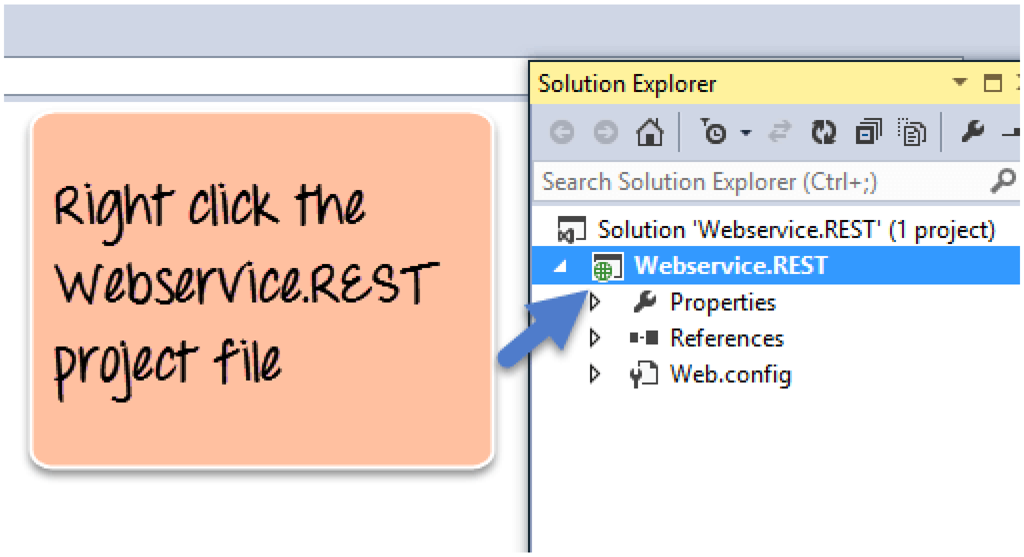
Name: Webservice.REST
Location: C:\Guru99
Solution name: Webservice.REST

Once done you will see the project file created in your solution explorer in Visual Studio 2013.


Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'Webservice.REST' (1 project)
Webservice.REST
Properties
References
Web.config
Project name will be Webservice.REST

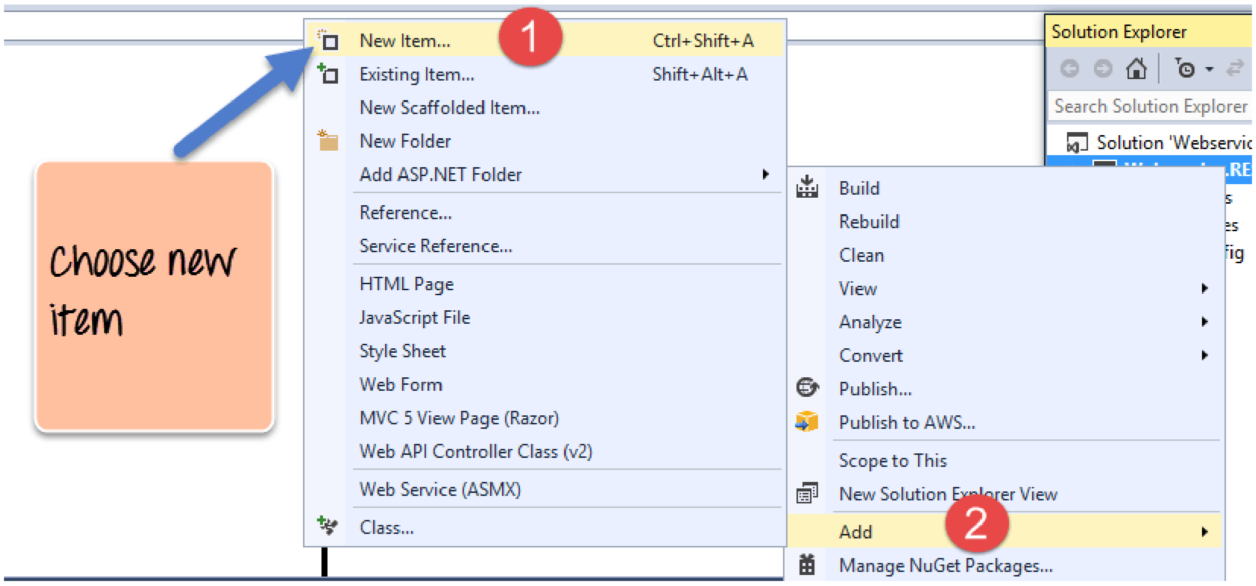
Step 3) The next step is to create the web service file which is going to have the RESTful web service

1. First Right-click on the project file as shown below


Right click the Webservice.REST project file
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'Webservice.REST' (1 project)
Webservice.REST
Properties
References
Web.config

2. In this step,

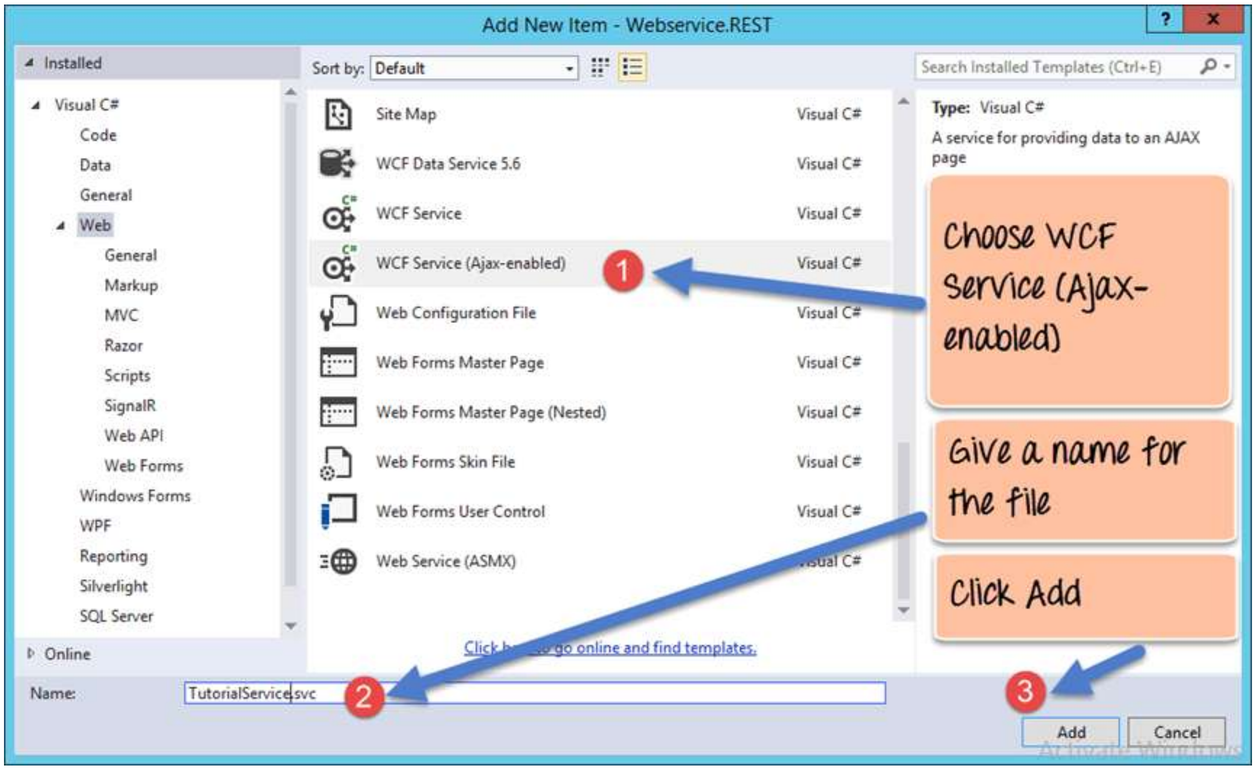
1. Right-click on the project file
2. Choose the option "Add->new item."



In the dialog box which appears, you need to perform the following

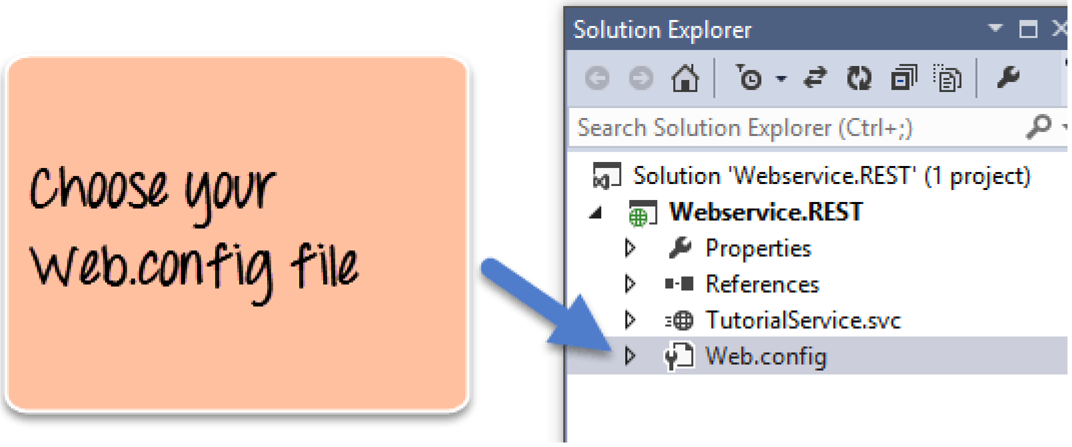
1. Choose the option of WCF Service (Ajax-enabled) – Choose a file of this type, it causes the Visual studio to add some basic code which helps one create a RESTful web service. WCF stands for Windows Communication Foundation. WCF is a library for applications of various platforms or the same platform to communicate over the various protocols such as TCP, HTTP, HTTPS. Ajax basically is Asynchronous JavaScript and XML. AJAX allows web pages to be updated asynchronously by exchanging small amounts of data with the server behind the scenes.
2. Next give a name for the service which is TutorialService in our case.

3. Finally, click the Add button to add the service to the solution.

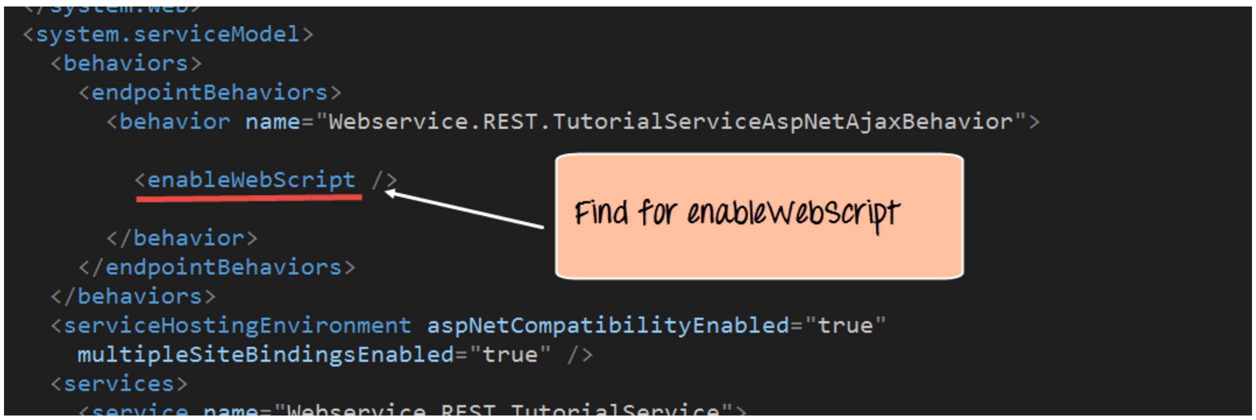

Add New Item - Webservice.REST
Installed
Visual C#
Code
Data
General
Web
General
Markup
MVC
Razor
Scripts
SignalR
Web API
Web Forms
Windows Forms
WPF
Reporting
Silverlight
SQL Server
Online
Sort by: Default
Search Installed Templates (Ctrl+E)
Site Map Visual C#
WCF Data Service 5.6 Visual C#
WCF Service Visual C#
WCF Service (Ajax-enabled) Visual C#
Web Configuration File Visual C#
Web Forms Master Page Visual C#
Web Forms Master Page (Nested) Visual C#
Web Forms Skin File Visual C#
Web Forms User Control Visual C#
Web Service (ASMX) Visual C#
Type: Visual C#
A service for providing data to an AJAX page
Choose WCF Service (Ajax-enabled)
Give a name for the file
Click Add
Click here to go online and find templates.
Name: TutorialService.svc
1
2
3
Add
Cancel

Step 4) The next step is to actually make a configuration change to enable this project to complete work with RESTful web services. This requires to make a change to the file called **Web.config**. This file appears in the same window as the Webservice project file. The file Web.config contains all configurations that make the web application work as it should. The change being made actually allows the application to send and receive data as a pure RESTful web service.

1. Click on the Web.config file to open the code

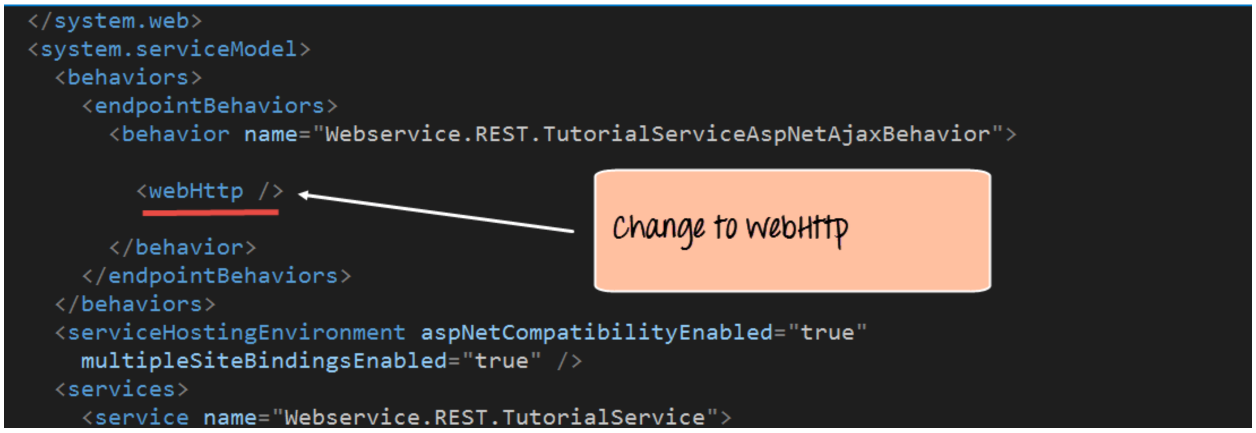

Choose your Web.config file
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'Webservice.REST' (1 project)
Webservice.REST
Properties
References
TutorialService.svc
Web.config

2. Find for the line <enableWebScript>



```
<system.serviceModel>
  <behaviors>
    <endpointBehaviors>
      <behavior name="Webservice.REST.TutorialServiceAspNetAjaxBehavior">

        <enableWebScript />

      </behavior>
    </endpointBehaviors>
  </behaviors>
  <serviceHostingEnvironment aspNetCompatibilityEnabled="true"
    multipleSiteBindingsEnabled="true" />
  <services>
```

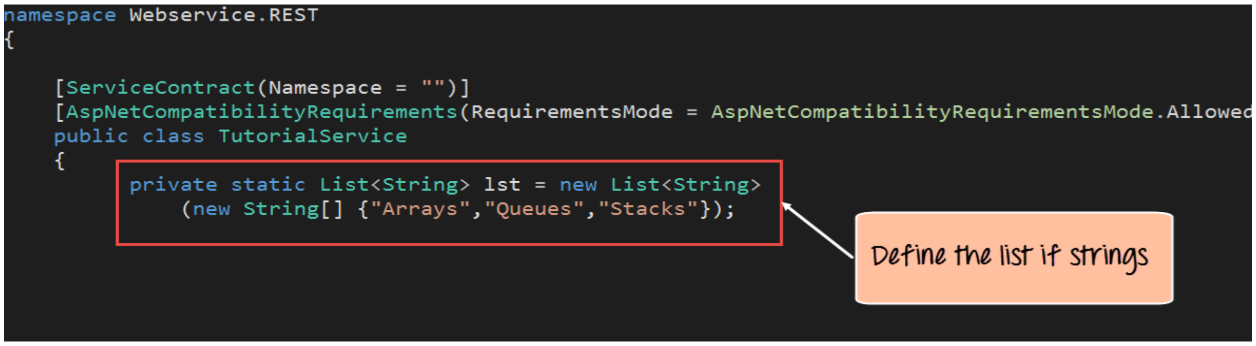
3. Change the line to <webHttp>



```
</system.web>
<system.serviceModel>
  <behaviors>
    <endpointBehaviors>
      <behavior name="Webservice.REST.TutorialServiceAspNetAjaxBehavior">

        <webHttp />

      </behavior>
    </endpointBehaviors>
  </behaviors>
  <serviceHostingEnvironment aspNetCompatibilityEnabled="true"
    multipleSiteBindingsEnabled="true" />
  <services>
    <service name="Webservice.REST.TutorialService">
```

Step 5) The next step is to add our code for implementation. All of the below-mentioned code has to be written in the TutorialService.svc file

1. The first bit is to add code to represent our data which will be used in our program. So we are going to have a list of string variables with values of "Arrays", "Queues" and "Stacks". This will represent the tutorials name available through our hosting web service.



```
namespace Webservice.REST
{

    [ServiceContract(Namespace = "")]
    [AspNetCompatibilityRequirements(RequirementsMode = AspNetCompatibilityRequirementsMode.Allowed
    public class TutorialService
    {
        private static List<String> lst = new List<String>
            (new String[] {"Arrays","Queues","Stacks"});
```

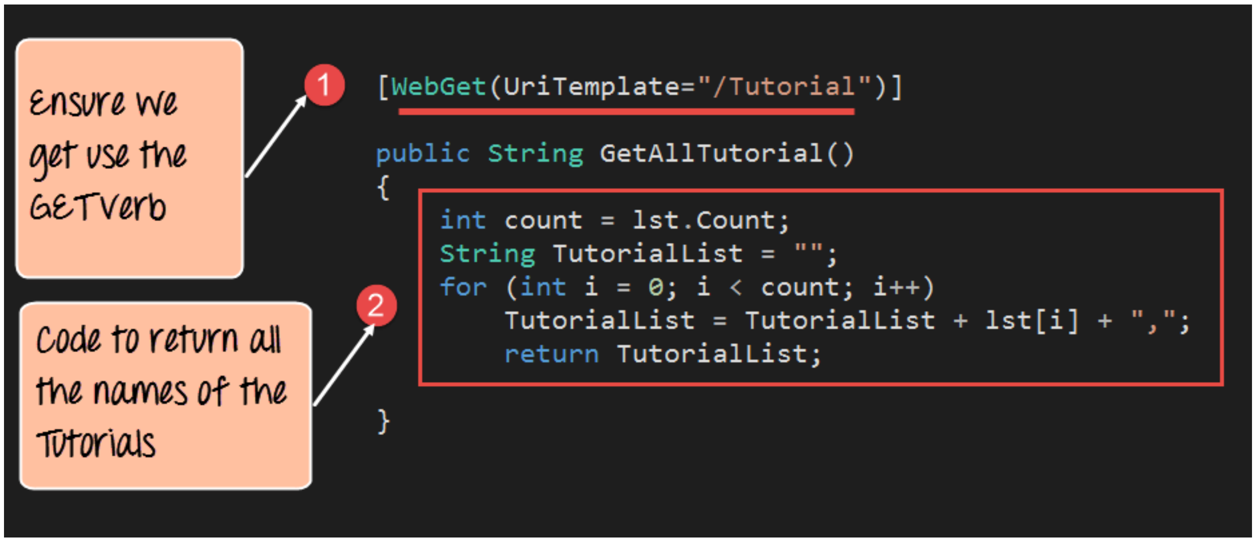
Code:-

```
namespace Webservice.REST
{
		[ServiceContnact(Namespace = "")]
		[AspNetCompatibilityRequirements(RequirementsMode                    =
AspNetCompatibilityRequirementsMode.Allowed
		public class TutorialService
		{
				private static List 1st = new List
				(new String[] {"Arrays","Queues","Stacks"});
```

Step 6) Next we will define the code for our GET method. This code will also reside in the same TutorialService.svc file. This code will run whenever we call the service from our browser.

The below method will be used to fulfill the below-mentioned scenario

- If a user wants a list of all Tutorials available, then the below code would need to be written to accomplish this.


Ensure we get use the GETVerb
1
Code to return all the names of the Tutorials
2

```
[WebGet(UriTemplate="/Tutorial")]

public String GetAllTutorial()
{
    int count = lst.Count;
    String TutorialList = "";
    for (int i = 0; i < count; i++)
        TutorialList = TutorialList + lst[i] + ",";
        return TutorialList;

}
```

Code:-

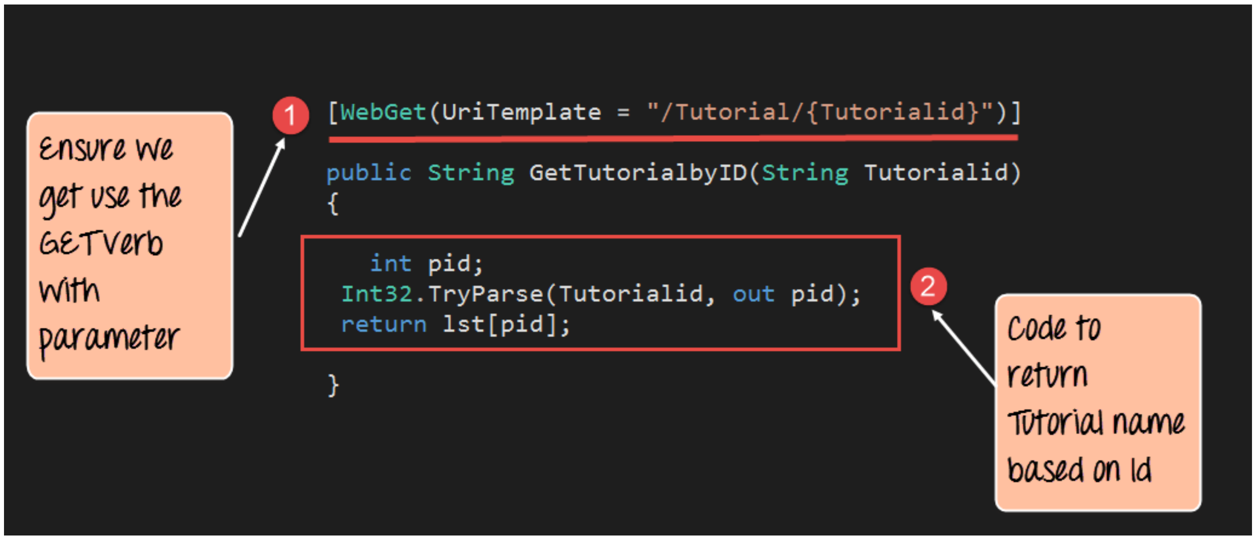
```
[WebGet(UriTemplate="/Tutorial")]

public String GetAllTutorial()
{
		int count = 1st.Count;
		String TutorialList = "";
		for (int i = 0; i < count; i++)
```

```
            TutorialList = TutorialList + lst[i] + ",";
            return TutorialList;
}
```

Code Explanation:-

1. The first line of code is the most important. It is used to define how we can call this method via a URL. So if the link to our web service is **http://localhost:52645/TutorialService.svc** and if we append the '/Tutorial' to the URL as **http://localhost:52645/TutorialService.svc/Tutorial** , the above code will be invoked. The attribute of 'WebGet' is a parameter which allows this method to be a RESTful method so that it can be invoked via the GET verb.
2. This section of code is used to go through our list of strings in the 'lst' variable and return all of them to the calling program.

Step 7) The code below ensures that if a GET call is made to the Tutorial Service with a Tutorial id, then it would return the corresponding Tutorial Name based on the Tutorial id.


Ensure we get use the GETVerb with parameter
1
2
Code to return Tutorial name based on Id

```
[WebGet(UriTemplate = "/Tutorial/{Tutorialid}")]
public String GetTutorialbyID(String Tutorialid)
{
   int pid;
 Int32.TryParse(Tutorialid, out pid);
 return lst[pid];
}
```

Code:-

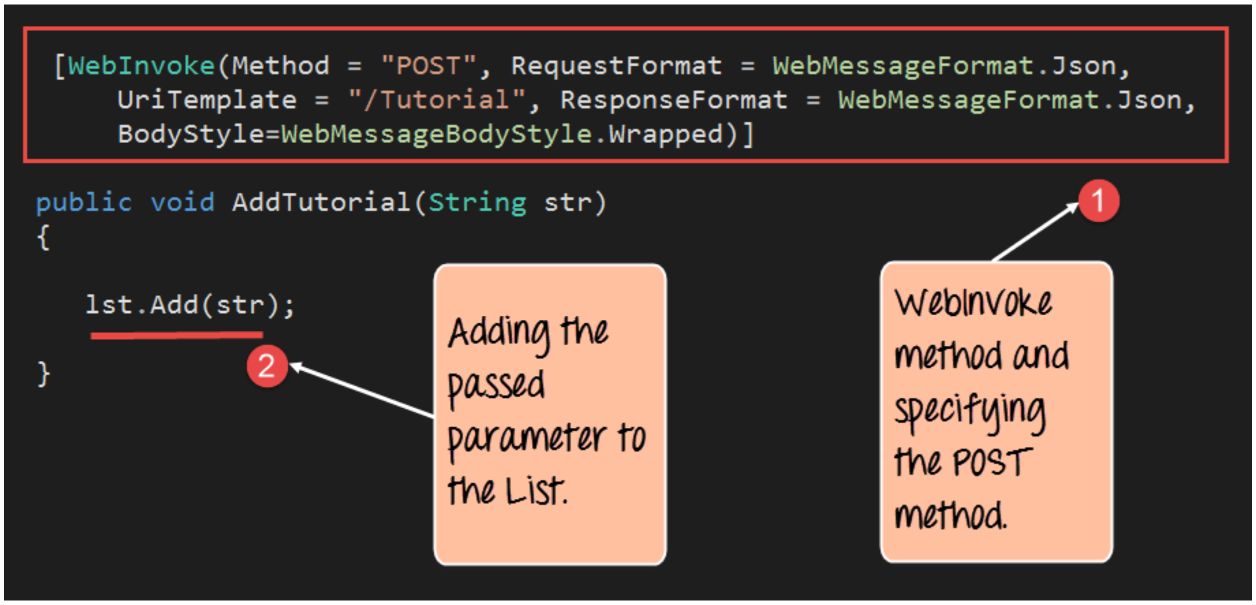
```
[WebGet (UriTemplate = "/Tutorial/{Tutorialid}")]

public String GetTutorialbyID(String Tutorialid)
{
            int pid;
            Int32.TryParse(Tutorialid, out pid);
            return lst[pid];
}
```

Code Explanation:-

1. The first line of code is the most important. It is used to define how we can call this method via a URL. So if the link to our web service is **http://localhost:52645/TutorialService.svc** and if we append the '/Tutorial/{Tutorialid}' to the URL, then we would be able to call the web service as **http://localhost:52645/TutorialService.svc/Tutorial/1** as an example. The web service would then need to return the Tutorial name which had the Tutorial id#1.
2. This section of code is used to return the "Tutorial name" which has the Tutorial id passed to the web method.

- By default, what needs to be remembered is that whatever is passed to the URL in the browser is a string.
- But you have to remember that the Index to our list has to be an integer, so we are adding the necessary code to first convert the Tutorialid to an Integer and then use it to access the index position in our list and
- Then return the value to the calling program accordingly.

Step 8) The next step is to write up the code for our POST method. This method will be invoked whenever we want to add a string value to our list of Tutorials via the POST method. For example, if you wanted to add the Tutorial name of "Software Testing" then you would need to use the POST method.



```
[WebInvoke(Method = "POST", RequestFormat = WebMessageFormat.Json,
    UriTemplate = "/Tutorial", ResponseFormat = WebMessageFormat.Json,
    BodyStyle=WebMessageBodyStyle.Wrapped)]

public void AddTutorial(String str)
{

   lst.Add(str);

}
```

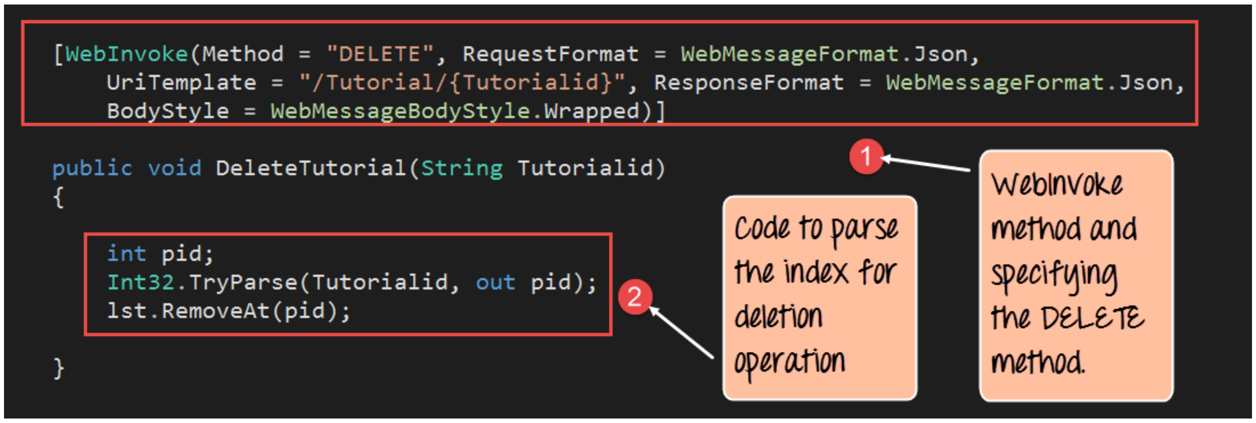
1
2
Adding the passed parameter to the List.
WebInvoke method and specifying the POST method.

Code Explanation:-

1. The first line is the 'WebInvoke' attribute which has been attached to our method. This allows the method to be invoked via the POST call. The RequestFormat and

ResponseFormat attribute have to be mentioned as JSON, since when posting values to a RESTFul web service, the values have to be in this format.
2. The second line of code is used to add the string value passed via the POST call to our existing list of Tutorial strings.

Step 9) Finally we are going to add our method to handle the DELETE operation. This method will be invoked whenever we want to delete an existing string value from our list of Tutorials via the DELETE method.



```
[WebInvoke(Method = "DELETE", RequestFormat = WebMessageFormat.Json,
    UriTemplate = "/Tutorial/{Tutorialid}", ResponseFormat = WebMessageFormat.Json,
    BodyStyle = WebMessageBodyStyle.Wrapped)]

public void DeleteTutorial(String Tutorialid)
{

    int pid;
    Int32.TryParse(Tutorialid, out pid);
    lst.RemoveAt(pid);

}
```

1
WebInvoke method and specifying the DELETE method.
2
Code to parse the index for deletion operation

Code:-

```
[WebInvoke(Method = "DELETE", RequestFormat = WebMessageFormat.Ison,
        UriTemplate = "/Tutorial/{Tutorialid}", ResponseFormat = WebMessageFormat.Json,
        BodyStyle = WebMessageBodyStyle.Wrapped)]

public void DeleteTutorial(String Tutorialid)
{
        int pid;
        Int32.TryParse(Tutorialid, out pid);
        1st.RemoveAt(pid);
}
```

Code Explanation:-

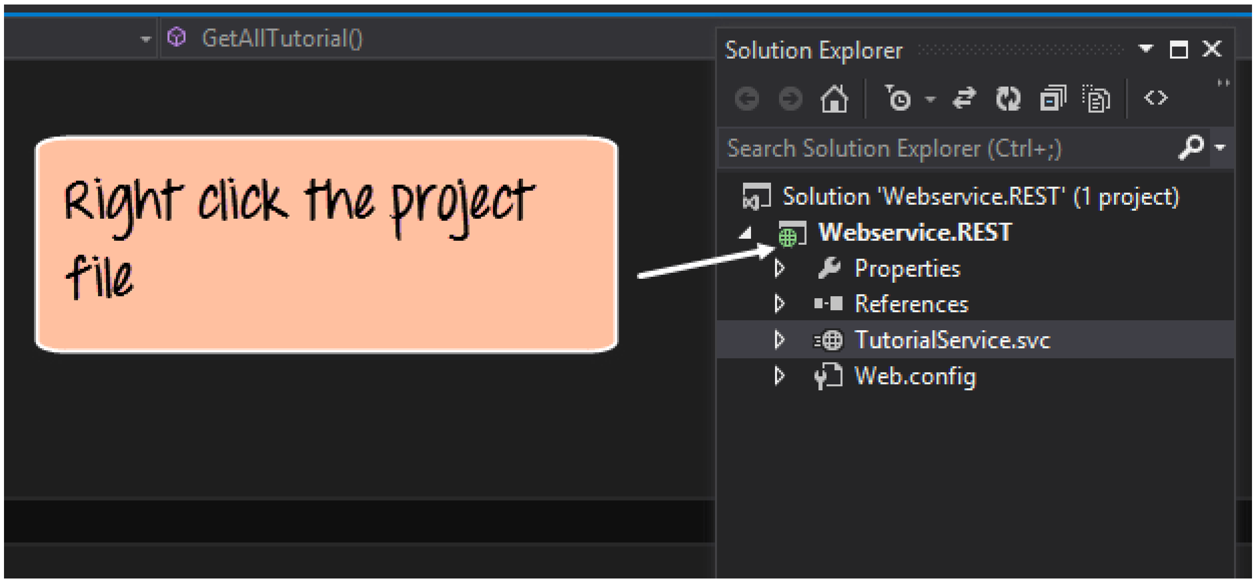
1. The first line is the 'WebInvoke' attribute which has been attached to our method. This allows the method to be invoked via the POST call. The RequestFormat and ResponseFormat attribute have to be mentioned as JSON, since when posting values to a RESTFul web service, the values have to be in this format. Note that the Method parameter is being set to "DELETE." This means that whenever we issue the DELETE verb, this method will be invoked.
2. The second line of code is used to take the Tutorialid sent via the DELETE call and subsequently delete that id from our list. (The **Int32** function in code is used to convert the Tutorial ID from a string variable to an integer).

Running your first Restful web service

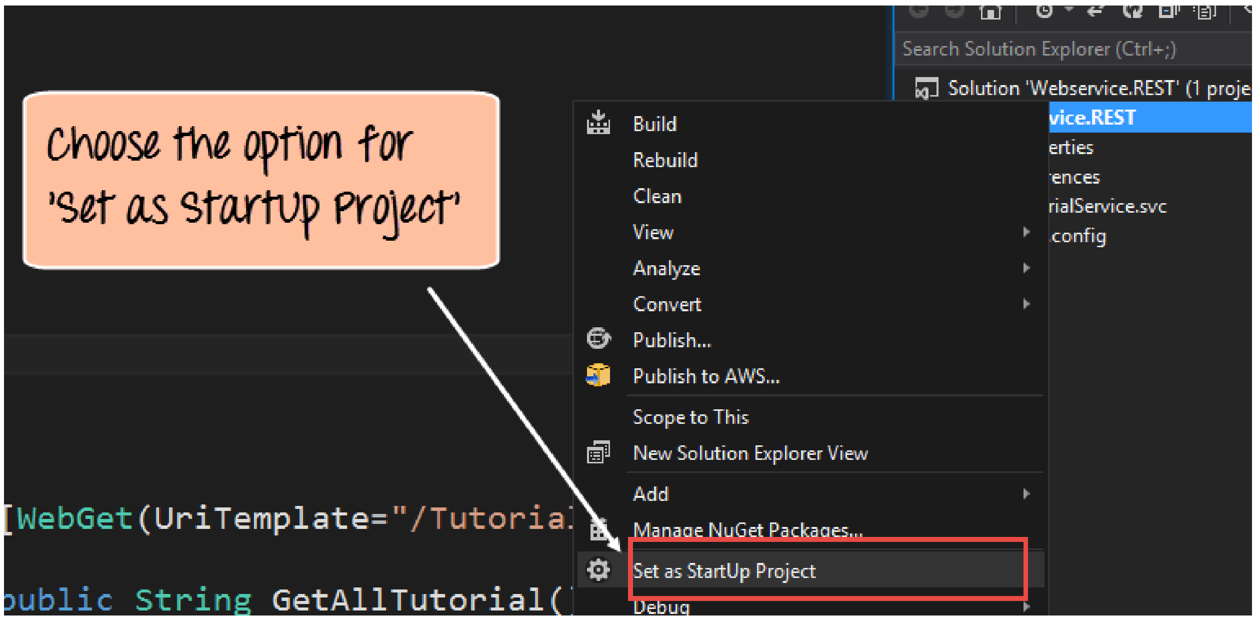
Now that we have created our entire web service in the above section. Let's see how we can run the Tutorial service so that it can be invoked from any client.

To run the web service, please follow the below steps

Step 1) Right click on the Project file – Webservice.REST

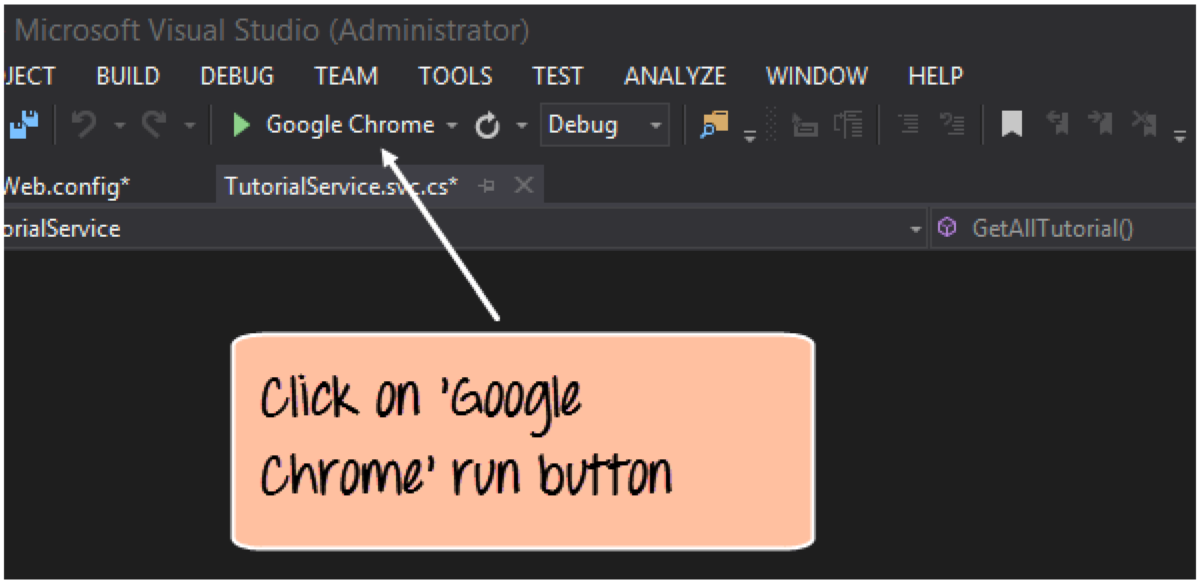

GetAllTutorial()
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'Webservice.REST' (1 project)
Webservice.REST
Properties
References
TutorialService.svc
Web.config
Right click the project file

Step 2) Choose the menu option 'Set as StartUp Project'. This will ensure that this project is run when Visual Studio runs the entire solution



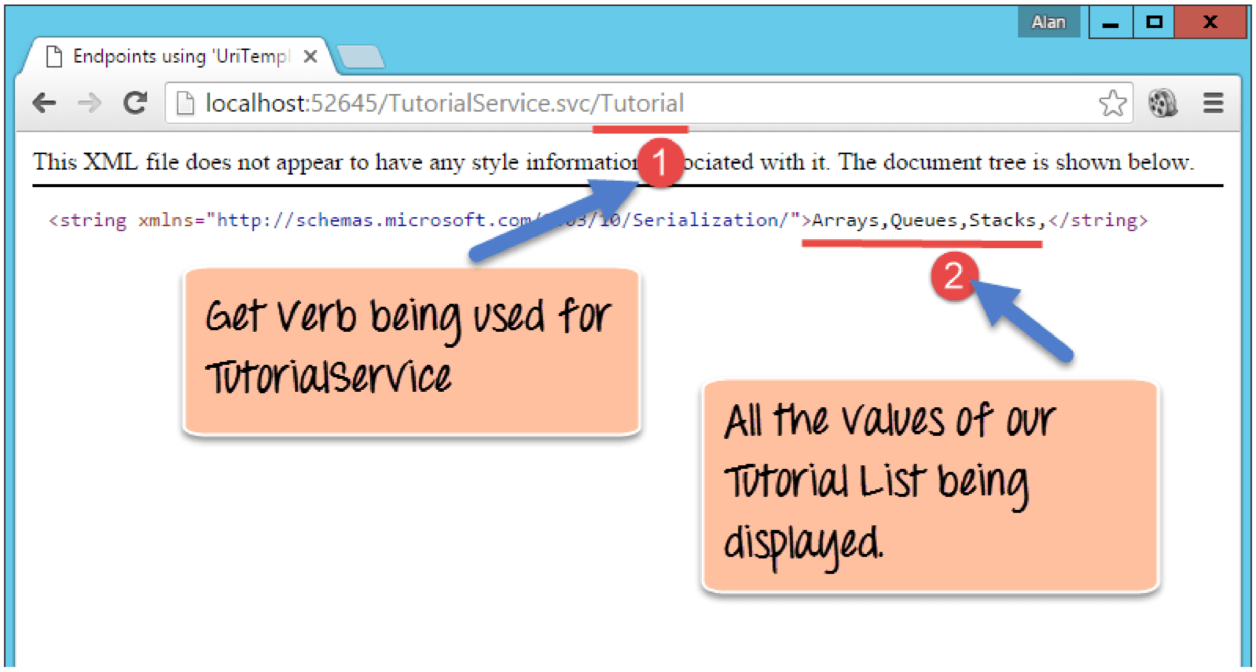
```
[WebGet(UriTemplate="/Tutoria
public String GetAllTutorial(
```

Step 3) The next step is to run the project itself. Now depending on the default browser installed on the system, the appropriate browser name will come next to the run button in Visual Studio. In our case, we have Google Chrome showing up. Just click on this button.


Microsoft Visual Studio (Administrator)
JECT BUILD DEBUG TEAM TOOLS TEST ANALYZE WINDOW HELP
Google Chrome
Debug
Web.config*
TutorialService.svc.cs*
orialService
GetAllTutorial()
Click on 'Google Chrome' run button

Output:-

When the project is run, you can browse to your TutorialService.svc/Tutorial section, and you will get the below output.


Alan
Endpoints using 'UriTempl
localhost:52645/TutorialService.svc/Tutorial
1
2
Get Verb being used for TutorialService
All the values of our Tutorial List being displayed.

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
<string xmlns="http://schemas.microsoft.com/2003/10/Serialization/">Arrays,Queues,Stacks,</string>
```

In the above output,

- You can see that the browser is invoking the 'GET' verb and executing the 'GetAllTutorial' method in the web service. This module is used to display all the Tutorials exposed by our web service.

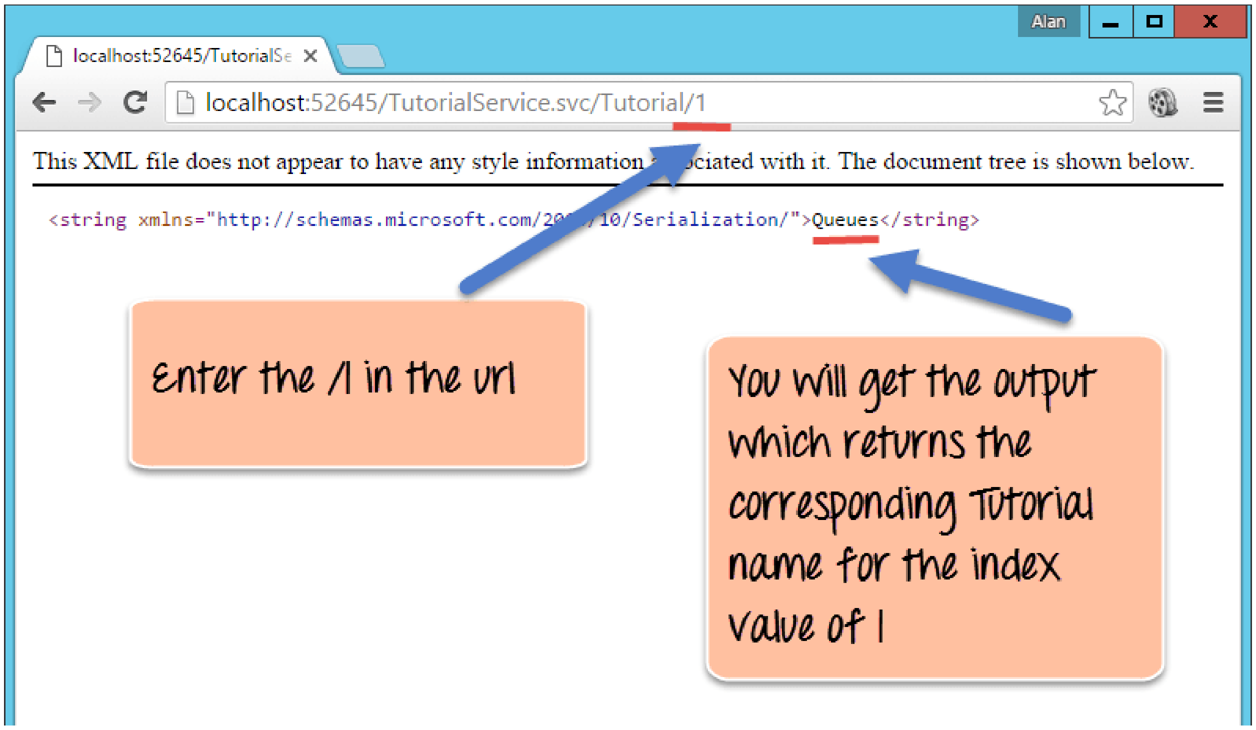
Testing your first Restful web service

In the above section, we have already seen how to use the browser to execute the 'GET' verb and invoke the 'GetAllTutorial.'

1. Let's now use the browser to execute the following use case scenario.

GET Tutorial/Tutorialid - When a client invokes this Restful API, they will be given the Tutorial name based on the Tutorialid sent by the client

In your browser, append the string /1 after the Tutorial word in the URL. If you hit the enter button, you will get the below output


Alan
localhost:52645/TutorialSe
localhost:52645/TutorialService.svc/Tutorial/1
Enter the /1 in the url
You will get the output which returns the corresponding Tutorial name for the index value of 1

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
<string xmlns="http://schemas.microsoft.com/2003/10/Serialization/">Queues</string>
```

Now you will see the output of Queues which actually corresponds to the number 1 in our list of Tutorial Strings. This means that the 'GetTutorialbyID' method is now being invoked from our Webservice. It also shows that the value of 1 is being passed successfully via the browser to our web service and to our method and that is why we are getting the correct corresponding value of "Queues" in the browser.

2. Next let's consume our web service by executing the below scenario. For this, you need to install the tool called "Fiddler" which is a free downloadable tool from the site.

POST Tutorial/Tutorialname - When a client invokes this Restful API, the client will submit a request to insert a Tutorialname. The web service will then add the submitted Tutorial name to the collection.

Run the Filddler tool and perform the below steps;

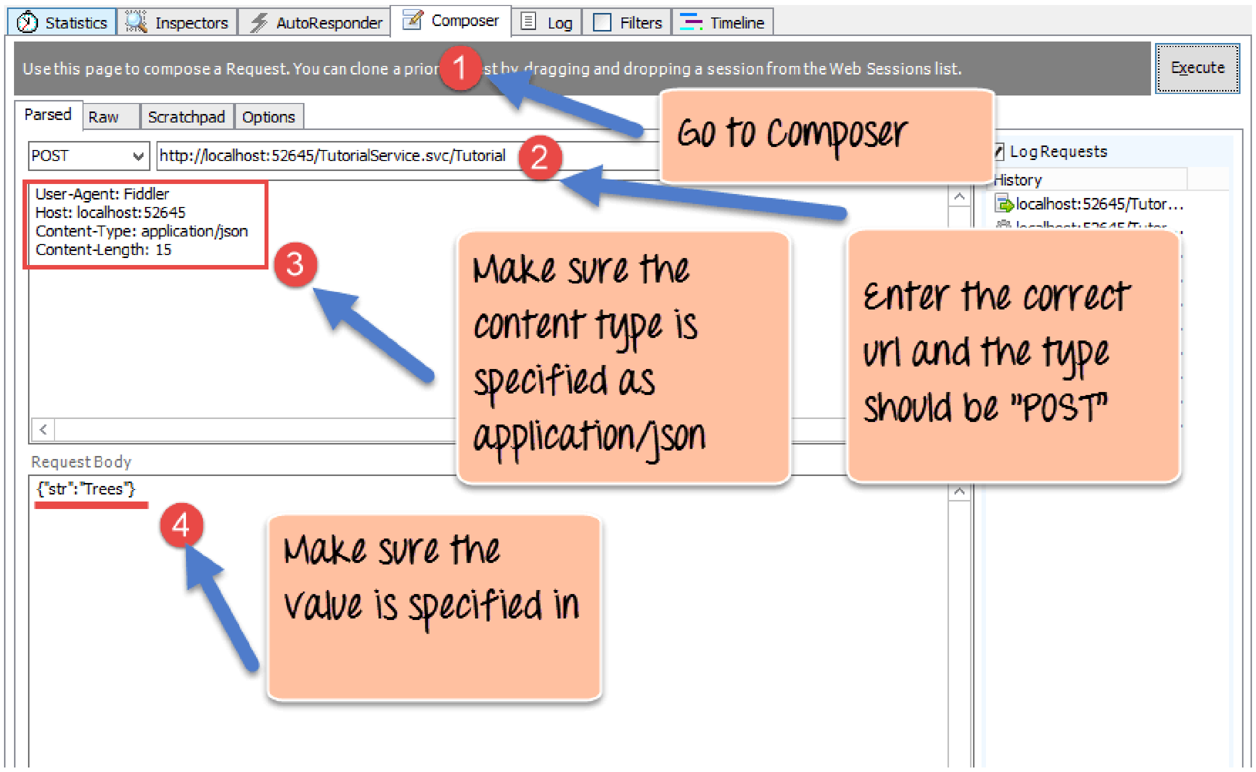
1. Go to the composer section. This is used to create requests which can be submitted to any web

   application.

2. Make sure the request type is "POST" and the correct URL is being hit, which in our case should be **http://localhost:52645/TutorialService.svc/Tutorial**

3. Make sure the Content-Type is marked as application/json. Remember that our POST request method in our Web service only accepts json style data so we need to ensure this is specified when we are sending a request to our application.
4. Finally, we need to enter our data. Remember that our method for POST accepts a parameter called 'str.' So here we are specifying that we want to add a value called "Trees" to our collection of Tutorial names and ensure that it is tagged to the str variable name.

Finally, just click the Execute button in fiddler. This will send a request to the web service to POST the data "Trees" to our web service.

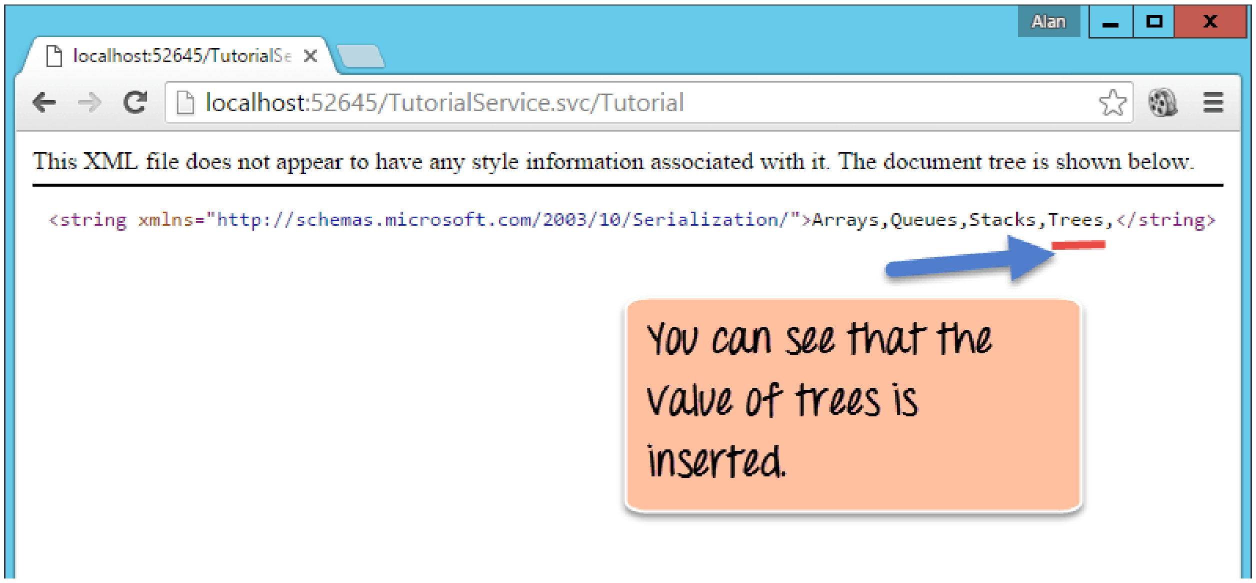

Statistics
Inspectors
AutoResponder
Composer
Log
Filters
Timeline
Use this page to compose a Request. You can clone a prior ... st by dragging and dropping a session from the Web Sessions list.
Execute
Parsed
Raw
Scratchpad
Options
1
Go to Composer
POST
2
Log Requests
History
localhost:52645/Tutor...
3
Make sure the content type is specified as application/json
Enter the correct url and the type should be "POST"
Request Body
4
Make sure the value is specified in

```
http://localhost:52645/TutorialService.svc/Tutorial
```

```
User-Agent: Fiddler
Host: localhost:52645
Content-Type: application/json
Content-Length: 15
```

```
{"str":"Trees"}
```

Now, when we browse to the Tutorial URL to show all the strings in our Tutorial list, you will now see the value of "Trees" is also present. This shows that the POST request to the web service was successfully executed and that it was successfully added to our Tutorial List.



This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
<string xmlns="http://schemas.microsoft.com/2003/10/Serialization/">Arrays,Queues,Stacks,Trees,</string>
```

localhost:52645/TutorialSe
localhost:52645/TutorialService.svc/Tutorial
You can see that the value of trees is inserted.

3. Next let's consume our web service by executing the below scenario. For this also we need to use the fiddler tool

DELETE Tutorial/Tutorialid- When a client invokes this Restful API, the client will submit a request to delete a Tutorialname based on the Tutorialid. The web service will then delete the submitted Tutorial name from the collection.

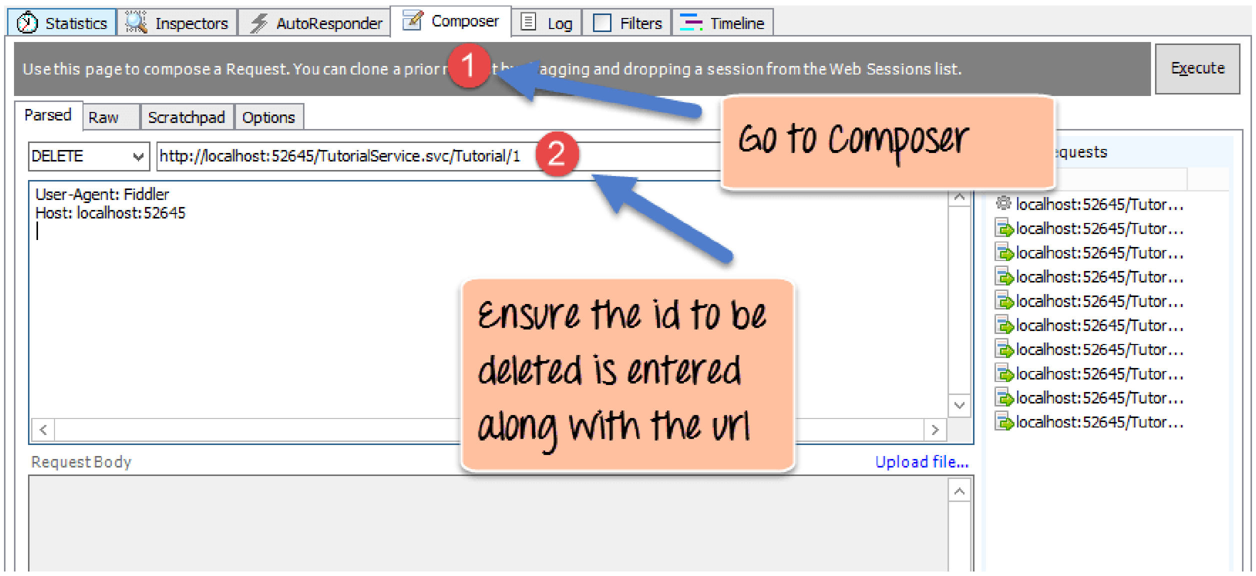
Run the Filddler tool and perform the below steps

1. Go to the composer section. This is used to create requests which can be submitted to any web

   application.

2. Make sure the request type is "DELETE" and the correct URL is being hit, which in our case should be **http://localhost:52645/TutorialService.svc/Tutorial**. Ensure that the id which is used to delete a string in the list sent via the URL as a parameter. In our example, we are sending 1 so this will delete the 2nd element in our collection which is "Queues".

Finally, just click the Execute button in fiddler. This will send a request to the web service to DELETE the data "Queues" to our web service.



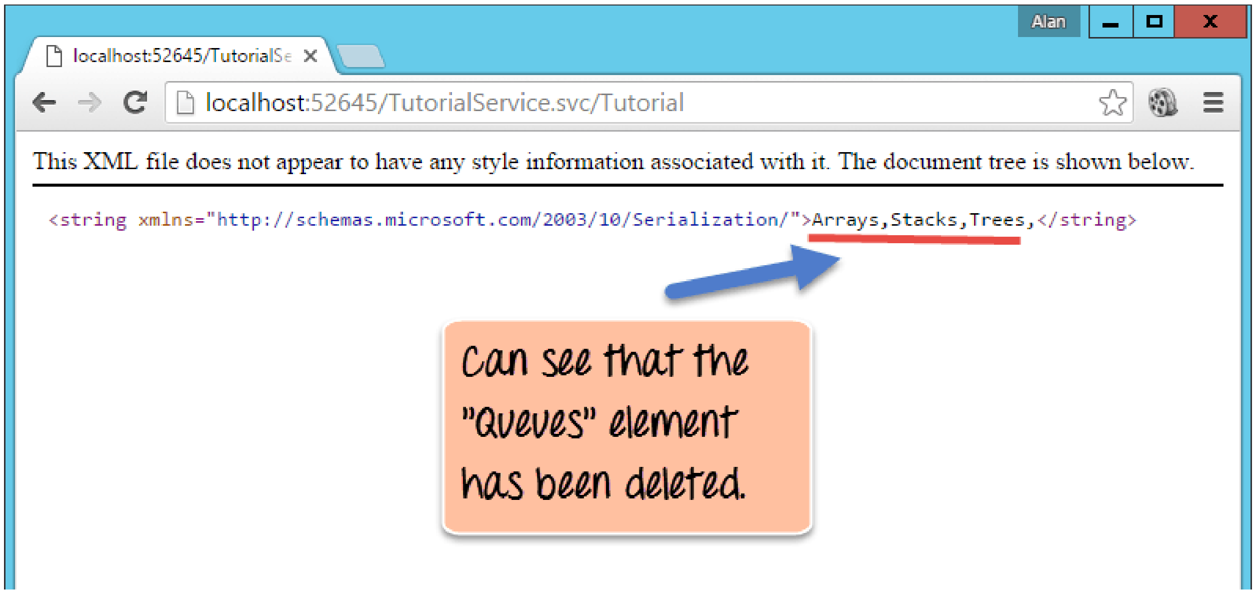
Use this page to compose a Request. You can clone a prior r... th... agging and dropping a session from the Web Sessions list.

DELETE http://localhost:52645/TutorialService.svc/Tutorial/1

```
User-Agent: Fiddler
Host: localhost:52645
```

Now, when we browse to the Tutorial URL to show all the strings in our Tutorial list, you will notice that the value of "Queues" is no longer present.

This shows that the DELETE request to the web service was successfully executed. The element at index no 1 in our list of Tutorial strings was successfully deleted.


Alan
localhost:52645/TutorialSe
Can see that the "Queues" element has been deleted.

localhost:52645/TutorialService.svc/Tutorial

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
<string xmlns="http://schemas.microsoft.com/2003/10/Serialization/">Arrays,Stacks,Trees,</string>
```

Experiment No:-5

Aim:- Design & develop ASP.net web application using session.

Objective:-
To understand Working with the Session Object
To understand Adding Data to the Session Object
To understand Lifetime of the Session Object
To understand Removing Objects from the Session Object

Theory:
Working with the Session Object

Programming with query strings and hidden fields is cumbersome if you need to manipulate more than a trivial amount of state data. Luckily, ASP.NET gives you a better way to store state for each user on the server with the Session object. Every time a new browser hits your ASP.NET application, a new Session object is created for that Web browser. You can store data in the Session object, and it will be available from hit to hit for the same Web browser. Sessions expire after 20 minutes of inactivity by default, although you can change this behavior, as we'll show shortly.
Sessions aren't magic. By default, ASP.NET uses a cookie containing a unique and random ID that it uses to look up the Session object in the ASP.NET server process.
TIP

If needed, you can turn off the cookies to instantiate cookieless sessions. You can also offload the storage of session state onto a different server or into a SQL Server database, for "Web farm" scenarios. We'll explain these different session configurations on Day 18, "Configuring Internet Applications."

Adding Data to the Session Object
You can easily add data to the Session object. The line

```
Session["ValidUser"] = true;
```

automatically creates a new variable called ValidUser (if it doesn't exist already) and sets it to true. By default, every variable added to the Session object is of the .NET Object type.

Because variables in the Session object are of type Object, cast them to the appropriate type when accessing them. For instance,

```
String strUserName = Session["UserName"];  //error!
```

generates a compiler error, complaining that you can't assign an Object to a String. The following line would work, however:

```
String strUserName = (String) Session["UserName"];
```

You might be wondering whether it's appropriate to store large custom objects in Session. The answer is that you should avoid storing large amounts of data in session if possible. You can easily overburden your Web server by storing large amounts of data in Session, especially if your Web site has many users. Databases are a better choice for storing large amounts of state data.
The Session object is of type HTTPSessionState. Its default property is the Item collection, which allows you to access the stored items by using the [] notation.
Lifetime of the Session Object
A new session is created once for each new browser that hits your ASP.NET Web site. If a user stops hitting your Web site, his Session will time out after 20 minutes of inactivity, by default.
You can find out how long the Session timeout setting is by using the Timeout method. The following code line prints "20" by default:

```
<% Response.Write(Session.Timeout.ToString()); %>
```

You can change the timeout for the Session object by assigning the Timeout property to a certain value, in minutes, such as

```
Session.Timeout = 5;
```

Removing Objects from the Session Object

Because sessions time out, you don't really need to remove objects from them. However, you can remove objects by using the Session.Remove() method. You also can remove everything in the Session object by using the RemoveAll() method. You might want to use these two methods to conserve Web server resources, especially if you store large objects in Session.

Listing 3.6 shows a page that lets you add and remove strings to the current Session.

Code:-

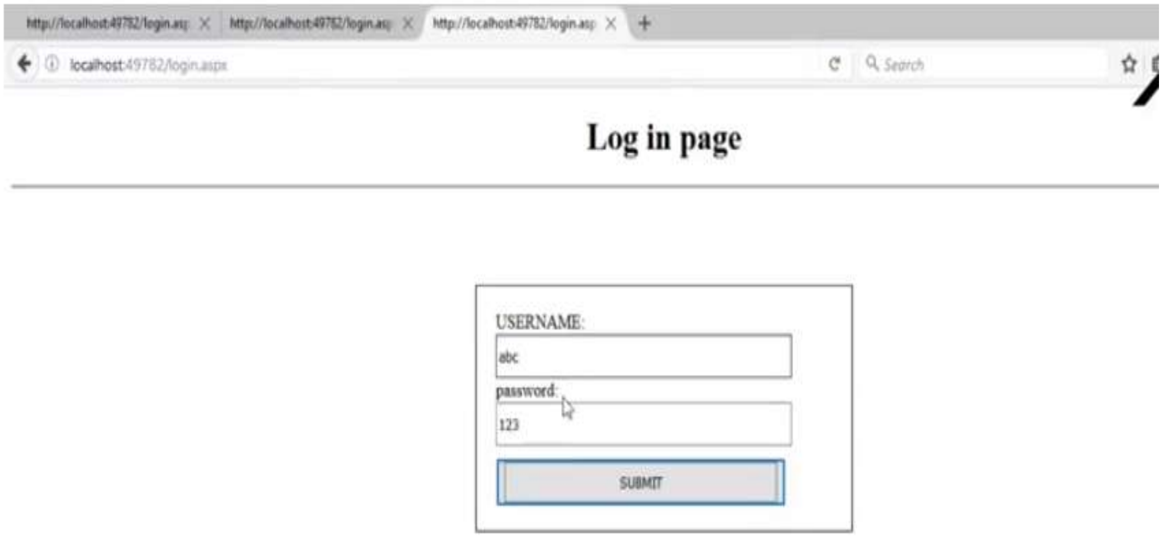
Listing 3.6 SessionPopulate.aspx: Adding and Removing Strings from Session State

```
<%@ language="C#" %>

<script runat="server">

void AddClicked(Object Sender, EventArgs e)

{

  Session[Key.Text] = Value.Text;

}

void RemoveClicked(Object Sender, EventArgs e)

{

  Session.Remove(Key.Text);

}

</script>

<html>

<body>
```

```
<h3>Current items in Session</h3>

<form method="post" runat="server">

<table border="1">

  <tr>

    <td><b>Item Name</b></td>

    <td><b>Value</b></td>

  </tr>

<%

  String strSesKeyName;

  String strSesItem;

  for(int i=0; i<Session.Count; i++) {

    strSesKeyName = Session.Keys[i];

    strSesItem = (String) Session[i];

    Response.Write("<tr><td>" + strSesKeyName + "</td><td>" +

                    strSesItem + "</td></tr>");

  }

%>

</table>
```

```
<br>

Key <asp:textbox id="Key" runat="server"/>

Value <asp:textbox id="Value" runat="server" /><br>

<asp:button text="Add/Modify Key/Value pair"

            onclick="AddClicked" runat="server" />

<asp:button text="Remove Key" onclick="RemoveClicked" runat="server" />

</form>

</body>

</html>
```

Output:-



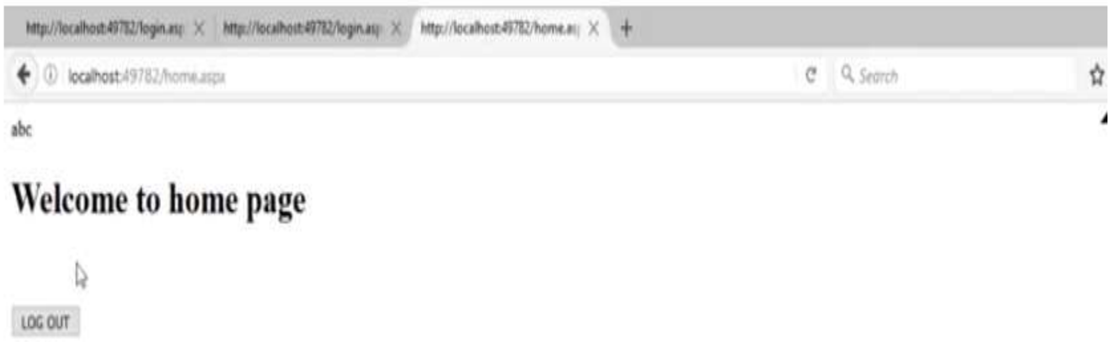
Log in page

USERNAME:

abc

password:

123

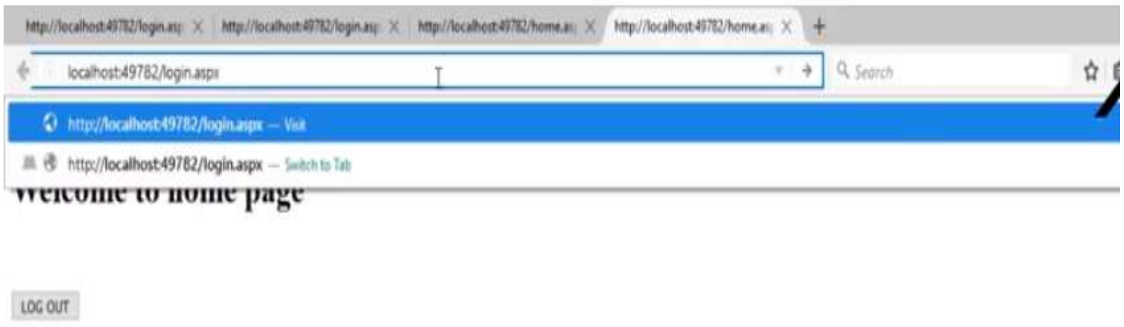

http://localhost:49782/login.asp | http://localhost:49782/login.asp | http://localhost:49782/home.as

localhost:49782/home.aspx

abc

Welcome to home page

LOG OUT


http://localhost:49782/login.as | http://localhost:49782/login.as | http://localhost:49782/home.as | http://localhost:49782/home.as

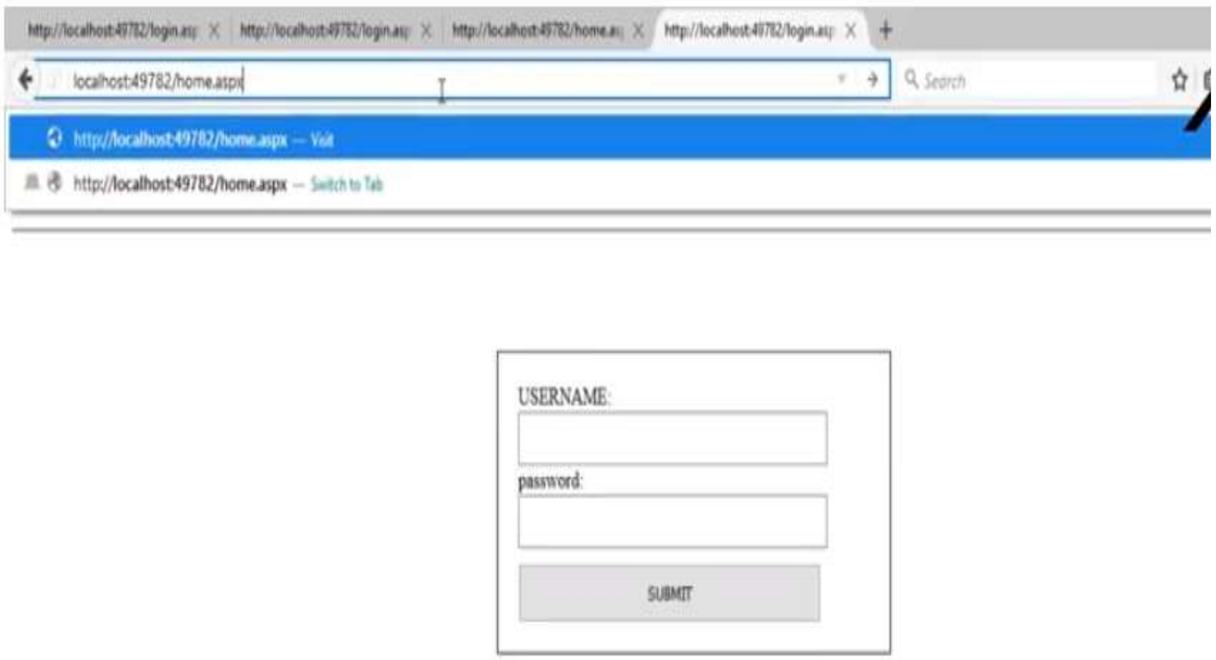
localhost:49782/login.aspx

http://localhost:49782/login.aspx — Visit

http://localhost:49782/login.aspx — Switch to Tab

Welcome to home page

LOG OUT


http://localhost:49782/login.asp
http://localhost:49782/login.asp
http://localhost:49782/home.as
http://localhost:49782/login.asp
localhost:49782/home.aspx
Search
http://localhost:49782/home.aspx — Visit
http://localhost:49782/home.aspx — Switch to Tab
USERNAME:
password:
SUBMIT

Experiment No:-6

Aim:- Call an external web service through SharePoint workflow.

Objectives:-

To understand a custom web services in a SharePoint workflow.

Theory:

Creating web services for SharePoint workflows

With the support for calling web services and the lack of supporting custom code within workflows, developers will now need to know how to create services. There are plenty of options for creating custom web services for use in SharePoint workflows. The **HttpSend** activity and **DynamicValue** data type are best suited for RESTful services and those that conform to the OData Protocol.

OData is a protocol for creating and consuming data based on the principles of REST services. It was developed in an effort to standardize exchanging data using the mature, reliable, and robust HTTP protocol. Once the OData specification was complete, different organizations implemented the protocol on their own technology stacks. Microsoft implemented its own version of OData and branded it Windows Communication Foundation (WCF) Data Services 5.0.

The RESTful services implemented by SharePoint actually support OData because they were built using WCF Data Services, specifically WCF Data Services 5.0, which implements the OData 3.0 specification.

Walkthrough: Create a workflow with Visual Studio 2012

The following walkthrough demonstrates how to create a custom workflow that calls an OData web service on the Northwind database. You can find the Northwind database hosted at OData.org.

When the workflow is completed, users will enter a customer ID, then start the workflow. When started, the workflow retrieves additional customer information and updates the list item with the data it has retrieved.

1. Start Visual Studio 2012 and create a new SharePoint-hosted app project.
2. In this project, create a new custom list and name it "Customers".
3. In this new list, create the following fields. Leave the default data type for each field as **string**:
   - CustomerId (renamed from the default "Title" field)

- Customer Name
- Job Title
- Address
- Country/Region
- Business Phone

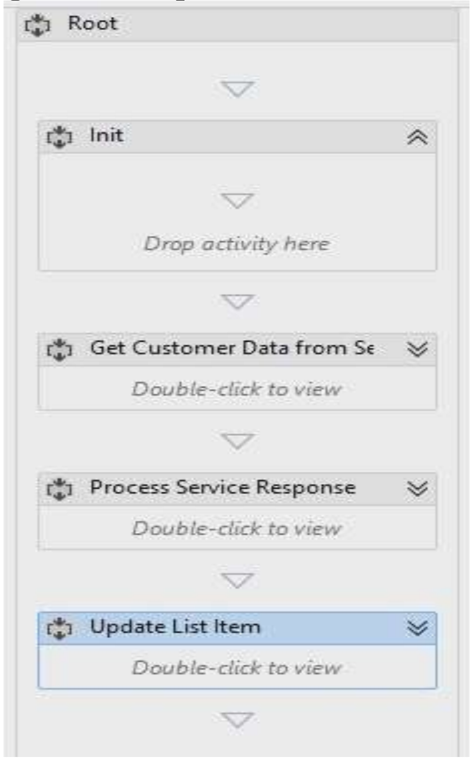
- Fax Number

1. Now, add a workflow to the project by clicking in **Solution Explorer** on **Add** > **New Item**; then, in the **Add New Item** dialog box, select the **Workflow** project item from the **Office/SharePoint** category.
2. Name the workflow "CompleteCustomerDetails" and click **Next**.
3. When prompted by the **Customization wizard**, name the workflow "Complete Customer Details" and set it to be a **List** workflow. Cick **Next**.
4. On the next wizard page, check the box to create an association, select the **Customer** list, then select **Create New** for the workflow history and task lists. Click **Next**.
5. On the final wizard page, check the box to start the workflow manually; leave the option to start automatically **un** -checked. Click **Finish**.
6. At this point, Visual Studio displays the workflow designer surface that contains a single **Sequence**activity.
7. Change the name of the **Sequence** activity to **Root**.
8. Add four more **Sequence** activities inside the Root activity and name them as follows:
   - Init

- Get Customer Data From Service
- Process Service Response
- Update List Item

1. At this point, the workflow will appear as shown in Figure 2.

Figure 2. Complete Customer Details Workflow with Four Empty Sequences



Get the customer ID entered by the user

The first thing the workflow needs to do is retrieve the customer ID, as entered by the user. To do this, you need to create two variable.

1. Click the **Variables** tab at near the bottom of the workflow designer and create two variables
   - **CustomerItemProperties** (data type = **DynamicValue**; scope = **Init**). Use this variable to store the result set returned by the activity that gets all properties from the list item.

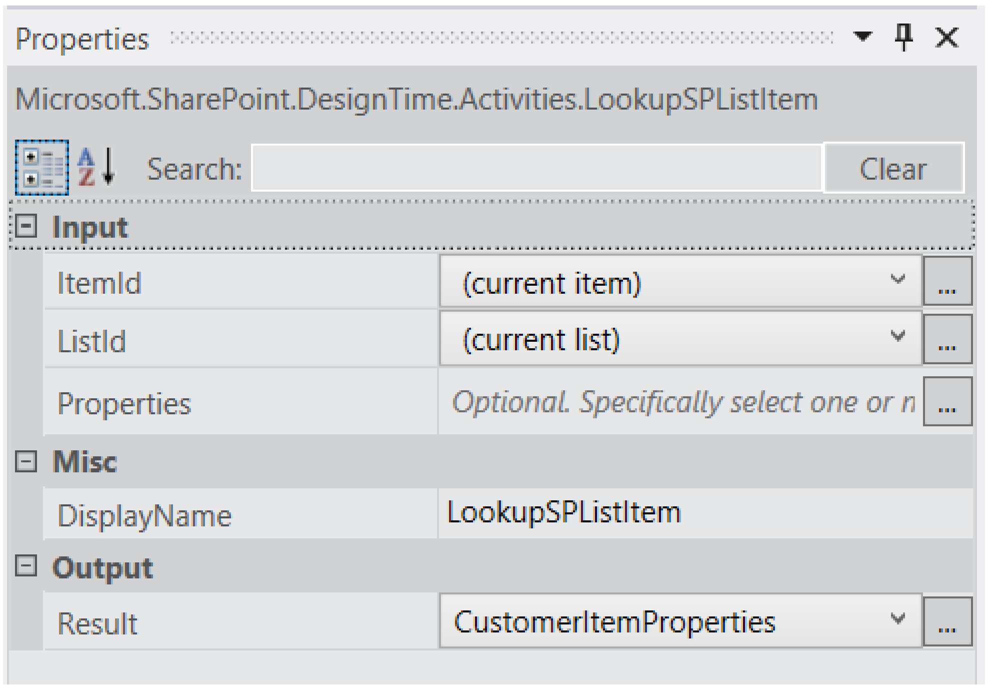
Note

The **DynamicValue** data type is not shown by default. To find it, select the **Browse for Types**option in the **Variable Type** column. In the search box at the top of the dialog, enter **DynamicValue**, and then select the **Microsoft.Activities.DynamicValue**.

   - **CustomerId** (data type = **String**; scope = **Root**): Use this variable to store the customer ID entered by the user.

   Locate the **LookupSpListItem** activity in the **SP - List** section of the toolbox and drag it to the **Init**sequence. Set the activity properties as shown in Figure 3.

Figure 3. Properties Tool Window for the LookupSPListItem Activity


Properties
Microsoft.SharePoint.DesignTime.Activities.LookupSPListItem

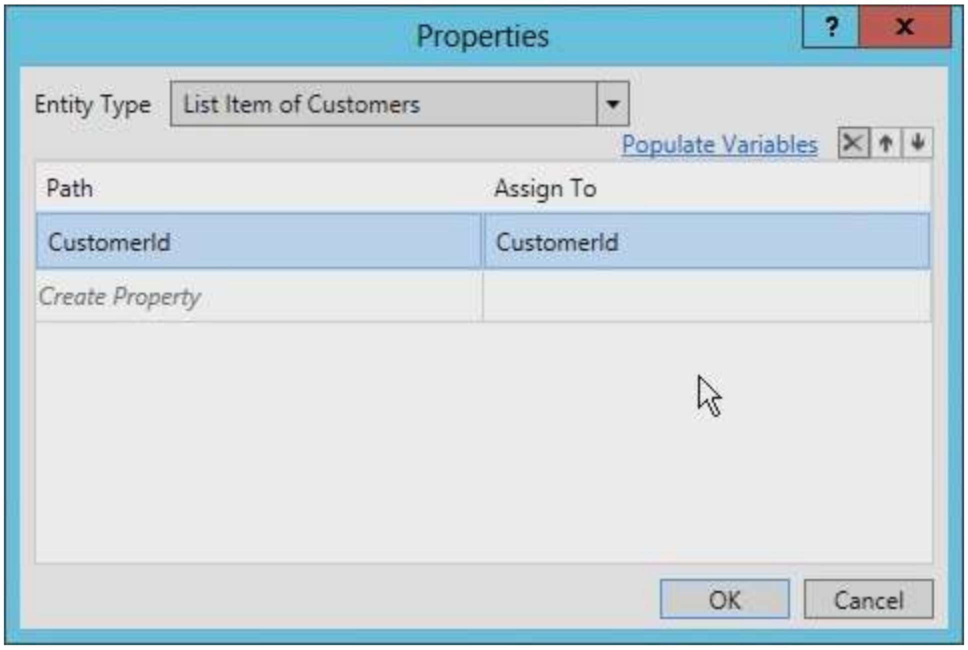
Input	
ItemId	(current item)
ListId	(current list)
Properties	*Optional. Specifically select one or n*
Misc	
DisplayName	LookupSPListItem
Output	
Result	CustomerItemProperties

Copy

This activity tells Workflow Manager to use the SharePoint REST API to retrieve the properties of the current list item and to store the **JSON** response in the **DynamicValue** variable that you just created.

1. Retrieve the customer ID from the list item by clicking the Get Properties link in the **LookupSpListItem**activity. Doing this adds a **GetDynamicValueProperties** activity to the design surface.
2. In the **Properties** dialog box, click the ellipsis (**???**) to open the Property selector, shown in Figure 4. In the wizard, set the **Entity Type** to **List Item of Customers**, then add a single property, CustomerId, with the Path set to CustomerId and Assign To set to CustomerId (the variable previously created), as shown in the following figure.
3. Click **Create Property** and enter **CustomerId** in the **Path** column.
4. In the **Assign To** column, enter **CustomerId**, which is the variable we created earlier. Figure 4 shows the completed **Properties** dialog box.

Figure 4. Properties dialog for the GetDynamicValueProperties Activity.



Entity Type: List Item of Customers

Path	Assign To
CustomerId	CustomerId
Create Property	

Call the Northwind OData web service

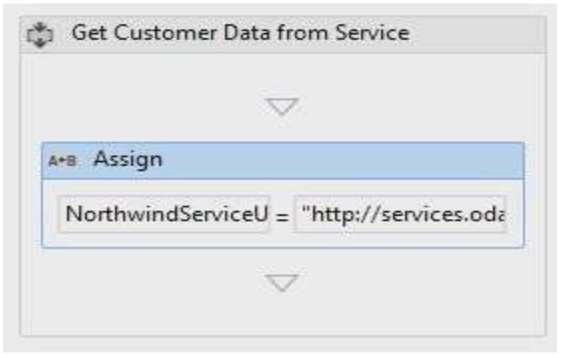
The workflow now has a reference to the customer ID, so the next step is to call the web service. To do this, we'll be working primarily with the **Get Customer Data from Service** sequence.

1. Select the **Get Customer Data from Service** sequence and create two new variables:
   - **NorthwindServiceUri** (data type = **String**; scope = **Get Customer Data from Service**). This variable stores the URI that is used to query the web service.

- **NorthwindServiceResponse** (data type = **DynamicValue**; scope = **Root**): This variable will store the web service response.

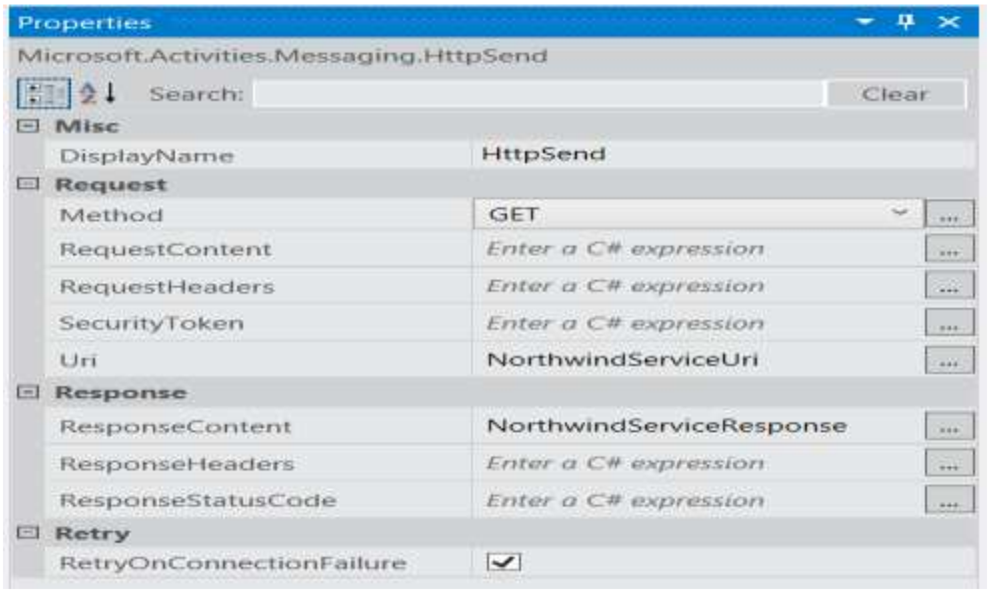
1. To create the URL to query the web service, start by locating an **Assign** activity in the workflow toolbox and drag it to the **Get Customer Data from Service** sequence. Notice that the **Assign** activity has two parts representing a name-value pair.
2. Set the left portion of the **Assign** activity to **NorthwindServiceUri**.
3. Set the right portion of the activity to the string"http://services.odata.org/Northwind/Northwind.svc/Customers('" + CustomerId + "')?$format=json". Figure 5 shows the completed activity.

Figure 5. Assign Activity Used to Set a Variable Containing the OData Service


Get Customer Data from Service
Assign
NorthwindServiceU = "http://services.oda

1. Drag an **HttpSend** activity from the toolbox to the **Get Customer Data from Service** sequence, immediately following the **Assign** activity.
2. Set the properties on the **HttpSend** activity using the values shown in Figure 6.

Figure 6. HttpSend Properties


Properties
Microsoft.Activities.Messaging.HttpSend

Misc	
DisplayName	HttpSend
Request	
Method	GET
RequestContent	*Enter a C# expression*
RequestHeaders	*Enter a C# expression*
SecurityToken	*Enter a C# expression*
Uri	NorthwindServiceUri
Response	
ResponseContent	NorthwindServiceResponse
ResponseHeaders	*Enter a C# expression*
ResponseStatusCode	*Enter a C# expression*
Retry	
RetryOnConnectionFailure	✓

Process the Northwind OData web service response

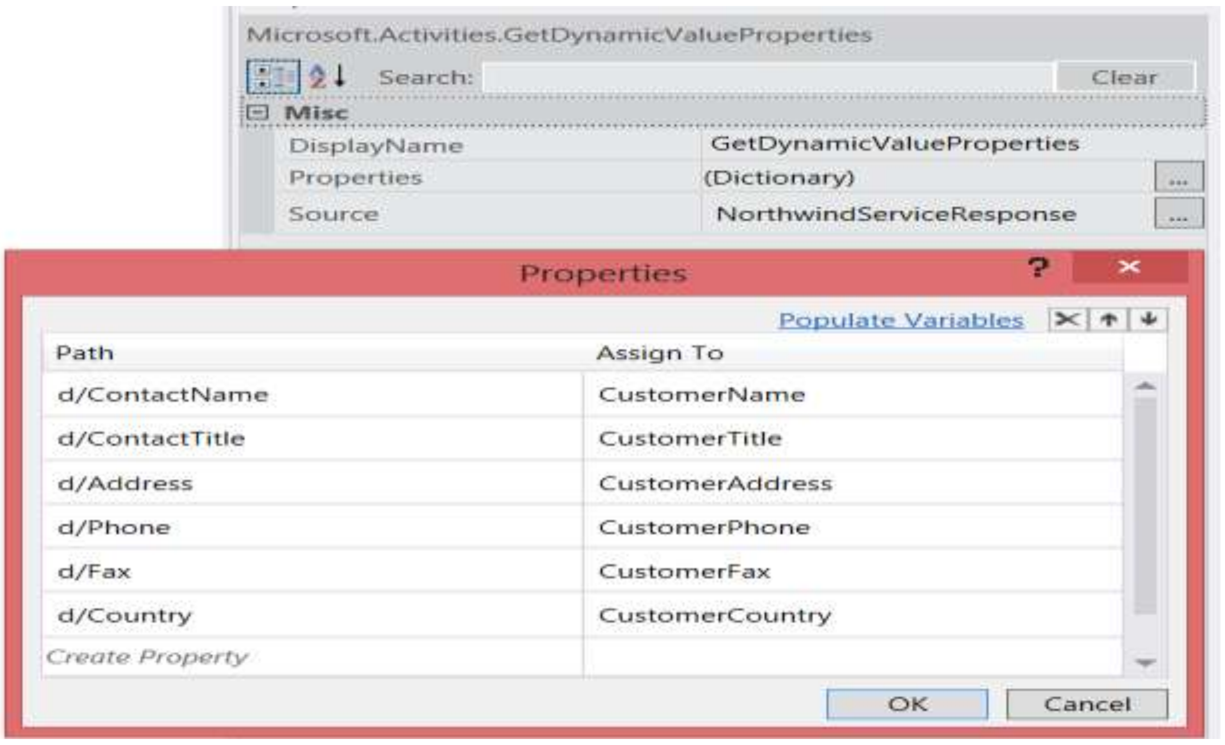
Once the web service request has been made and the results are stored in a local variable, the next step is to process the response. Each value in the response needs to be added to a different variable.

1. Create a variable for each of the fields that we created at the start of this walkthrough (except the customer ID field), shown here:
   - Customer Name

- Job Title
- Address
- Country/Region
- Business Phone
- Fax Number

1. Name each of these variables according to its respective field name.
2. All of the variables should be of type **String**; all of the variables should be scoped to **Root**.
3. Add a **GetDynamicValueProperties** activity to the **Process Service Request** sequence.
4. In the **Properties** window, set the **Source** value to **NorthwindServiceResponse**, as shown in Figure 7.
5. Click the ellipsis button (**???**) button on the **Properties** property and then provide values in the **Path** and **Assign To** columns as shown in Figure 7. Notice that the values in the **Assign To** column are the variable you created for each of the **Customers** list fields.

Figure 7. Properties tool window for GetDynamicValueProperties and contents for Properties dialog


Microsoft.Activities.GetDynamicValueProperties

DisplayName	GetDynamicValueProperties
Properties	(Dictionary)
Source	NorthwindServiceResponse

Path	Assign To
d/ContactName	CustomerName
d/ContactTitle	CustomerTitle
d/Address	CustomerAddress
d/Phone	CustomerPhone
d/Fax	CustomerFax
d/Country	CustomerCountry
Create Property	

Update the customer list item

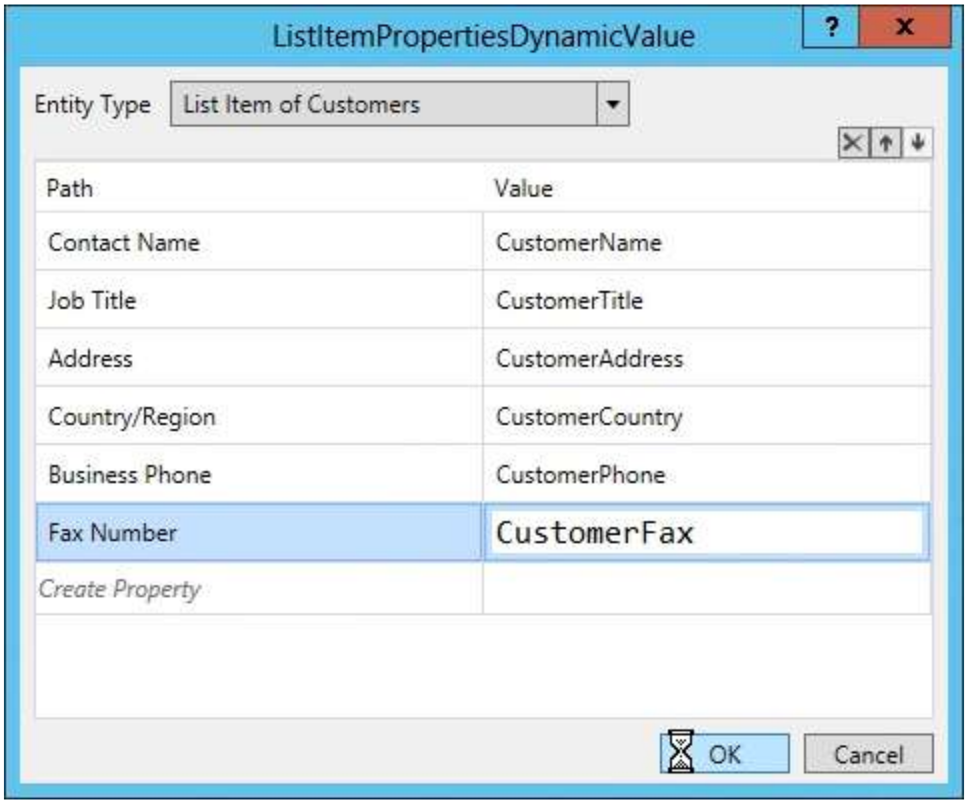
The last step is to update the list item.

1. Add an **UpdateListItem** activity to the **Update List Item** sequence and use the **Properties** window to set the following values:
   - **ListId**: (current list)

- **ItemId**: (current item)

1. Click the ellipsis button (**???**) button on the **ListItemPropertiesDynamicValues** property and in the resulting dialog box, set **Entity Type** to **List Item of Customers**.
2. Finally, for each of the values extracted from the web service, set the values on the list item to the variables in the workflow, as shown in Figure 8.

Figure 8. ListItemPropertiesDynamicValue Dialog with Values Set



Path	Value
Contact Name	CustomerName
Job Title	CustomerTitle
Address	CustomerAddress
Country/Region	CustomerCountry
Business Phone	CustomerPhone
Fax Number	CustomerFax

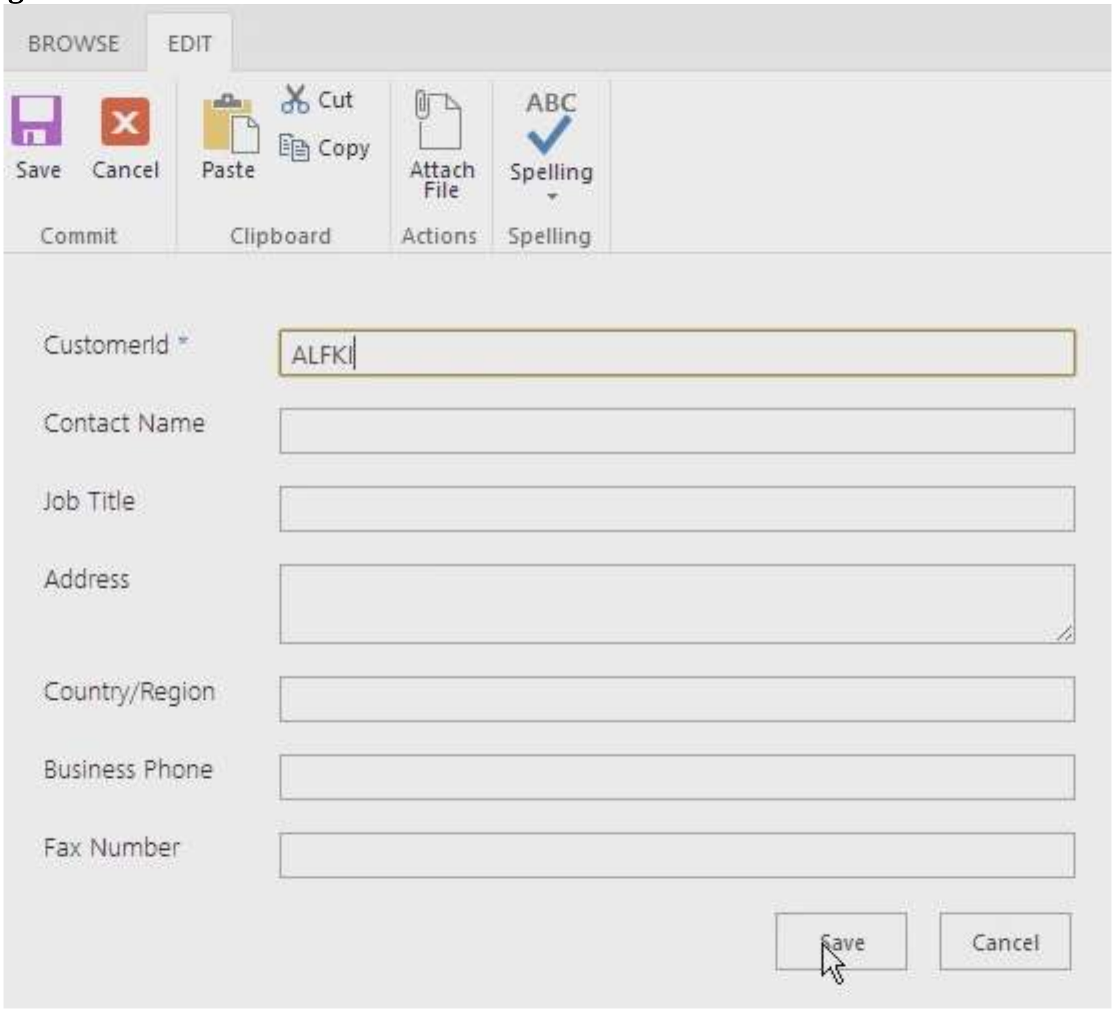
Output:-

Test the workflow

The workflow is now complete and should function properly. To confirm its stability, you should test it.

1. Press **F5** to start debugging; Visual Studio builds and deploys the SharePoint-hosted app.
2. When the browser opens, navigate to the **Customers** list, create a single customer record with a **Customer Id** of "ALFKI", as shown in Figure 9, and then save the item.

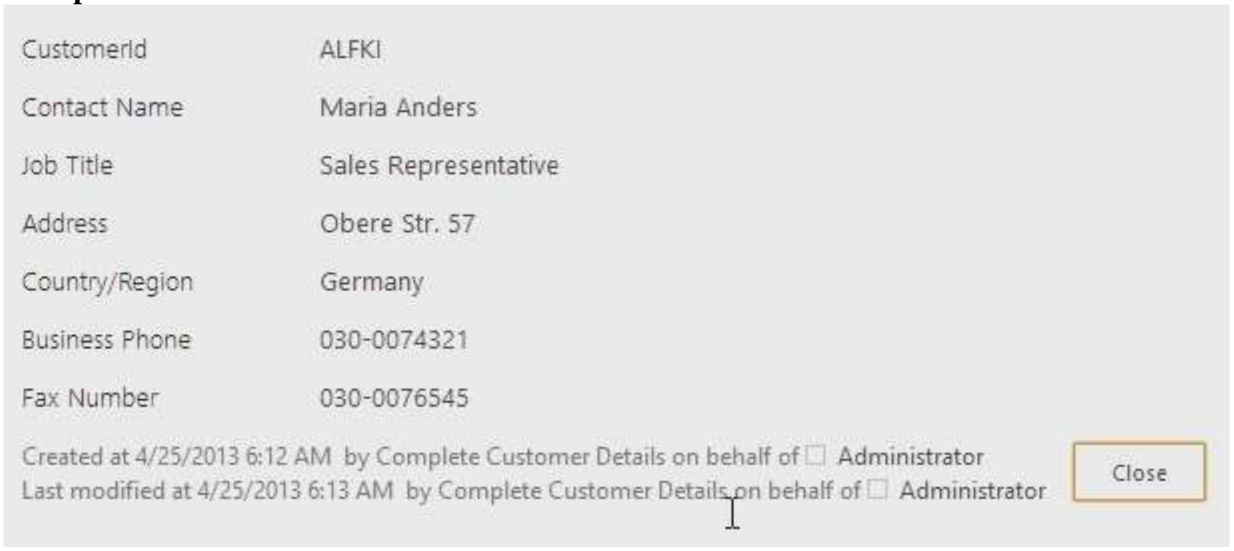
Figure 9. New List Item



CustomerId *	ALFKI
Contact Name	
Job Title	
Address	
Country/Region	
Business Phone	
Fax Number	

Next, manually start the workflow and then go back to the list item. Keep refreshing the page to see the workflow update the list item, as shown in Figure 10

Figure 10. Updated List Item



CustomerId	ALFKI
Contact Name	Maria Anders
Job Title	Sales Representative
Address	Obere Str. 57
Country/Region	Germany
Business Phone	030-0074321
Fax Number	030-0076545

Created at 4/25/2013 6:12 AM by Complete Customer Details on behalf of □ Administrator
Last modified at 4/25/2013 6:13 AM by Complete Customer Details on behalf of □ Administrator

Notice that the lis

t item was updated by the SharePoint hosted app on behalf of the person who started the workflow. In this walkthrough, however, it was started by the administrator.

Experiment No:-7

Aim:- Create custom action using sharepoint workflow.

Objective:-

To understand Tasks assigned to users and groups that can be created and monitored by the workflows.

To understand Forms that collect information from users when workflows are associated with a type of content (for example, a site, list, or library) or when a workflow is started.

Theory:

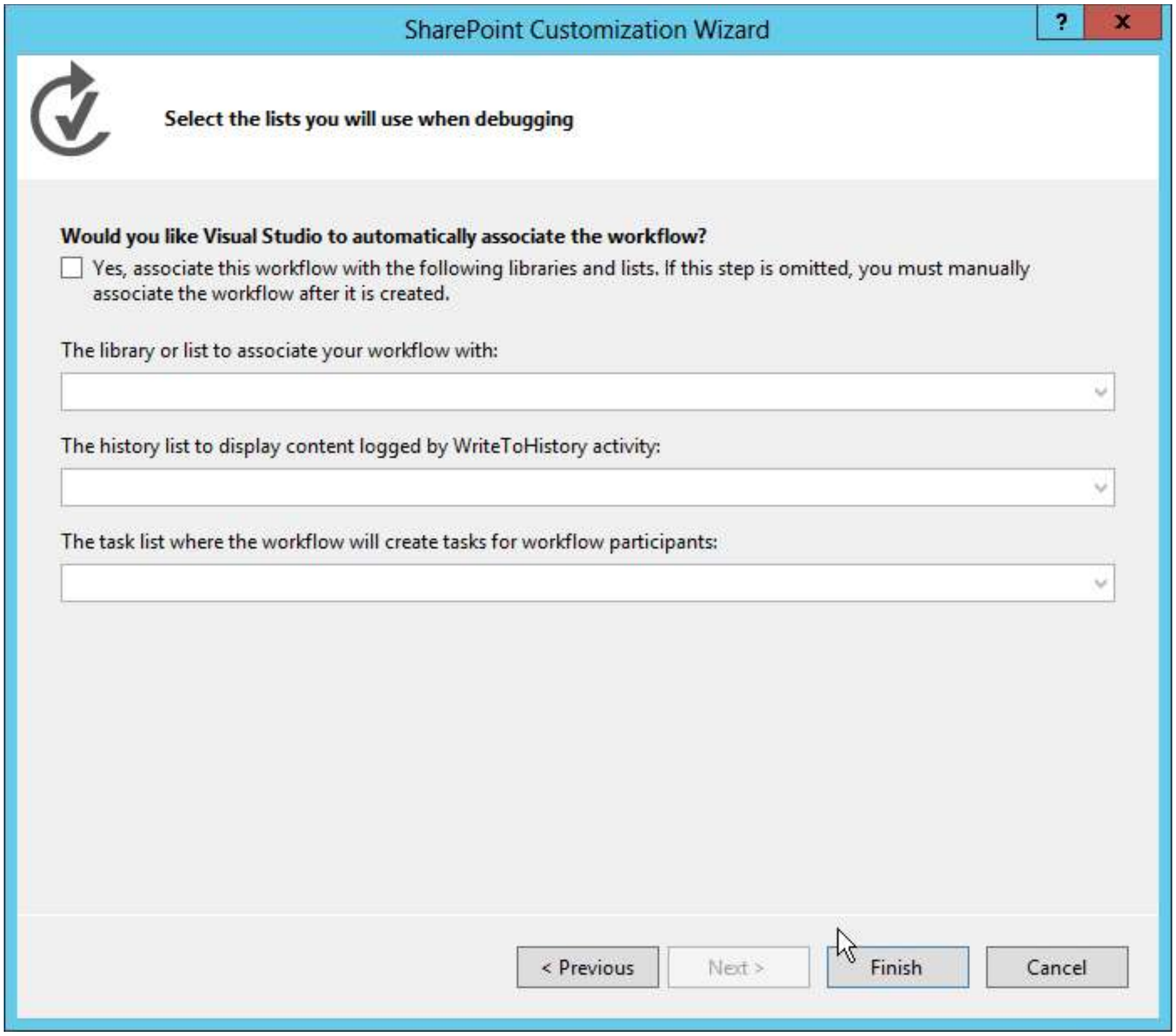
Walkthrough: Create and deploy a custom association form

In this walkthrough we demonstrate how to create a custom association form and use it to collect information that is then passed to the workflow. Before you begin, ensure that you have access to a SharePoint developer site.

Create the custom association form

1. Create a custom workflow using Visual Studio 2012.
2. In Visual Studio, create a new SharePoint app project and configure it as a SharePoint-hosted app.
3. Add a new **Announcement** list to the project. This list will be associated with the workflow and events on this list will trigger the workflow.
4. Add a workflow item the project by right-clicking the project in the **Solution Explorer** and selecting **Add**, followed by **New Item**. In the **Add New Item** dialog box, select the **Workflow** project item from the **Office/SharePoint** category. Enter "SampleWorkflow" as the name and then click **Next**. When prompted by the SharePoint Customization Wizard, set the new item to be a **List Workflow**.
5. The next page of the **SharePoint Customization Wizard**, shown in Figure 1., allows you to automatically create a workflow association. However, when you're creating a custom association forms, you do **not**want to exercise this option. Instead, deselect this check box and click **Finish**.

   Figure 1. Deselect the option to automatically associate the workflow.


SharePoint Customization Wizard

Select the lists you will use when debugging

Would you like Visual Studio to automatically associate the workflow?

- [ ] Yes, associate this workflow with the following libraries and lists. If this step is omitted, you must manually associate the workflow after it is created.

The library or list to associate your workflow with:

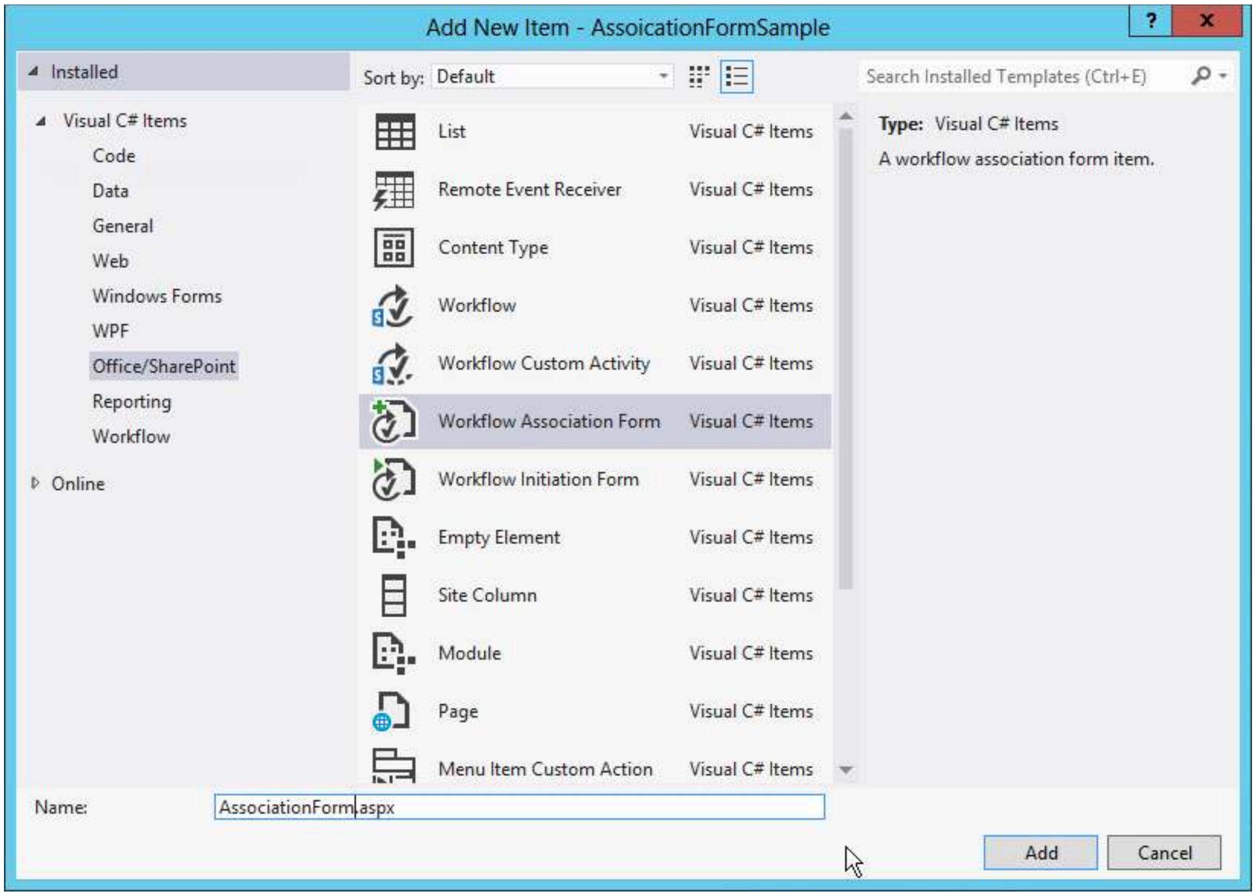
The history list to display content logged by WriteToHistory activity:

The task list where the workflow will create tasks for workflow participants:

1. Next, add the form to the Visual Studio project by right-clicking the workflow item in **Solution Explorer**, then selecting **Add**, **New Item**.

   This last step is important because it tells the **Add New Item** dialog that the context is a workflow item. This then causes the **Add New Item** dialog to display the two form project item templates (Initiation form and Association form) as options, as shown in Figure 2.

Figure 2. Selecting the Association Form template.



Type: Visual C# Items

A workflow association form item.

Name: AssociationForm.aspx

1. Select the **Workflow Association Form** item and remove the "1" from the field name. Click Add to complete the process.

Update HTML and JavaScript in the default association form

Once the new form is added to the project, Visual Studio automatically opens it. At this point, you only need to do two things to the form to make it compatible with your workflow:

- Update the HTML form to reflect the data elements that you need to collect from the user, as well as indicate how the form should be rendered.
- Update the default JavaScript to pull the values from the updated form and match the property names to the names of the arguments that you created in the workflow.

1. Open the form to view the code.
2. Locate the server control shown in the following code snipped:

Copy

```
<WorkflowServices:WorkflowAssociationFormContextControl
ID="WorkflowAssociationFormContextControl1" runat="server" />
```

Copy

This server control performs two important tasks. First, it adds the JavaScript libraries needed by

Code:-

1. Scroll down in the source file until you locate the sample HTML table and replace it with the following:

XMLCopy

```
 <table>
   <tr>
     <td colspan="2">
       String:<br /><textarea id="strInput" rows="1" columns="50"/>
     </td>
   </tr>
   <tr>
       <td><button id="Save" onclick="return runAssocWFTask()">Save</button></td>
       <td><button id="Cancel" onclick="location.href = cancelRedirectUrl; return false;">Cancel</button></td>
   </tr>
</table>
```

Copy

This table displays a simple HTML textbox which is used to pass information into the workflow association. Note that the form has two buttons These buttons are used to save or cancel the workflow. When you click the **Save** button, the workflow calls the JavaScript function, **runAssocWfTask()**, which is located a few lines farther down in the source file. We need to modify that next.

Update the Workflow Services JSOM to create the workflow association

Immediately following the HTML form section in the source file there is a *ecmascriptshort* block of code that is about 200 lines long. This code block illustrates the new Workflow Services JavaScript Client Side Object Model (JSOM) API implementation in SharePoint. For the most part, this JavaScript code should be left unchanged because it does some important things:

- Determines whether a new associated workflow task list should be created and, if so, creates it.
- Determines whether a new associated workflow history list should be created and, if so, creates it.
- Creates a new workflow association with the specified name, workflow definition, startup options, and associated lists.

The critical part that you need to be concerned with when creating the custom association forms is where the values from the form are collected and passed into the new association. We cover this in the following procedure.

1. In the workflow association JSOM script block, locate the JavaScript function **associateWF()**.
2. Inside the script block for this function, locate the line that defines a new array named **metadata**:

var metadata = new Object();

1. Next, add a collection of name-value pairs representing your form fields that you wish to pass into SharePoint. For the custom form in this walkthrough, all you need is the following JavaScript, so update the block that sets the **metadata** variable as follows:

XMLCopy

```
var strInputValue = document.getElementById("strInput").value;
if (strInputValue) {
  metadata['AssociationFormValue'] = strInputValue;
}
```

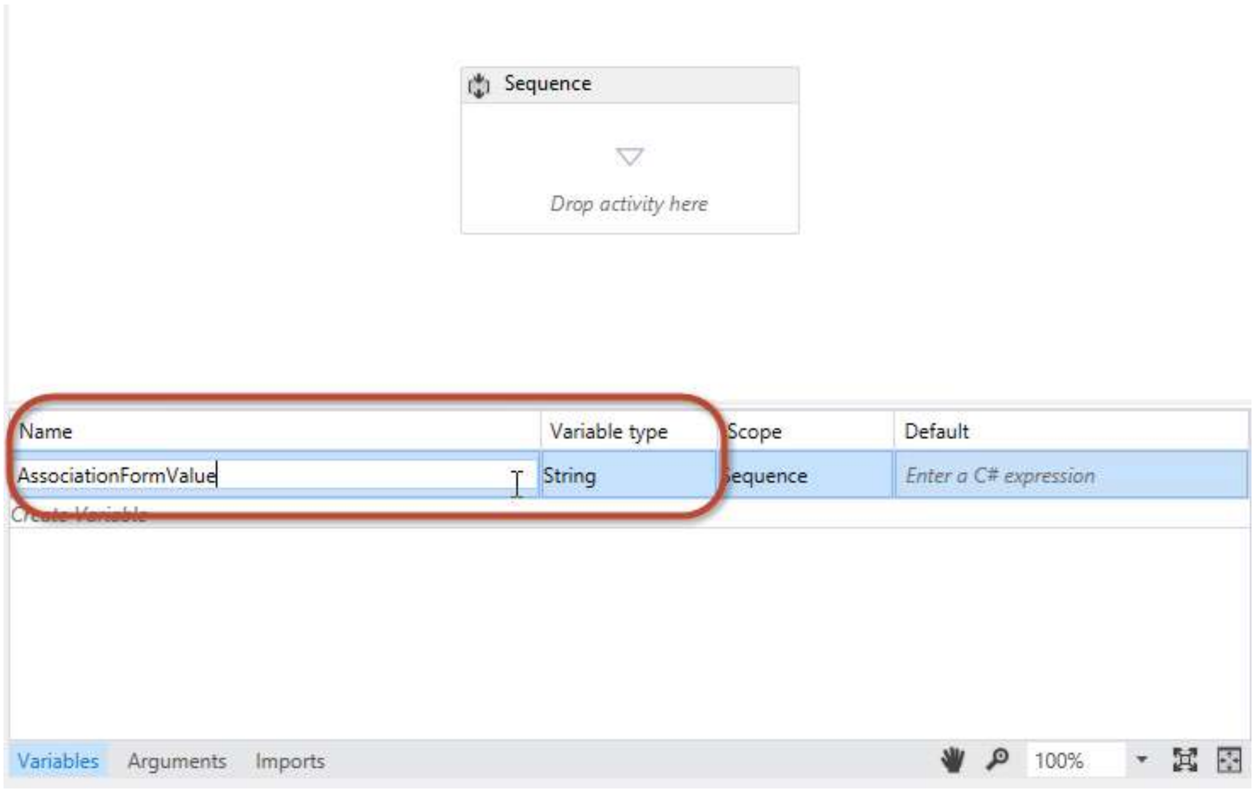
1. At this point the custom association form is complete.

Consume the association form values in the workflow

With the form complete, the next step is to configure the workflow to use the values being passed in from the association form. When a value is passed in from the association form, it is passed in as a configuration value. To obtain this, use a special activity to extract the configuration value from the workflow's association metadata and store it in a variable for later use.

1. Open the workflow in Visual Studio, go to the **Variables** tab, and create a new string variable called **AssociationFormValue**, as shown in Figure 3.

   Figure 3. Creating the AssociationFormValue variable.

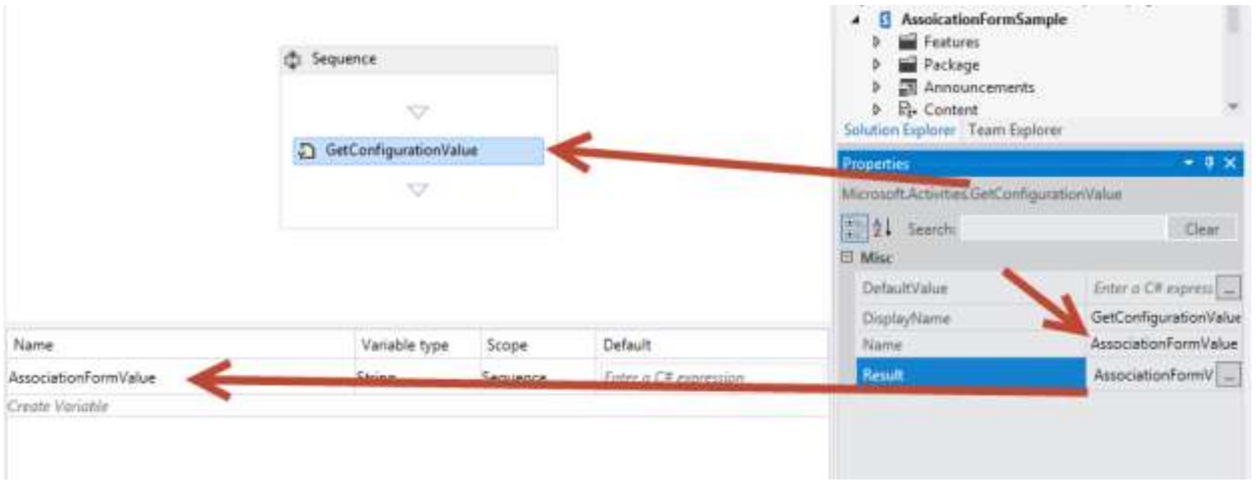


Name	Variable type	Scope	Default
AssociationFormValue	String	Sequence	Enter a C# expression

Create Variable

Drag and drop a **GetConfigurationValue** activity on the workflow designer surface and set the **Name**property to the name of the metadata property used in the form, as shown in Figure 4.

Figure 4.



Name	Variable type	Scope	Default
AssociationFormValue	String	Sequence	Enter a C# expression

DefaultValue	Enter a C# express
DisplayName	GetConfigurationValue
Name	AssociationFormValue
Result	AssociationFormV

1. Set the **Result** property to the name of the variable, as shown in Figure 4.

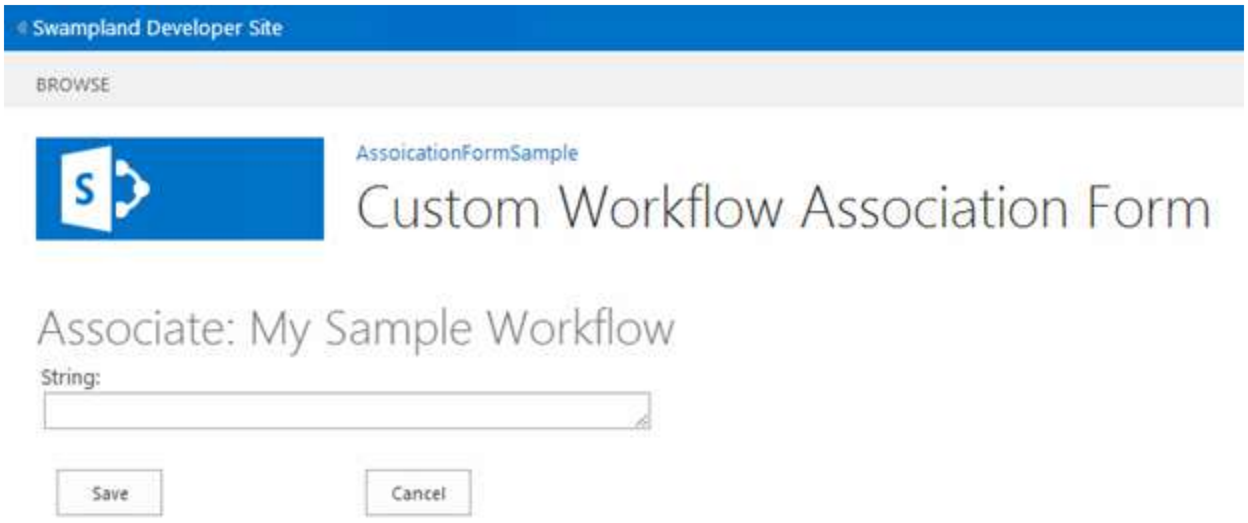
   This activity pulls the **AssociationFormValue** property value out of the workflow's metadata and stores it in the local variable. To see the contents of the variable, add a **WriteToHistory** activity to the workflow and set its **Message** property to write the value of the variable to the history list.

2. You have completed the steps necessary to associate the form values with the workflow. Save your work and test the form.

Test the custom association form

1. To test the workflow, press **F5**, or click the **Start** button in Visual Studio. This walkthrough presumes an on-premises, local installation of SharePoint, so Visual Studio launches the Workflow Manager Test Service Host utility and deploys the workflow to the developer site.
2. Create the association by navigating to the **Announcements** list, then on the ribbon select the **List** tab and click the **Workflow Settings**, **Workflow Settings** button, then click the **Add a workflow** link. At this point you are presented with the SharePoint association form.
3. In the association form, select the workflow you wish to test and give it a name.
4. Opt to create new task and history lists, set the workflow to start manually, and then click **Next**.
5. Because you have specified a custom association form in the workflow definition, the custom association form shown in Figure 5 opens.

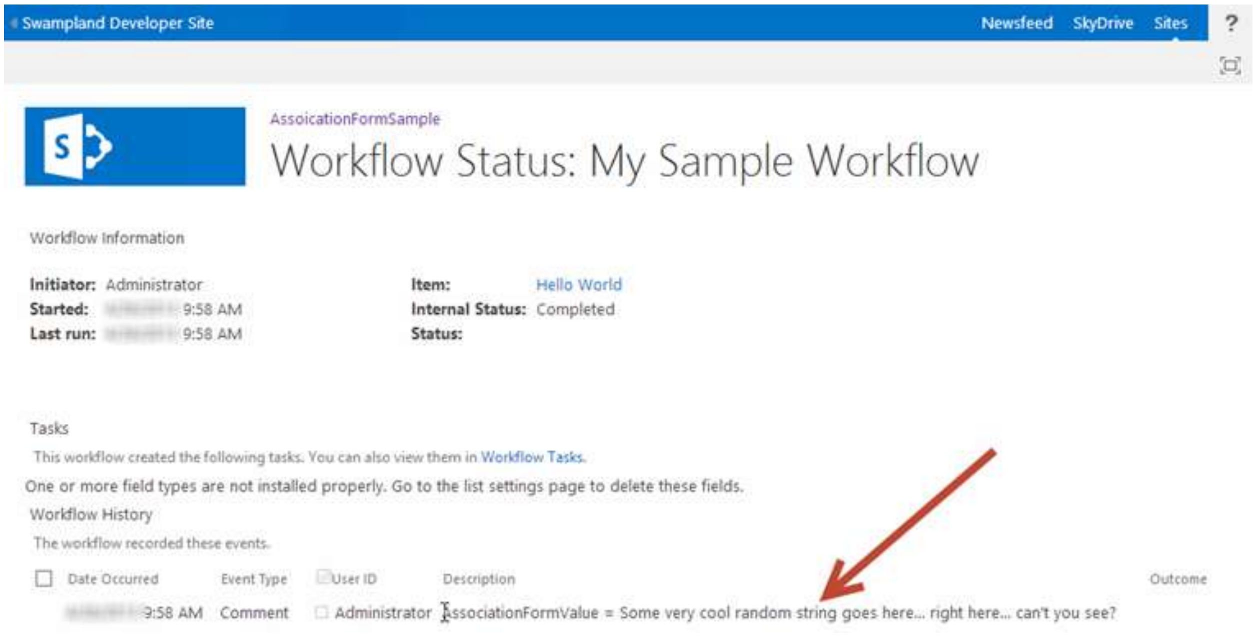
Figure 5. The custom workflow association form.



1. Enter a value in the form field and click **Save**. This creates the association and stores the custom value in the metadata for the workflow association.
2. To verify the workflow can extract the value from the configuration settings, navigate back to the **Announcements** list and create a new item. After creating the item, start the custom workflow manually. Once the workflow has started, navigate to the item's workflow

instance status page and confirm that the value that has been written to the history list, as illustrated in Figure 6.

Figure 6. Workflow status page.


Swampland Developer Site

Newsfeed SkyDrive Sites ?

AssoicationFormSample

Workflow Status: My Sample Workflow

Workflow Information

Initiator: Administrator
Started: [redacted] 9:58 AM
Last run: [redacted] 9:58 AM

Item: Hello World
Internal Status: Completed
Status:

Tasks

This workflow created the following tasks. You can also view them in Workflow Tasks.

One or more field types are not installed properly. Go to the list settings page to delete these fields.

Workflow History

The workflow recorded these events.

Date Occurred	Event Type	User ID	Description	Outcome
[redacted] 9:58 AM	Comment	Administrator	AssociationFormValue = Some very cool random string goes here... right here... can't you see?	

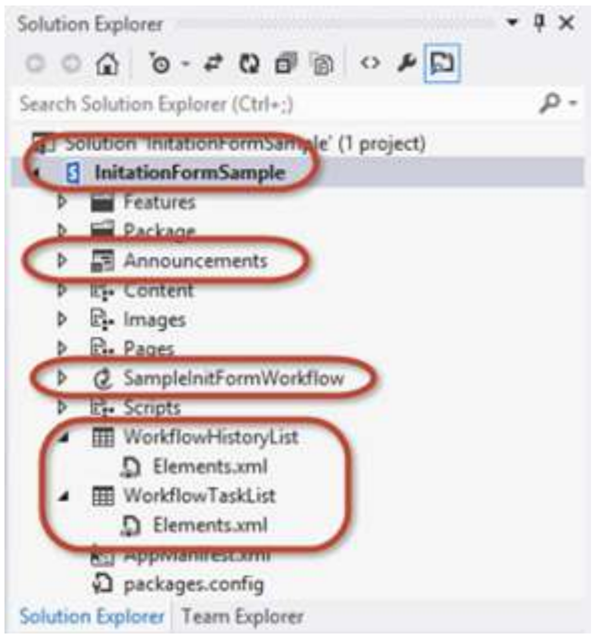
Walkthrough: Creating a custom initiation form

This walkthrough demonstrates creating a custom association form and using it to collect information from the user when the workflow is started manually.

Create a new workflow project

1. Start by creating a custom workflow using Visual Studio 2012, ensuring you have access to a SharePoint developer site.
2. Create a new SharePoint project that is configured as a SharePoint-hosted app.
3. Add a new **Announcement** list to the project. We'll use this list as the container for items that we use to trigger the workflow.
4. Next, add a workflow item to the project by right-clicking on the project icon in **Solution Explorer** and selecting **Add**, **New Item**, and then, in the **Add New Item** dialog box, selecting the **Workflow** project item. and
5. Name the new workflow "SampleInitFormWorkflow", then click **Next**.
6. When prompted, set the new workflow item to be a **List Workflow** associated with the **Announcements**list; set the workflow to start manually. (Note that the initiation form will not be displayed if the workflow starts automatically.)
7. At this point, the project appears in **Solution Explorer** as shown in Figure 7. Note that some elements, like the **WorkflowHistoryList** and **WorkflowTaskList**, were added automatically when the association was created.

Figure 7. Appearance of the project in Solution Explorer.

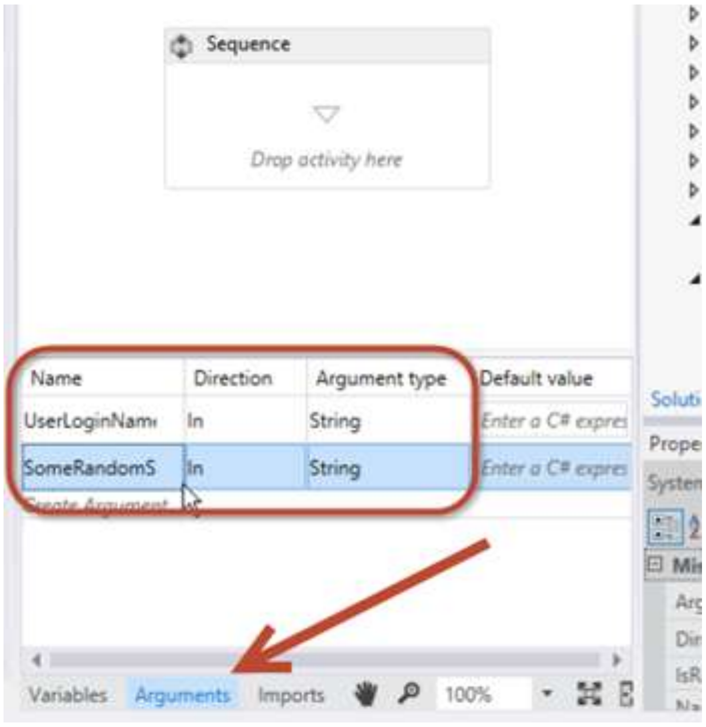


Add arguments to collect initiation form values

The workflow initiation form prompts users for two pieces of information that it needs for the workflow to start: a random string, plus a user that is selected using the people picker control. To enable this, you configure two arguments whose values the Workflow Services CSOM API will take from the form when it is submitted.

1. In the workflow designer, click the **Arguments** tab at the bottom of the screen and create two arguments, as shown in Figure 8.
   Name them **UserLoginName** and **SomeRandomString**.

   Figure 8. Configuring the initiation form arguments.



Name	Direction	Argument type	Default value
UserLoginNam	In	String	Enter a C# expres
SomeRandomS	In	String	Enter a C# expres
Create Argument			

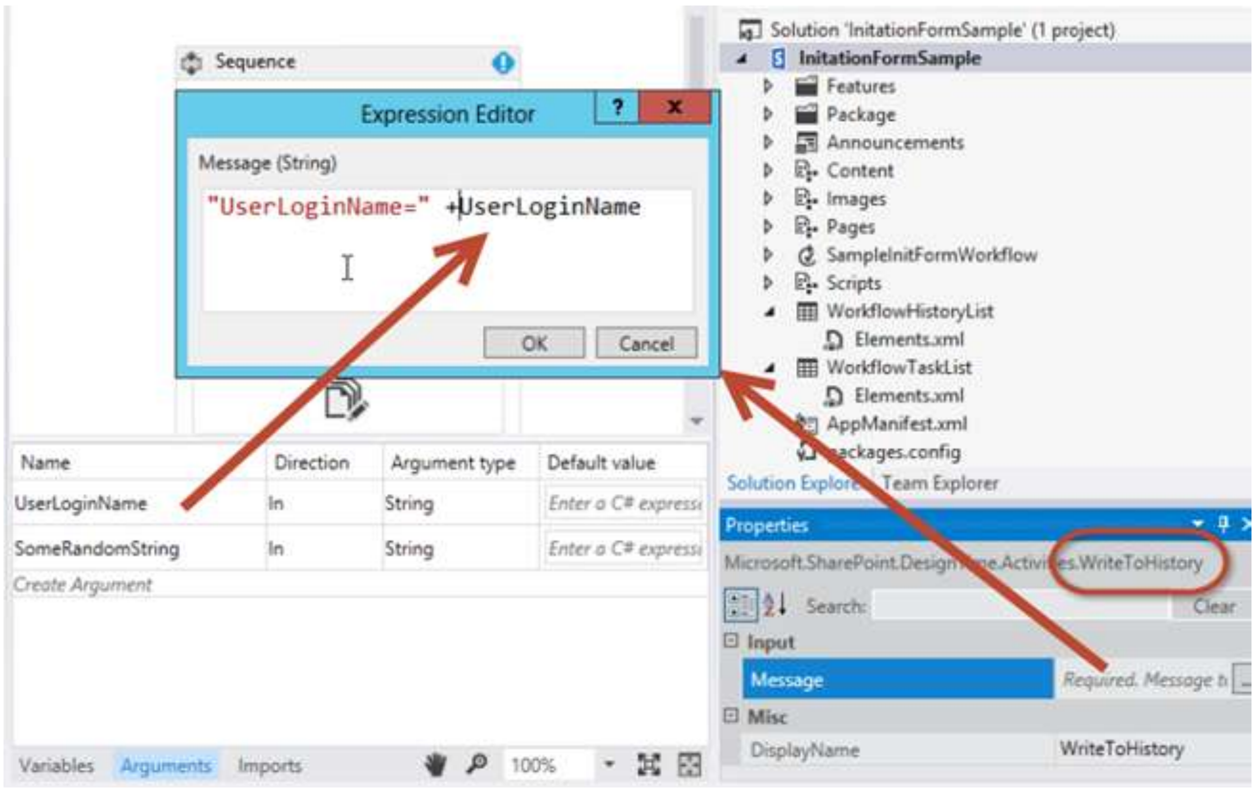
1. Set the **Argument type** to **String** for both; also for both, set the **Direction** to **In**, as shown in Figure 8.

   You can think of the **Direction** property as if it were a property on a .NET class. When the direction is set to **In**, the property would have a public **Set** method, but a private **Get** method. When the direction is set to **Out**, the property would have a public **Get** but a private **Set**. Finally, when set to **In/Out**, both the **Get**and **Set** methods would be public.

2. To view the contents of these two arguments, add a pair of **WriteToHistory** activities to the workflow and configure each one to write the contents of the arguments to the history list.

   You can use these arguments the same way you use variables, but keep in mind when setting **Direction**that you are dictating their read/write capability. Figure 9 shows what one of these activities might look like when configured:

Figure 9. Configuring a WriteToHistory activity to test arguments



Name	Direction	Argument type	Default value
UserLoginName	In	String	Enter a C# express
SomeRandomString	In	String	Enter a C# express
Create Argument			

Add the initiation form project item

With the workflow configured to accept two input arguments from the form, the next step is to add the form to the project.

1. Right-click the workflow item in **Solution Explorer** and select **Add**, then select **New Item**.
2. Select the **Workflow Initiation Form** project item and remove the number "1" from the name so the form is called "InitForm.aspx", then click **Add**. This causes Visual Studio to add the new ASPX page to the **Pages** module that is already present in the project tree. This ensures that the form will be provisioned to the **Pages** subfolder in the app. Visual Studio also modified properties on the workflow item.
3. Select the workflow item **SimpleInitFormWorkflow** in **Solution Explorer** and in the **Properties** grid, notice which properties on the initiation form have been set. One of them is actually pointing to the site relative path of the form that was just added to the **Pages** module.

Review and update the default initiation form

When you added the new initiation form to the project, Visual Studio 2012 automatically. As was the case with the association form, this new initiation form needs two tasks performed:

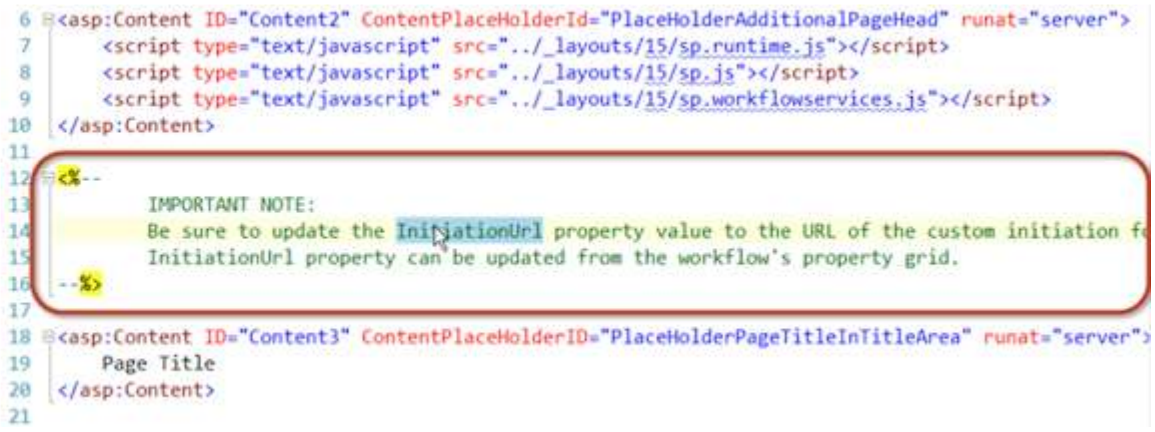
- Update the HTML form to specify data elements that should be collected from the user, as well as specifying how the form should be presented.

- Update the default JavaScript block to pull the user-input values from the form and match the property names to the names of the arguments that we created in the workflow.

Note

In the source file, note that first ASP.NET content placeholder, **PlaceHolderAdditionalPageHead**, contains references to the Workflow Services CSOM library (**sp.workflowservices.js**) and the core SharePoint CSOM libraries (**sp.js** and **sp.runtime.js**). Immediately following this section of code, there is a server-side comment that contains a commented note. Be mindful that this note, shown in Figure 10, is irrelevant and should be ignored.

Figure 10. Code comment to disregard.



```
<asp:Content ID="Content2" ContentPlaceHolderId="PlaceHolderAdditionalPageHead" runat="server">
    <script type="text/javascript" src="../_layouts/15/sp.runtime.js"></script>
    <script type="text/javascript" src="../_layouts/15/sp.js"></script>
    <script type="text/javascript" src="../_layouts/15/sp.workflowservices.js"></script>
</asp:Content>

<%--
        IMPORTANT NOTE:
        Be sure to update the InitiationUrl property value to the URL of the custom initiation f
        InitiationUrl property can be updated from the workflow's property grid.
--%>

<asp:Content ID="Content3" ContentPlaceHolderID="PlaceHolderPageTitleInTitleArea" runat="server"
    Page Title
</asp:Content>
```

Update the HTML form

1. Scroll down in the form's code file until you reach the ASP.NET content placeholder namedPlaceHolderMain. Notice that the first part of this section contains an HTML table that contains three form fields. We only need two.
2. Update this HTML table by replacing it with the following:

Copy

```
<table>
  <tr>
    <td>
      String:<br />
      <input type="text" id="strInput" />
    </td>
  </tr>
  <tr>
    <td>
      User Picker:<br />
```

```
     <SharePoint:PeopleEditor AllowEmpty="false" ValidatorEnabled="true" MultiSelect="false" ID="peoplePicker" runat="server" />
    </td>
  </tr>
  <tr>
    <td>
     <input type="button" name="startWorkflowButton" value="Start" onclick="StartWorkflow()" />
     <input type="button" name="cancelButton" value="Cancel" onclick="RedirFromInitForm()" />
     <br />
    </td>
  </tr>
</table>
```

The table now contains two input controls. The first is a standard HTML text box whose ID is **strInput**. The second is a SharePoint people picker control whose ID is **peoplePicker**. This latter is a server-side control; however, it is allowed on the page because it has been deployed to every SharePoint computer. Further, the control is referenced at the top of the initiation form.

Now notice the two buttons on the form, **Start** ("startWorkflowButton") and **Cancel** ("cancelButton"). Clicking the Start button calls the **StartWorkflow()**JavaScript function. The function itself is located in a script block farther down in the form file and is the subject of the next change that we need to make.

Update the JSOM code block to start the workflow

1. Locate the JavaScript code block that immediately follows the HTML table that we just modified. We'll leave the code in the script block largely unchanged.

   This code demonstrates enormous power and flexibility available in the Workflow Services JavaScript Client Side Object Model (JSOM) API that has been implemented in SharePoint. From a high level, the code performs the following steps.

2. Locate the following line of code: var wfParams = new Object();
3. Immediately following this line, replace the existing code with your own code to pull values from the two HTML form fields that we created a few moments ago: **strInput** and **peoplePicker**. To do this, add the following reference to the jQuery library to the **PlaceHolderAdditionalPageHead** element in the HTML markup. This makes it easier to grab values from the form.

Copy

```
<script type="text/javascript" src="../Scripts/jquery-1.8.2.min.js"></script>
```

1. Now, return to the portion of the JSOM script block where the parameters are collected from the form. Replace the existing JavaScript with the following code:

Copy

```
  var wfParams = new Object();
// get people picker value
var html = $("ctl00_PlaceHolderMain_peoplePicker_upLevelDiv");
wfParams['UserLoginName'] = $("#divEntityData", html).attr("key");

// get string input
var strInputValue = $("strInput").value;
wfParams['SomeRandomString'] = strInputValue
```

1. Save your changes.

The code that we've modified in the preceding procedure does a large amount of work:

- Obtains these three parameters from the query string in the URL:
  - The item ID that the new workflow instance will be associated with, saved in the JavaScript variable **[itemId]**.
  - The ID of the workflow association on the current site, list, or document library, saved in the JavaScript variable **[subscriptionId]**.
  - The URL that the user came from, saved in the JavaScript **[redirectUrl]** variable. This is where the user will be taken to when they complete the form and the workflow has been started.
- Creates an array of properties to be sent to the workflow, saved in the JavaScript variable **[wfParams]**. These are the values that you will need to collect from the form and the second of the two minimal steps required for editing the custom initiation form.
- Obtains references to a SharePoint CSOM client context as well as the necessary workflow services.
- Once the script is connected to the Workflow Services Subscription Service (referenced in the variable **[subscriptionService]**), it performs one of the following tasks:
  -
  - If the script obtained an item ID from the query string in the first step, then it starts a new instance of the workflow on the specified list item by calling the function **[startWorkflowOnListItem()]** from the Workflow Services instance service.
  - If no ID was found, it starts a new instance of the workflow on the current site by calling the function **[startWorkflow()]** from the Workflow Services instance service.

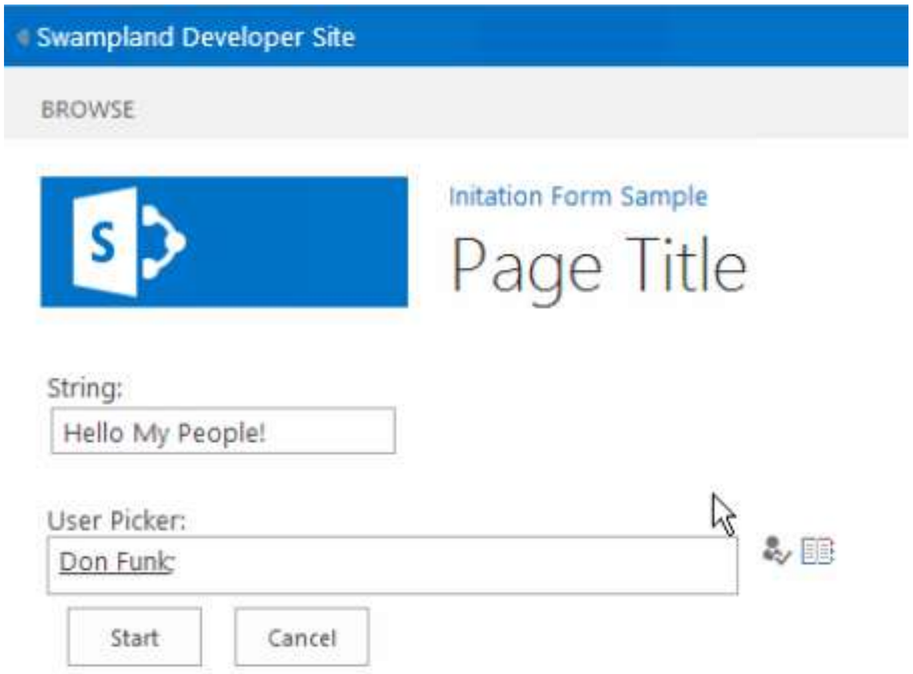
Output:-
Test the custom initiation form

Test the workflow by pressing F5 or by clicking the **Start** button in Visual Studio 2012. If you are testing in an on-premises local installation of SharePoint, Visual Studio 2012 will start the Workflow Manager Test Service Host utility and deploy the workflow to the developer site. After a moment, the developer site will open.

Navigate to the **Announcements** list and create a new item. After creating the item, start the custom workflow.

Because the workflow definition contains a reference to an initiation form, the user is taken to that form first. Fill in the requested values, then click the **Start** button. This triggers the JavaScript on the page, which starts the workflow instance, as shown in Figure 11.

Figure 11. Triggering the workflow.

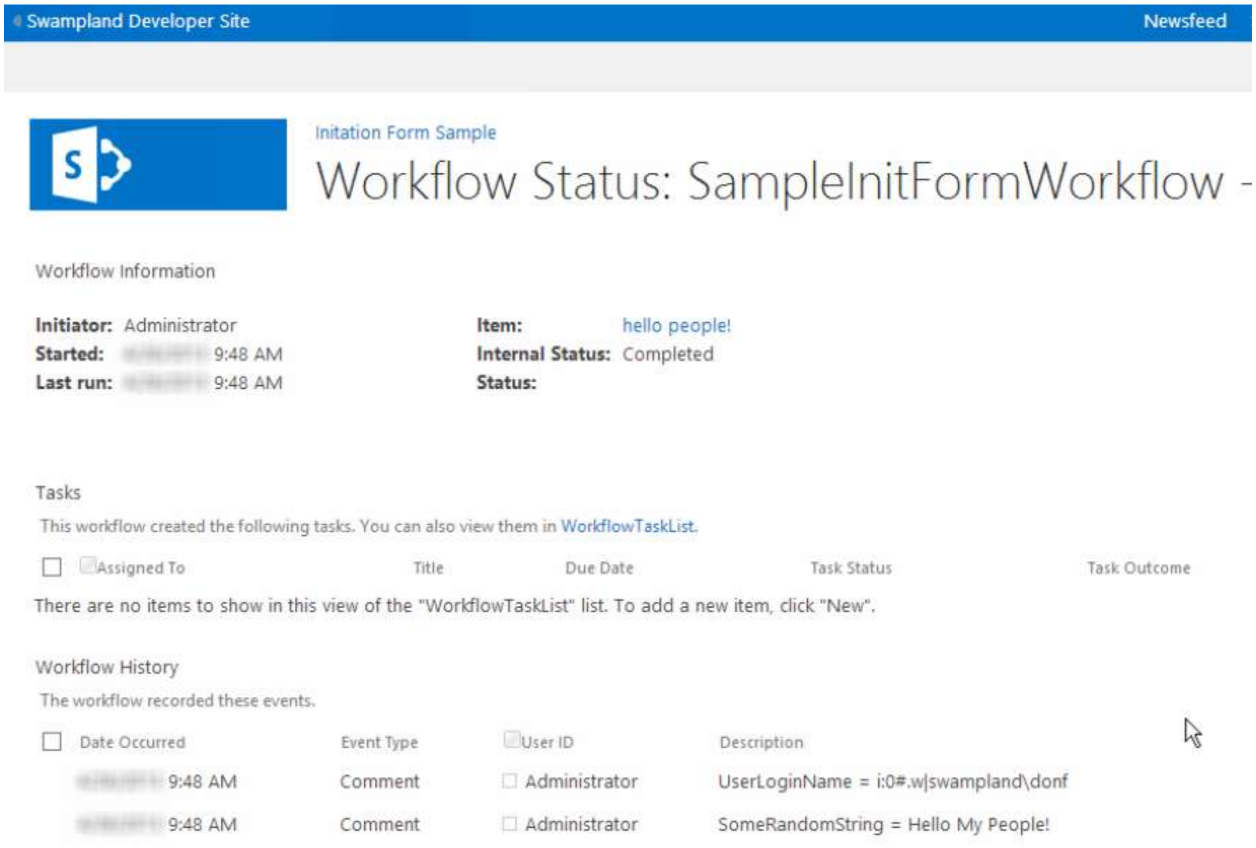


String:
Hello My People!

User Picker:
Don Funk

After starting the workflow, the page redirects the user back to the originating page. Give the workflow a few moments to start, then go back to the item and view the workflow instance's status page. Notice that the history list contains the values that were submitted in the form and then sent to SharePoint once the workflow instance was created using the Workflow Services JSOM.

Figure 12. On completing the workflow.



Swampland Developer Site

Newsfeed

Initation Form Sample

Workflow Status: SampleInitFormWorkflow -

Workflow Information

Initiator:	Administrator	**Item:**	hello people!
Started:	9:48 AM	**Internal Status:**	Completed
Last run:	9:48 AM	**Status:**	

Tasks

This workflow created the following tasks. You can also view them in WorkflowTaskList.

Assigned To	Title	Due Date	Task Status	Task Outcome

There are no items to show in this view of the "WorkflowTaskList" list. To add a new item, click "New".

Workflow History

The workflow recorded these events.

Date Occurred	Event Type	User ID	Description
9:48 AM	Comment	Administrator	UserLoginName = i:0#.w|swampland\donf
9:48 AM	Comment	Administrator	SomeRandomString = Hello My People!

Experiment No:-8

Aim:- Create a simple calculator using sharepoint workflow.

Objective:-

- Calculator must possess the following function buttons
- +, _, *, /,log, sqr, cube, sqrt, cube root, mod, %, sin, cos, tan,cancel, 1/x, pie

Theory:-

Steps to create windows application

-In Visual Studio, select File|New|Project. Visual studio will display the new project dialog box.

-In the new project dialog box, click the windows Application icon. In the name field, type a project name that describes the program you are binding, such as Demo program. Then, in the location field, type the name of the folder in which you want visual studio to place the projects folder and files. Click ok. Visual studio will display a design window where you can drag and drop controls onto your form.

-To display the toolbox that contains the control you can drag and drop on to your form, select View|Toolbox.Visual studio will open the Toolbox window.

-In the toolbox window, locate the various button control and textbox control

-In the property window give the text to the button control

-As mentioned above the calculator must possess the buttons

-Use operators and select case statements to execute the programme.

-To run the programme select the debug menu and then press start

Steps to create traffic signal windows application

-In Visual Studio, select File|New|Project. Visual studio will display the new project dialog box.

- In the new project dialog box, click the windows Application icon. In the name field, type a project name that describes the program you are binding, such as Demo program. Then, in the location field, type the name of the folder in which you want visual studio to place the projects folder and files. Click ok. Visual studio will display a design window where you can drag and drop controls onto your form.

- To display the toolbox that contains the control you can drag and drop on to your form, select View|Toolbox.Visual studio will open the Toolbox window.

- Design the Diagram of Square with signals on each terminals

-In the property window set the textbox property backcolor using colors red,blue and green as the colors of the signal

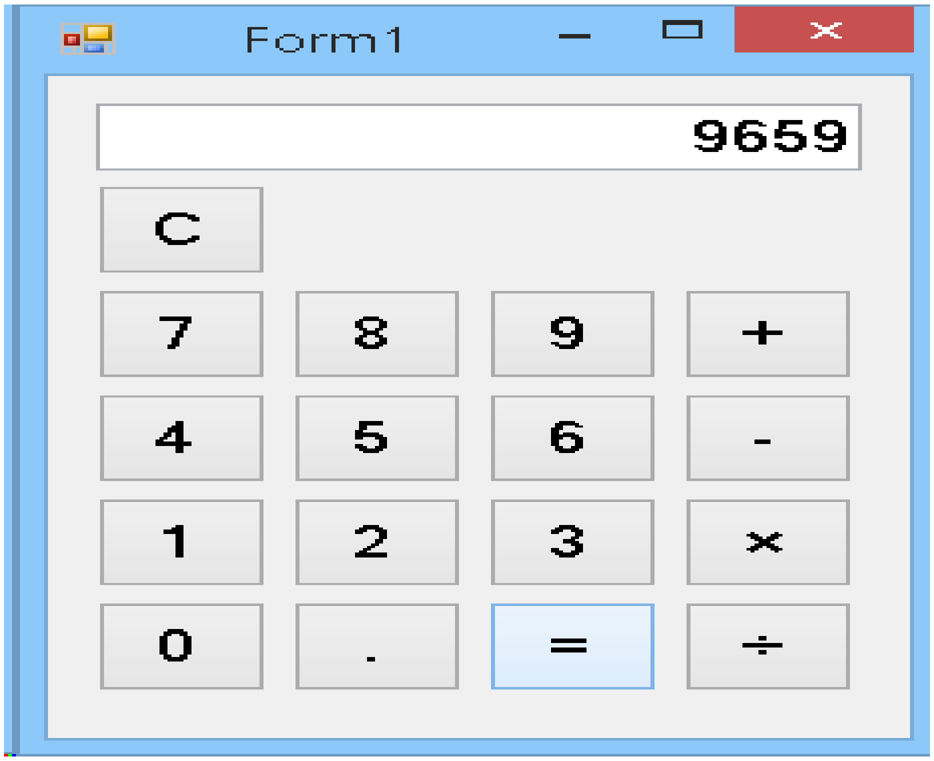
- Drag and drop the timer control. Set the properties enable to true and set interval.
-To run the programme select the debug menu and then press start.
Code:

```
/* Design a Calculator application*/

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace SimpleCalculator
{
   public partial class frmCalculator : Form
   {
      string operand1 = string.Empty;
      string operand2 = string.Empty;
      string result;
      char operation;

      public frmCalculator()
      {
         InitializeComponent();
      }

      private void frmCalculator_Load(object sender, EventArgs e)
      {
         btnOne.Click += new EventHandler(btn_Click);
         btnTwo.Click += new EventHandler(btn_Click);
         btnThree.Click += new EventHandler(btn_Click);
         btnFour.Click += new EventHandler(btn_Click);
         btnFive.Click += new EventHandler(btn_Click);
         btnSix.Click += new EventHandler(btn_Click);
         btnSeven.Click += new EventHandler(btn_Click);
         btnEight.Click += new EventHandler(btn_Click);
```

```
    btnNine.Click += new EventHandler(btn_Click);
    btnZero.Click += new EventHandler(btn_Click);
    btnDot.Click += new EventHandler(btn_Click);
}

void btn_Click(object sender, EventArgs e)
{
    try
    {
        Button btn = sender as Button;

        switch (btn.Name)
        {
            case "btnOne":
                txtInput.Text += "1";
                break;
            case "btnTwo":
                txtInput.Text += "2";
                break;
            case "btnThree":
                txtInput.Text += "3";
                break;
            case "btnFour":
                txtInput.Text += "4";
                break;
            case "btnFive":
                txtInput.Text += "5";
                break;
            case "btnSix":
                txtInput.Text += "6";
                break;
            case "btnSeven":
                txtInput.Text += "7";
                break;
            case "btnEight":
                txtInput.Text += "8";
                break;
            case "btnNine":
                txtInput.Text += "9";
                break;
                case "btnZero":
                    txtInput.Text += "0";
                    break;
                case "btnDot":
                    if(!txtInput.Text.Contains("."))
                        txtInput.Text += ".";
                    break;

            }
        }
        catch(Exception ex)
        {
            MessageBox.Show("Sorry for the inconvenience, Unexpected
error occured. Details: " +
                ex.Message);
        }
    }

    private void txtInput_KeyPress(object sender, KeyPressEventArgs e)
    {
        switch (e.KeyChar)
        {
            case '1':
            case '2':
            case '3':
            case '4':
            case '5':
            case '6':
            case '7':
            case '8':
            case '9':
            case '0':
            //case '+':
            //case '-':
            //case '*':
            //case '/':
            //case '.':
                break;
            default:
                e.Handled = true;
            MessageBox.Show("Only numbers, +, -, ., *, / are allowed");
            break;
    }
}

private void txtInput_TextChanged(object sender, EventArgs e)
{

}

private void btnPlus_Click(object sender, EventArgs e)
{
    operand1 = txtInput.Text;
    operation = '+';
    txtInput.Text = string.Empty;
}

private void btnMinus_Click(object sender, EventArgs e)
{
    operand1 = txtInput.Text;
    operation = '-';
    txtInput.Text = string.Empty;
}

private void btnMulitply_Click(object sender, EventArgs e)
{
    operand1 = txtInput.Text;
    operation = '*';
    txtInput.Text = string.Empty;
}

private void btnDivide_Click(object sender, EventArgs e)
{
    operand1 = txtInput.Text;
    operation = '/';
    txtInput.Text = string.Empty;
}

private void btnEqual_Click(object sender, EventArgs e)
{
    operand2 = txtInput.Text;

    double opr1, opr2;
    double.TryParse(operand1, out opr1);
    double.TryParse(operand2, out opr2);

    switch (operation)
    {
        case '+':
            result = (opr1 + opr2).ToString();
            break;

        case '-':
            result = (opr1 - opr2).ToString();
            break;

        case '*':
            result = (opr1 * opr2).ToString();
            break;

        case '/':
            if (opr2 != 0)
            {
                result = (opr1 / opr2).ToString();
            }
            else
            {
                MessageBox.Show("Can't divide by zero");
            }
            break;
    }

    txtInput.Text = result.ToString();
}

private void btnClear_Click(object sender, EventArgs e)
{
    txtInput.Text = string.Empty;
    operand1 = string.Empty;
    operand2 = string.Empty;
```

```
        }

        private void btnSqrRoot_Click(object sender, EventArgs e)
        {
            double opr1;
            if (double.TryParse(txtInput.Text, out opr1))
            {
                txtInput.Text = (Math.Sqrt(opr1)).ToString();
            }
        }

        private void btnByTwo_Click(object sender, EventArgs e)
        {
            double opr1;
            if (double.TryParse(txtInput.Text, out opr1))
            {
                txtInput.Text = (opr1 / 2).ToString();
            }
        }

        private void btnByFour_Click(object sender, EventArgs e)
        {
            double opr1;
            if (double.TryParse(txtInput.Text, out opr1))
            {
                txtInput.Text = (opr1 / 4).ToString();
            }
        }
    }
}
```

Output:-



Experiment No:-9

Aim :- Create a web forms application that integrated with office 365.

Objective:-

To understand Setting up your environment
To understand Communicate with the Exchange REST API
To understand Building a sample Web Forms application that integrated with Office 365
To understand From development to production: Publish your app to Windows Azure Web Sites

Theory :-

A simple web form with a Gauge.js indicator displaying the amount of unread e-mails in our Office 365 inbox.

Add integration to your Office 365

In this step we will add code to fetch information from our Office 365 Exchange mailbox. In my sample I want to collect a few simple variables like "**Unread Mail**" and "**Total Mails**" and display this in a simple javascript Gauge for visualization awesomeness. So, with that said – let's get to it.

Once you have created your ASP.NET Web Forms application project, simply right click and select "**Add" – "Connected Service…":**

Code:

Step 1: Clone or download this repository

From your Git Shell or command line:

```
git clone https://github.com/OfficeDev/O365-WebApp-SingleTenant.git
```

Step 2: Build the Project

1. Open the project in Visual Studio 2013.
2. Simply Build the project to restore NuGet packages.
3. Ignore any build errors for now as we will configure the project in the next steps.

Step 3: Configure the sample

Once downloaded, open the sample in Visual Studio.

Register Azure AD application to consume Office 365 APIs

Office 365 applications use Azure Active Directory (Azure AD) to authenticate and authorize users and applications respectively. All users, application registrations, permissions are stored in Azure AD.

Using the Office 365 API Tool for Visual Studio you can configure your web application to consume Office 365 APIs.

1. In the Solution Explorer window, **right click your project -> Add -> Connected Service**.
2. A Services Manager dialog box will appear. Choose **Office 365 -> Office 365 API** and click **Register your app**.
3. On the sign-in dialog box, enter the username and password for your Office 365 tenant.
4. After you're signed in, you will see a list of all the services.
5. Initially, no permissions will be selected, as the app is not registered to consume any services yet.
6. Select **Users and Groups** and then click **Permissions**
7. In the **Users and Groups Permissions** dialog, select **Enable sign-on and read users profiles'** and click **Apply**
8. Select **Contacts** and then click **Permissions**
9. In the **Contacts Permissions** dialog, select **Read users' contacts** and click **Apply**
10. Click **Ok**

After clicking OK, Office 365 client libraries (in the form of NuGet packages) for connecting to Office 365 APIs will be added to your project.

In this process, Office 365 API tool registered an Azure AD Application in the Office 365 tenant that you signed in the wizard and added the Azure AD application details to web.config.

Step 4: Update web.config with your Tenant ID

There is one extra configuration required if you are building a single tenant application.

In your web.config, update the **TenantId** value to your **Office 365 tenant Id** where the application is deployed.

To get the tenant Id of your Office 365 tenant:

- Log in to your Azure Portal and select your Office 365 domain directory.

NOTE: If you are unable to login to Azure Portal using your Office 365 credentials, You can also access your Office 365's Azure Portal directly from your Office 365 Admin Center

- Now, in the browser URL, locate the GUID. This will be your Office 365 tenant Id.
- Copy and paste it in the web.config where it says "paste-your-tenant-guid-here" :

Note: If you are deploying to a production tenant, you will need to ask your tenant admin for the tenant identifier.

Step 5: Build and Debug your web application

Now you are ready for a test run. Hit F5 to test the app.

Quick Look at the Authentication Code

The authentication startup class, **App_Start/Startup.Auth.cs** in the project contains the startup logic for Azure AD authentication.

The sample now uses a persistent ADAL token cache that uses a database for its token cache. You can see the token cache implementation in the following files:

- Models/ADALTokenCache.cs
- Models/ApplicationDbContext.cs

Sign In and Sign Out Controls

The sign in and sign out controls are already added to the views. You can find them under **Views\Shared** folder.

1. **_LoginPartial.cshtml** is the partial view that renders the Sign In and Sign Out actions.
2. **_LoginPartial.cshtml** is then rendered in _Layout.cshtml
3. The **AccountController.cs** has the required methods for sign in and sign out.

Requiring authentication to access controllers

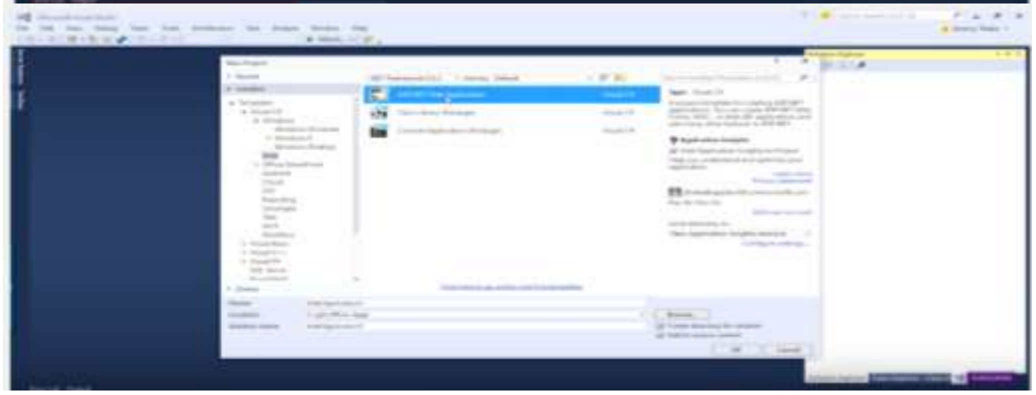
Applying **Authorize** attribute to all controllers in your project will require the user to be authenticated before accessing these controllers. To allow the controller to be accessed anonymously, remove this attribute from the controller. If you want to set the permissions at a more granular level, apply the attribute to each method that requires authorization instead of applying it to the controller class.

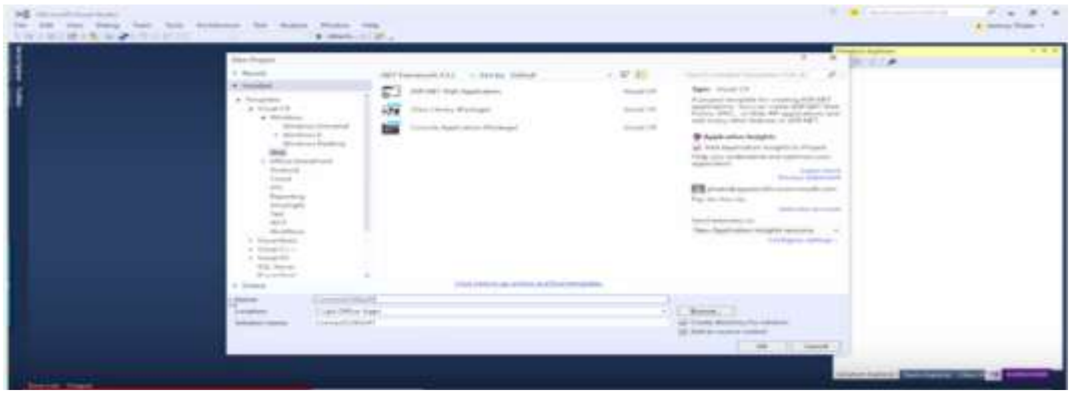
Write Code to call Office 365 APIs

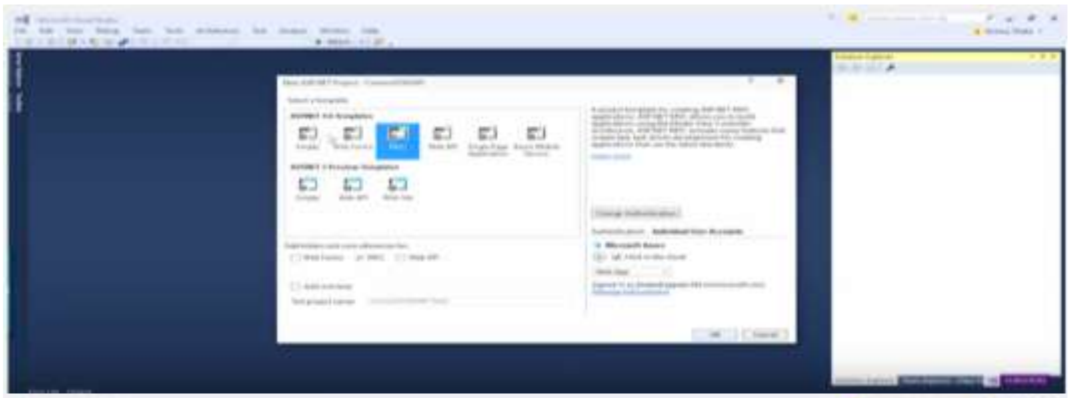
You can now write code to call an Office 365 API in your web application. You can apply the Autorize attribute to the desired controller or the method in which you wish to call Office 365 API.

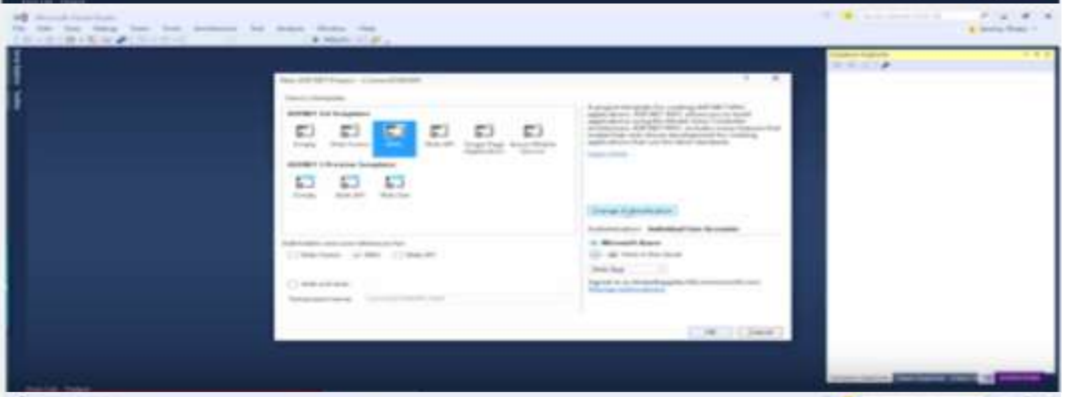
ContactsController.cs describes how to interact with the Office 365 API discovery service, get the endpoint URI and resource Id for Outlook Services and then query users' contacts.

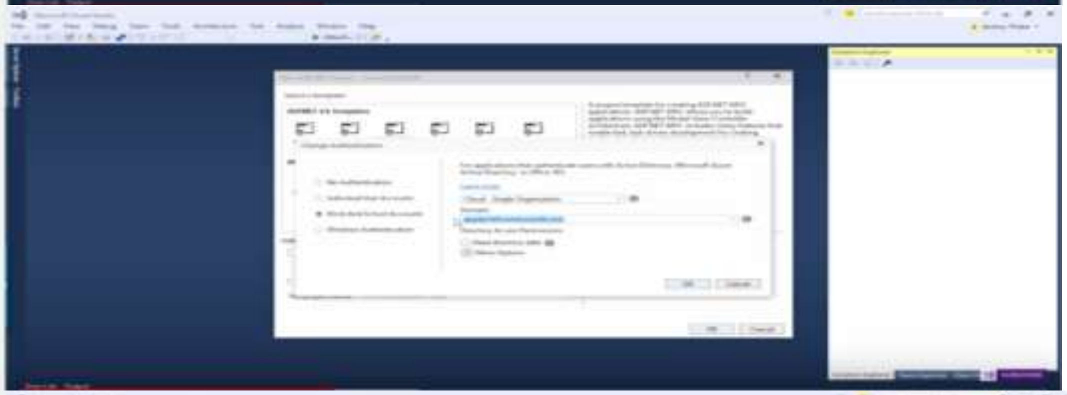
This project has adopted the Microsoft Open Source Code of Conduct. For more information, see the Code of Conduct FAQ or contact opencode@microsoft.com with any additional questions or comments.

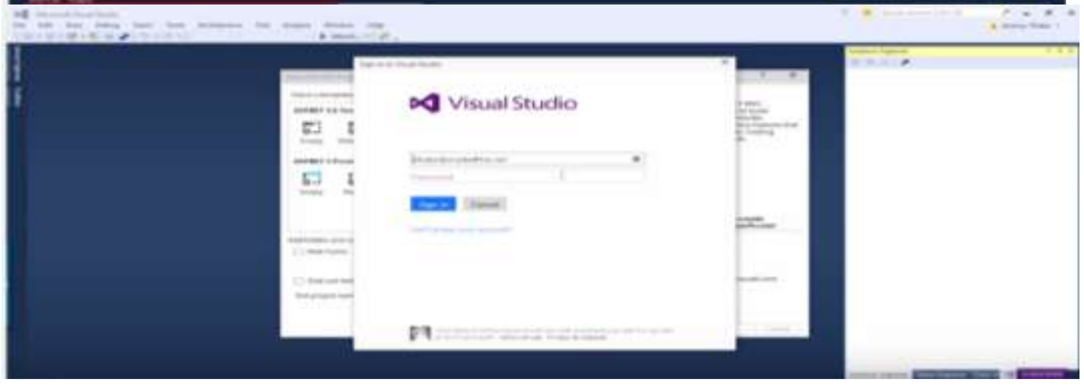


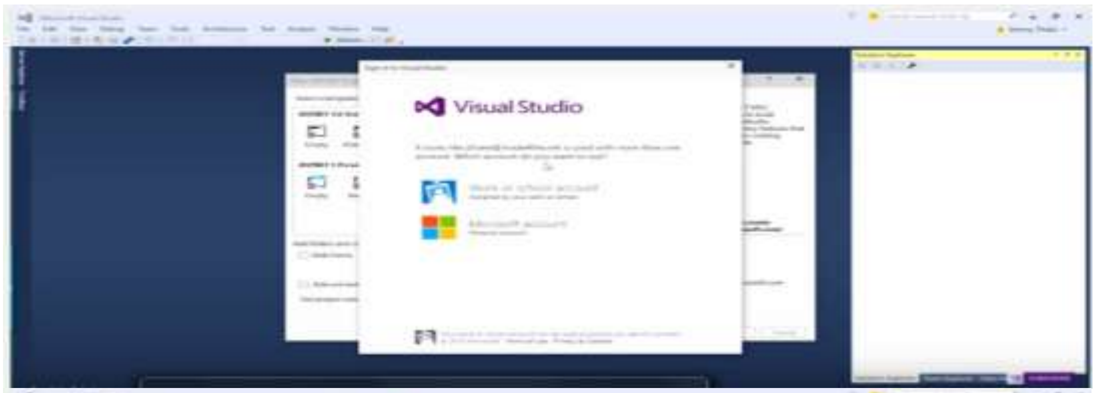


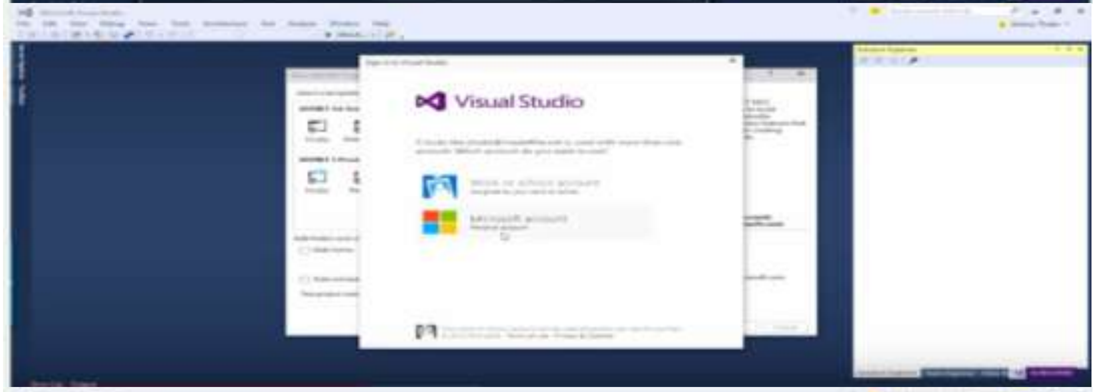


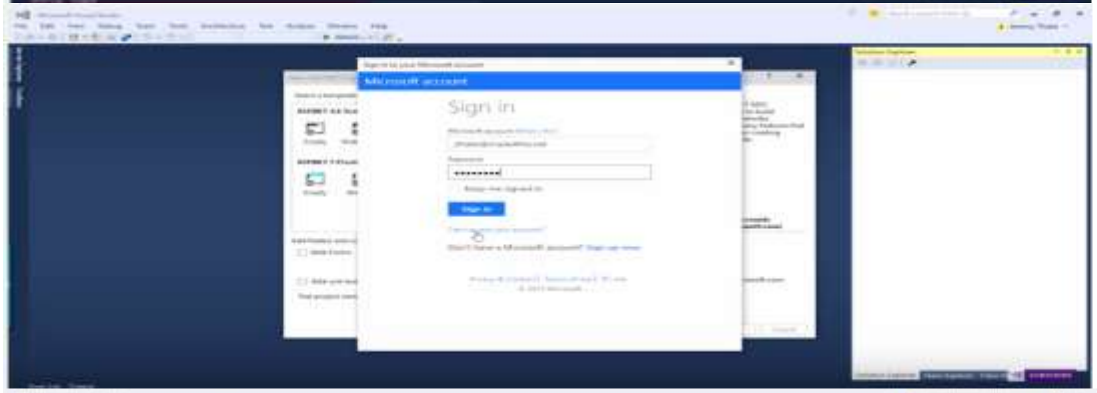


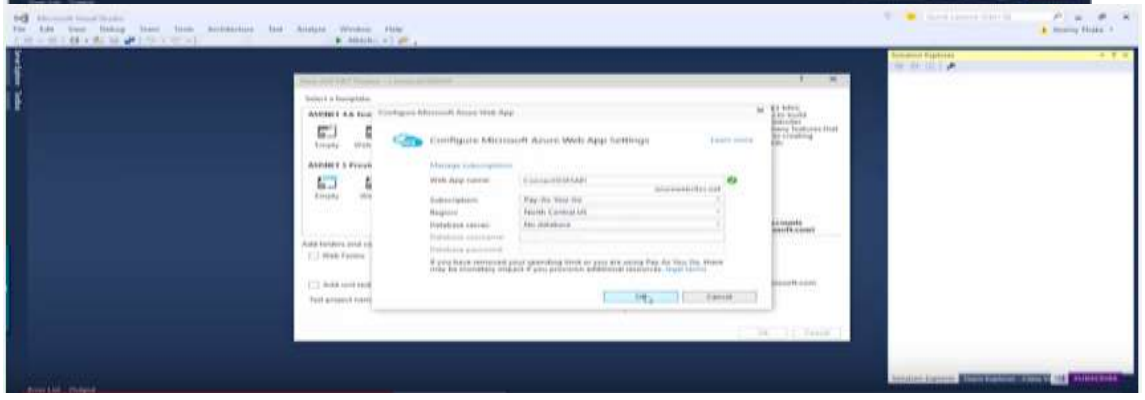


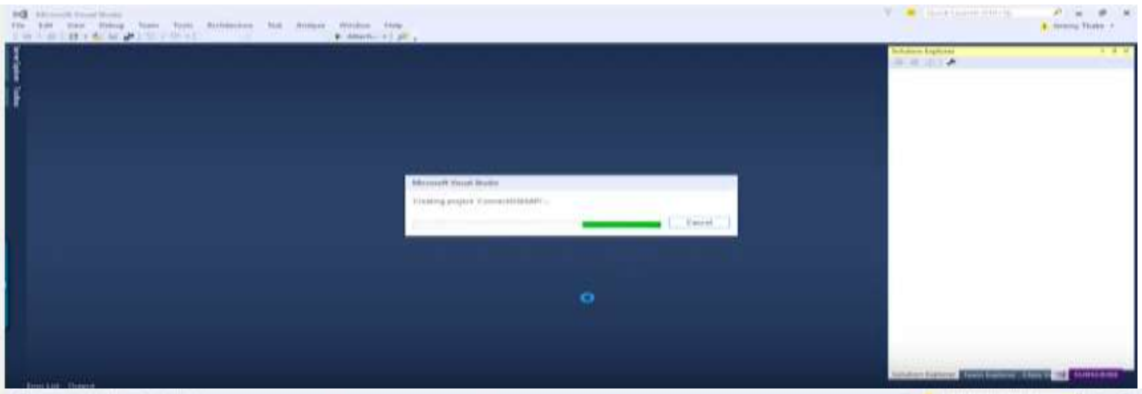

Visual Studio


Visual Studio
Microsoft account

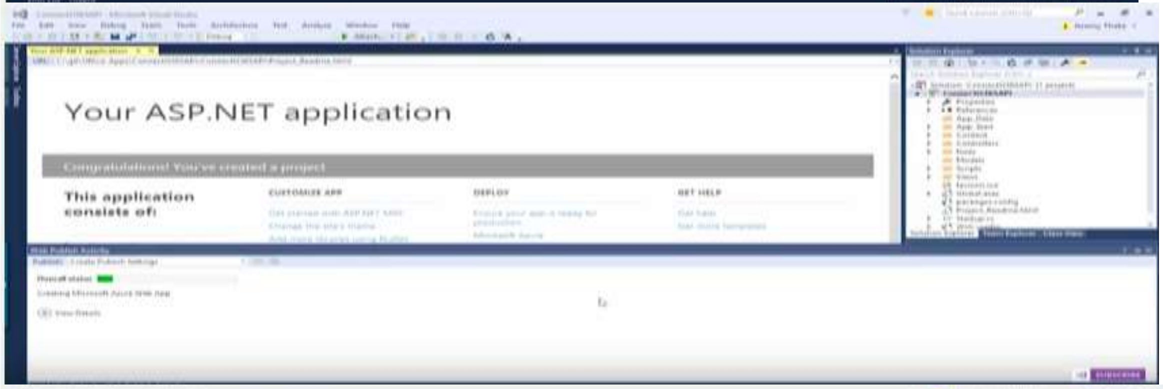

Visual Studio
Microsoft account


Microsoft account
Sign in


Configure Microsoft Azure Web App Settings
Web App name
Subscription
Region
Database server
Pay-As-You-Go
North Central US
No database

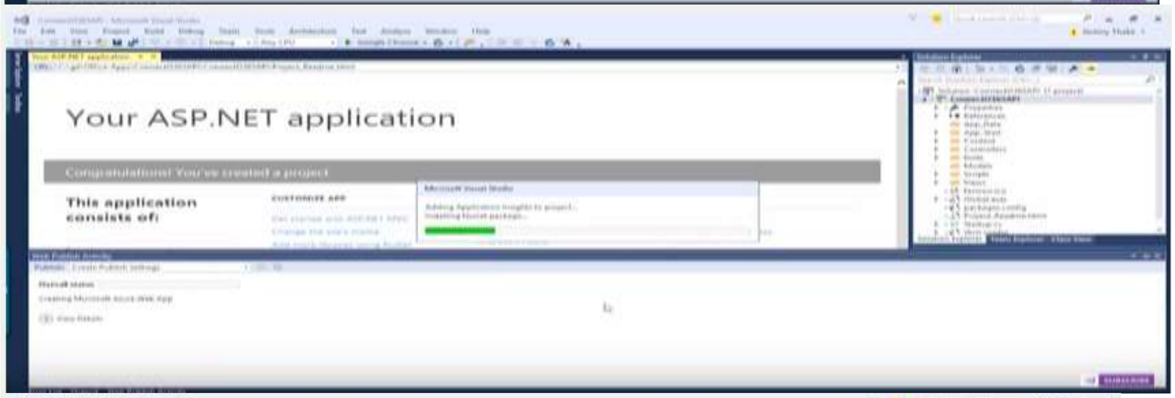

Microsoft Visual Studio

Creating project 'ConnectIOWAPI'...


Your ASP.NET application

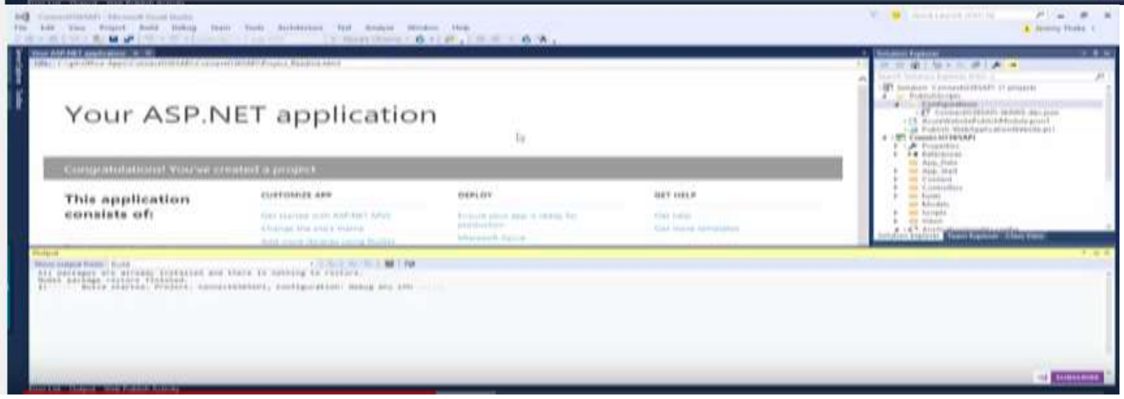
Congratulations! You've created a project

This application consists of:


Your ASP.NET application

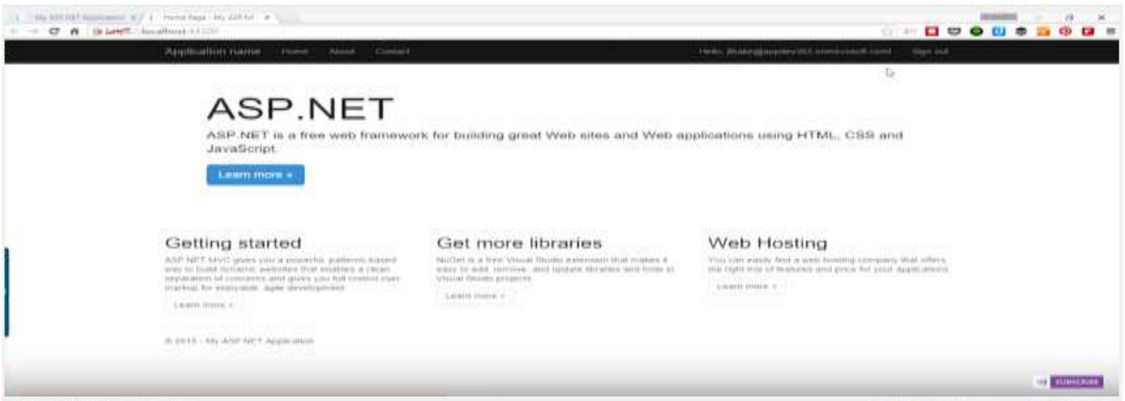
Congratulations! You've created a project

This application consists of:


Your ASP.NET application

Congratulations! You've created a project

This application consists of:



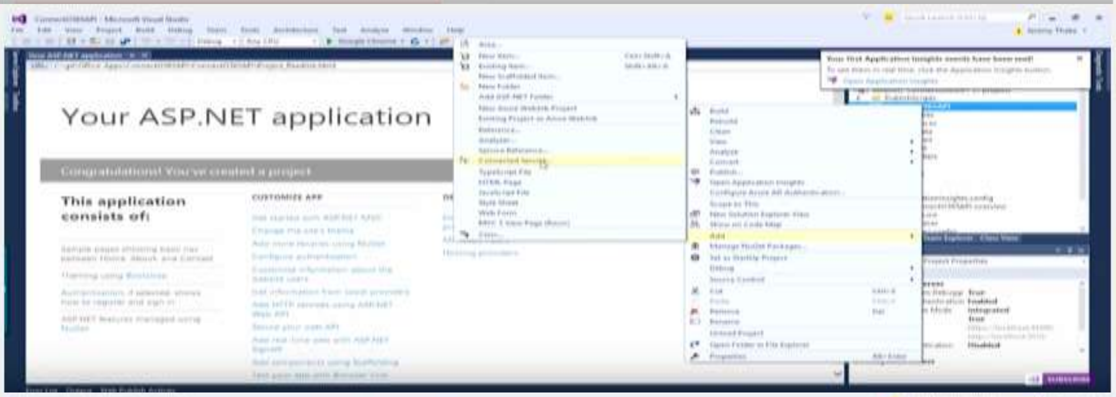
ASP.NET

ASP.NET is a free web framework for building great Web sites and Web applications using HTML, CSS and JavaScript.

Getting started

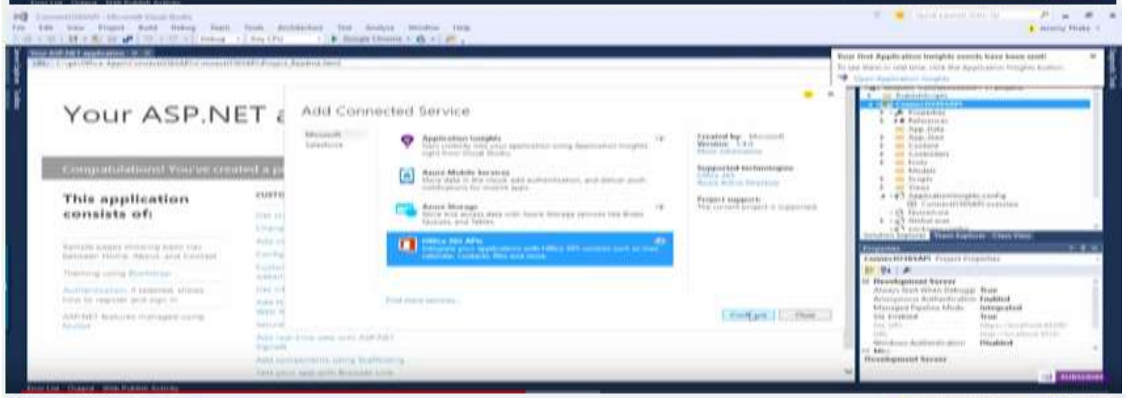
Get more libraries

Web Hosting


Your ASP.NET application

Congratulations! You've created a project

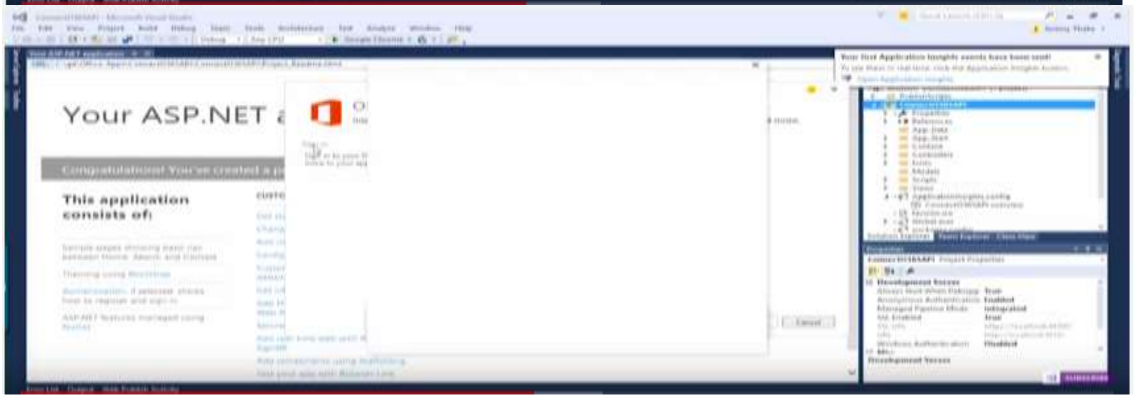
This application consists of:


Your ASP.NET a

Add Connected Service

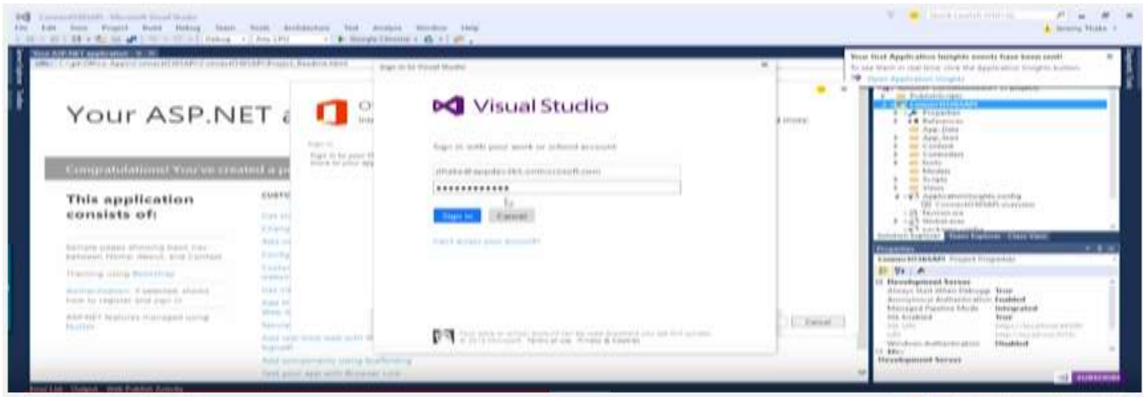
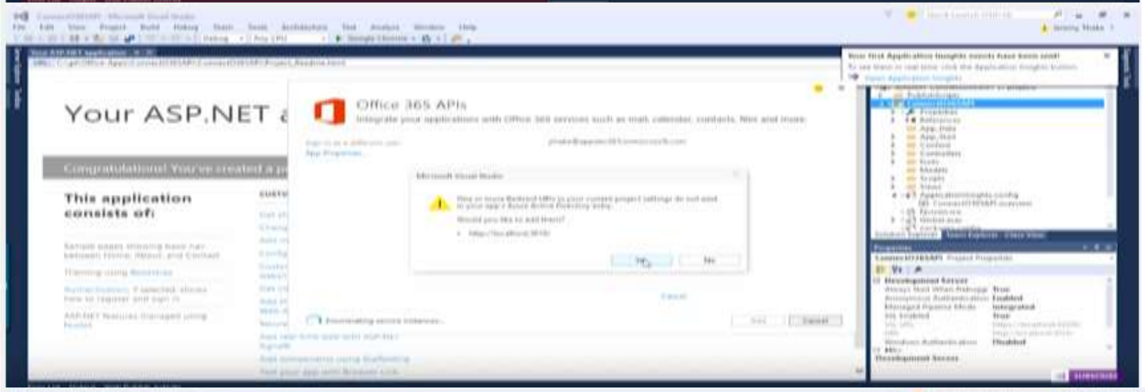
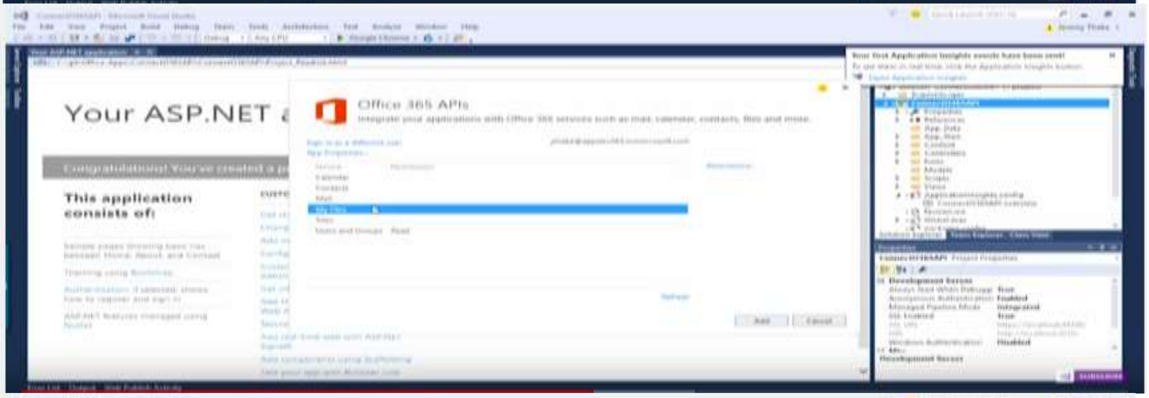
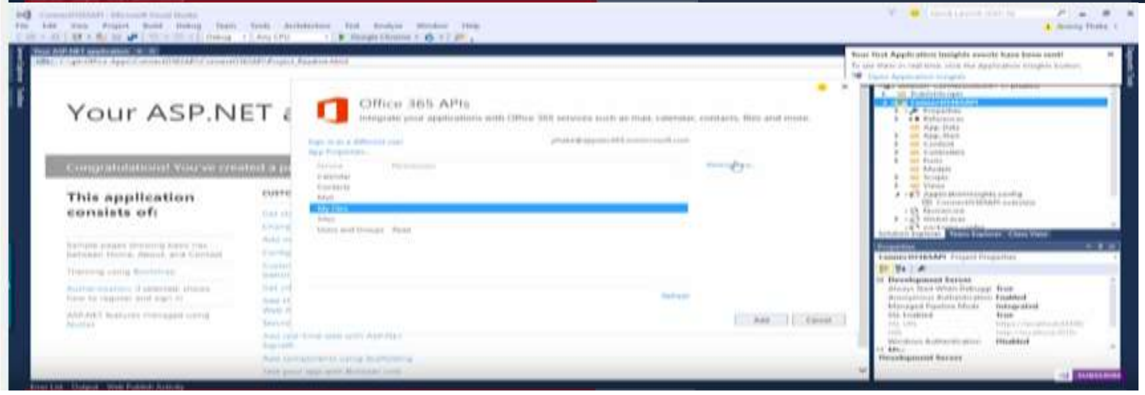
Congratulations! You've created a p

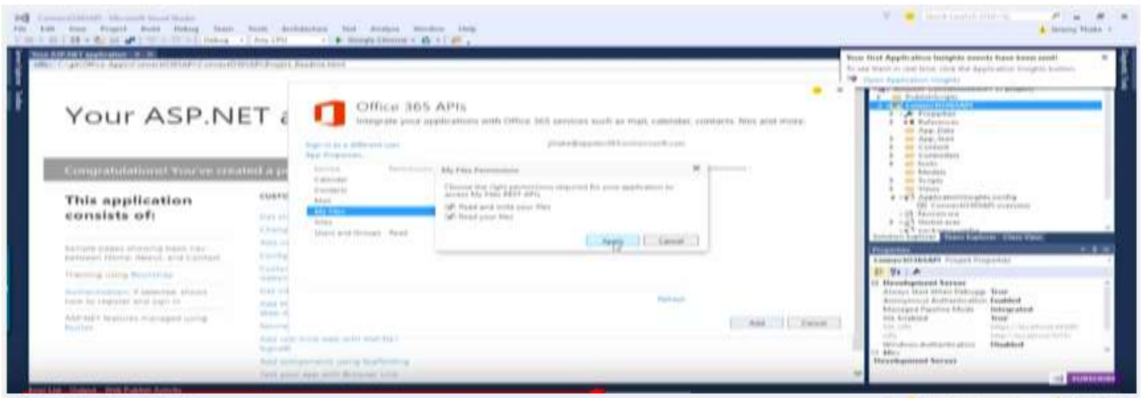
This application consists of:


Your ASP.NET a

Congratulations! You've created a p

This application consists of:


Your ASP.NET a
Congratulations! You've created a p
This application consists of:
Visual Studio
Sign in
Cancel

Your ASP.NET a
Office 365 APIs
Congratulations! You've created a p
This application consists of:
Microsoft Visual Studio
Yes
No
Cancel

Your ASP.NET a
Office 365 APIs
Congratulations! You've created a p
This application consists of:
Add
Cancel

Your ASP.NET a
Office 365 APIs
Congratulations! You've created a p
This application consists of:
Add
Cancel


Your ASP.NET a

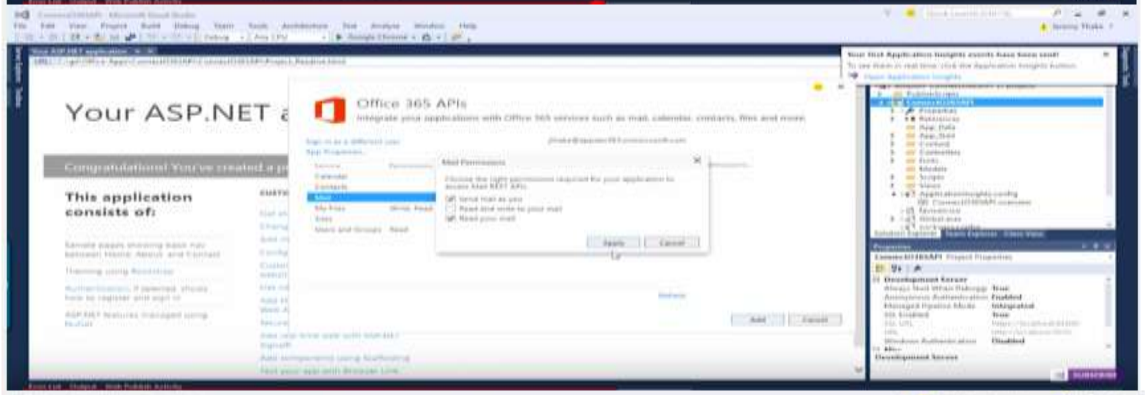
Office 365 APIs

Integrate your applications with Office 365 services such as mail, calendar, contacts, files and more.

My Files Permissions

Choose the right permissions required for your application to access My Files REST APIs

- [x] Read and write your files
- [x] Read your files


Your ASP.NET a

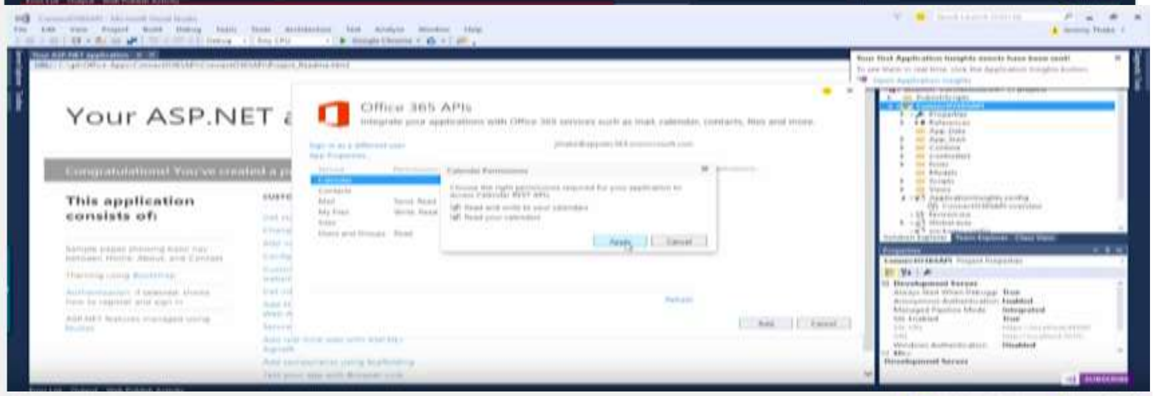
Office 365 APIs

Integrate your applications with Office 365 services such as mail, calendar, contacts, files and more.

Mail Permissions

Choose the right permissions required for your application to access Mail REST APIs

- [x] Send mail as you
- [ ] Read and write to your mail
- [x] Read your mail


Your ASP.NET a

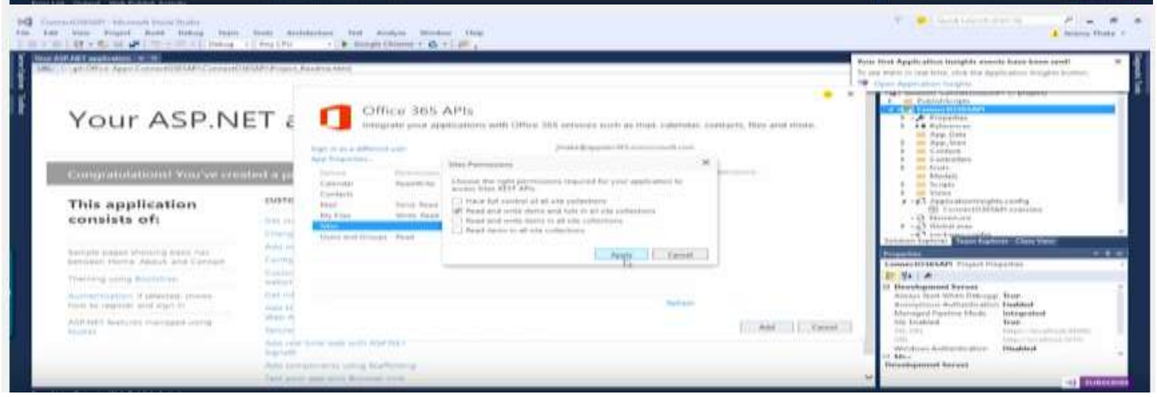
Office 365 APIs

Integrate your applications with Office 365 services such as mail, calendar, contacts, files and more.

Calendar Permissions

Choose the right permissions required for your application to access Calendar REST APIs

- [x] Read and write to your calendars
- [x] Read your calendars


Your ASP.NET a

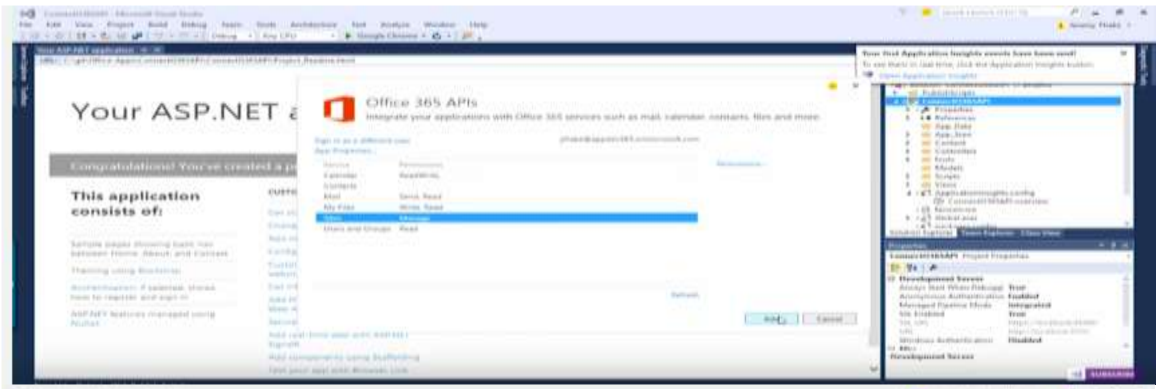
Office 365 APIs

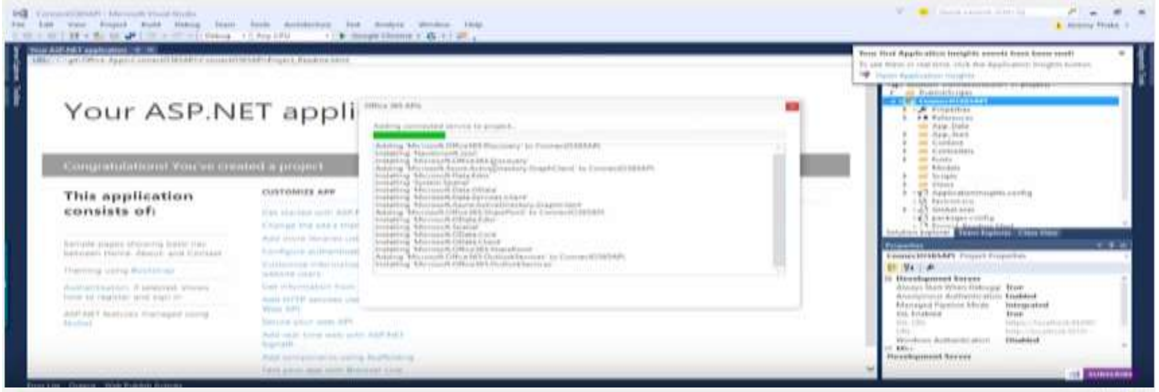
Integrate your applications with Office 365 services such as mail, calendar, contacts, files and more.

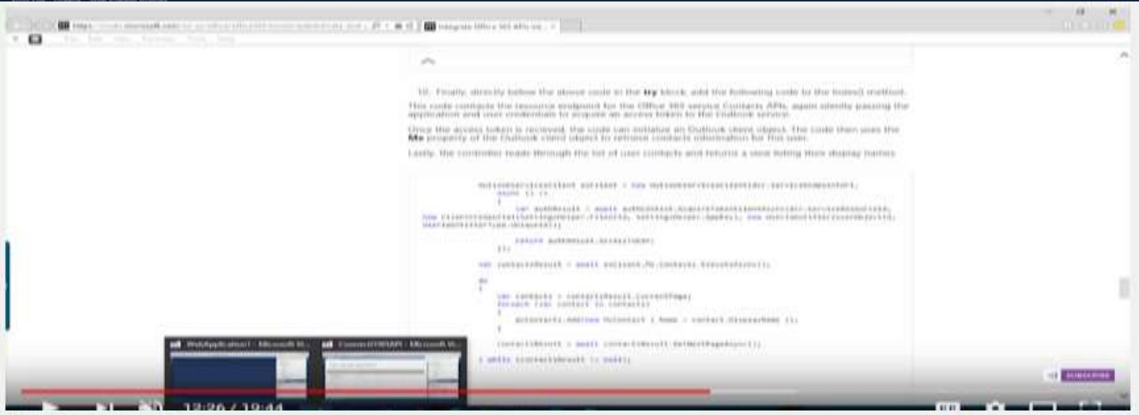
Sites Permissions

Choose the right permissions required for your application to access Sites REST APIs

- [ ] Have full control of all site collections
- [x] Read and write items and lists in all site collections
- [ ] Read and write items in all site collections
- [ ] Read items in all site collections


Your ASP.NET a
Office 365 APIs
Integrate your applications with Office 365 services such as mail, calendar, contacts, files and more.
Congratulations! You've created a p
This application consists of:
Add
Cancel

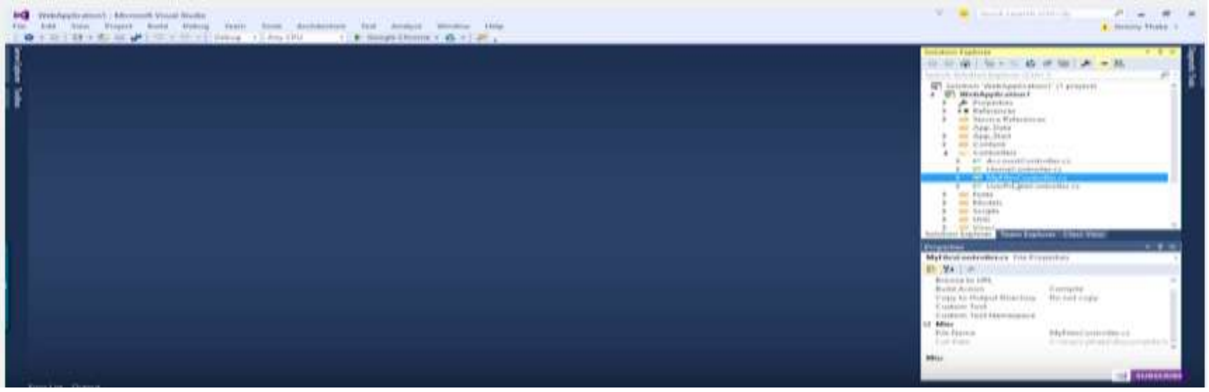

Your ASP.NET appli
Office 365 APIs
Adding connected service to project...
Congratulations! You've created a project
This application consists of:

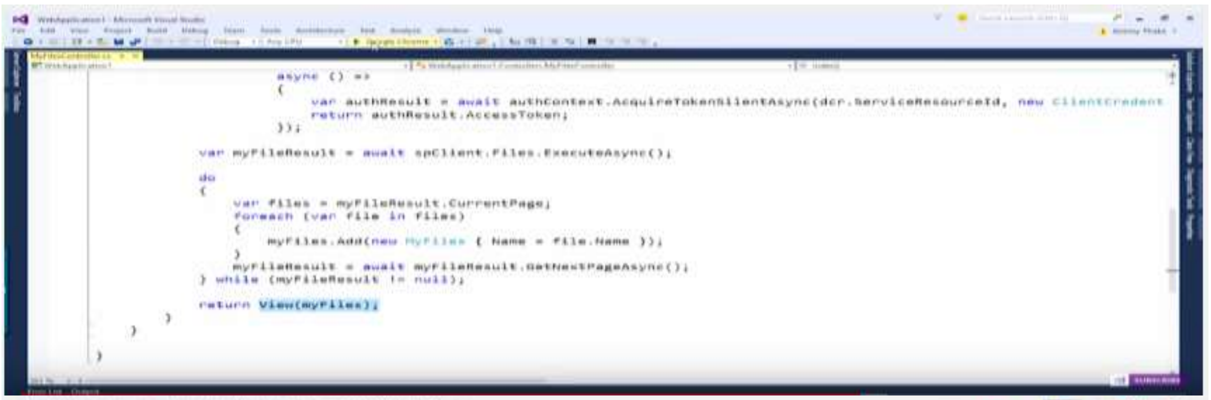

10. Finally, directly below the above code in the **try** block, add the following code to the Index() method. This code contacts the resource endpoint for the Office 365 service Contacts APIs, again silently passing the application and user credentials to acquire an access token to the Outlook service.

Once the access token is received, the code can initialize an Outlook client object. The code then uses the **Me** property of the Outlook client object to retrieve contacts information for this user.

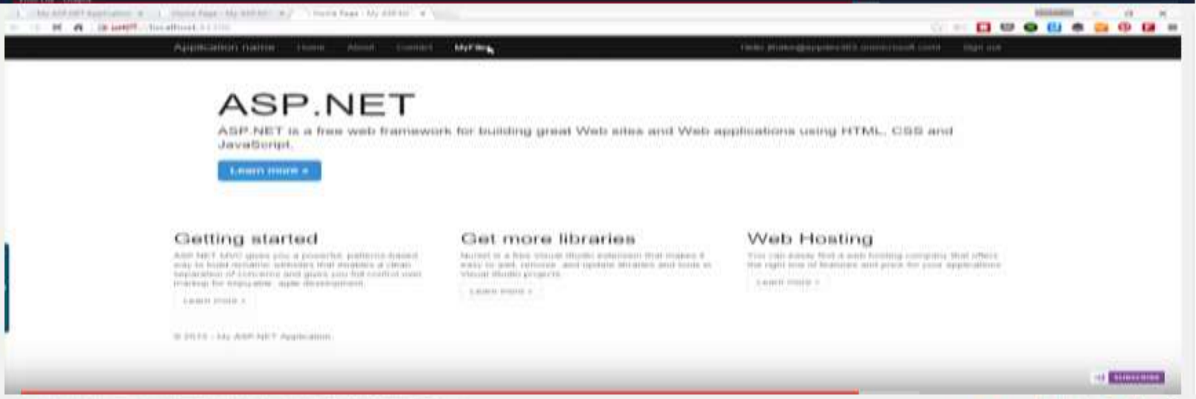
Lastly, the controller loads through the list of user contacts and returns a view listing their display names.

```
            OutlookServicesClient outClient = new OutlookServicesClient(dcr.ServiceEndpointUri,
                async () =>
                {
                    var authResult = await authContext.AcquireTokenSilentAsync(dcr.ServiceResourceId,
new ClientCredential(SettingsHelper.ClientId, SettingsHelper.AppKey), new UserIdentifier(userObjectId,
UserIdentifierType.UniqueId));

                    return authResult.AccessToken;
                });

            var contactsResult = await outClient.Me.Contacts.ExecuteAsync();

            do
            {
                var contacts = contactsResult.CurrentPage;
                foreach (var contact in contacts)
                {
                    myContacts.Add(new MyContact { Name = contact.DisplayName });
                }

                contactsResult = await contactsResult.GetNextPageAsync();

            } while (contactsResult != null);
```





```
                async () =>
                {
                    var authResult = await authContext.AcquireTokenSilentAsync(dcr.ServiceResourceId, new ClientCredent
                    return authResult.AccessToken;
                });

            var myFileResult = await spClient.Files.ExecuteAsync();

            do
            {
                var files = myFileResult.CurrentPage;
                foreach (var file in files)
                {
                    myFiles.Add(new MyFiles { Name = file.Name });
                }
                myFileResult = await myFileResult.GetNextPageAsync();
            } while (myFileResult != null);

            return View(myFiles);
        }
    }
}
```



ASP.NET

ASP.NET is a free web framework for building great Web sites and Web applications using HTML, CSS and JavaScript.

Getting started

ASP.NET MVC gives you a powerful, patterns-based way to build dynamic websites that enables a clean separation of concerns and gives you full control over markup for enjoyable, agile development.

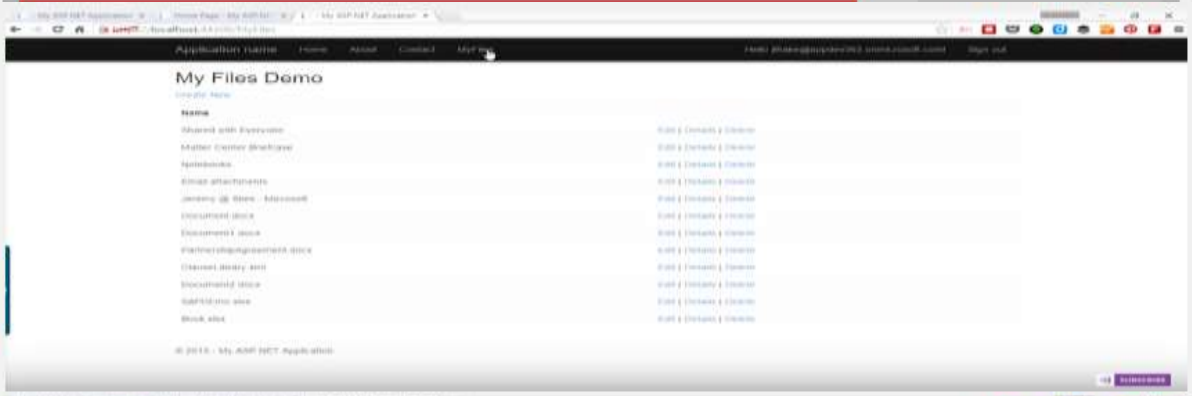
Get more libraries

NuGet is a free Visual Studio extension that makes it easy to add, remove, and update libraries and tools in Visual Studio projects.

Web Hosting

You can easily find a web hosting company that offers the right mix of features and price for your applications.

© 2015 - My ASP.NET Application

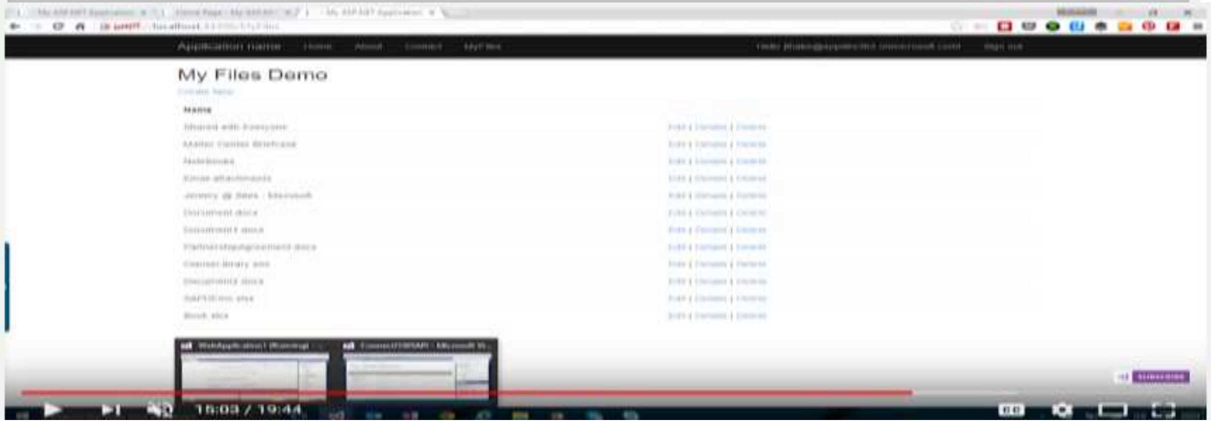


My Files Demo

Create New

Name	
Shared with Everyone	Edit | Details | Delete
Matter Center Briefcase	Edit | Details | Delete
Notebooks	Edit | Details | Delete
Email attachments	Edit | Details | Delete
Jeremy @ Sites - Microsoft	Edit | Details | Delete
Document.docx	Edit | Details | Delete
Document1.docx	Edit | Details | Delete
Partnershipagreement.docx	Edit | Details | Delete
[illegible]	Edit | Details | Delete
Document2.docx	Edit | Details | Delete
[illegible]	Edit | Details | Delete
Book.xlsx	Edit | Details | Delete

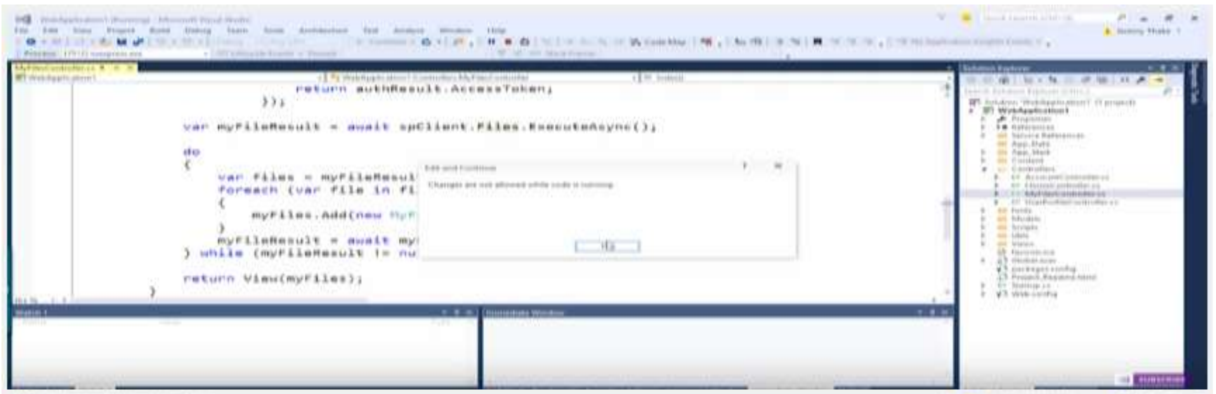
© 2015 - My ASP.NET Application



My Files Demo

Create New

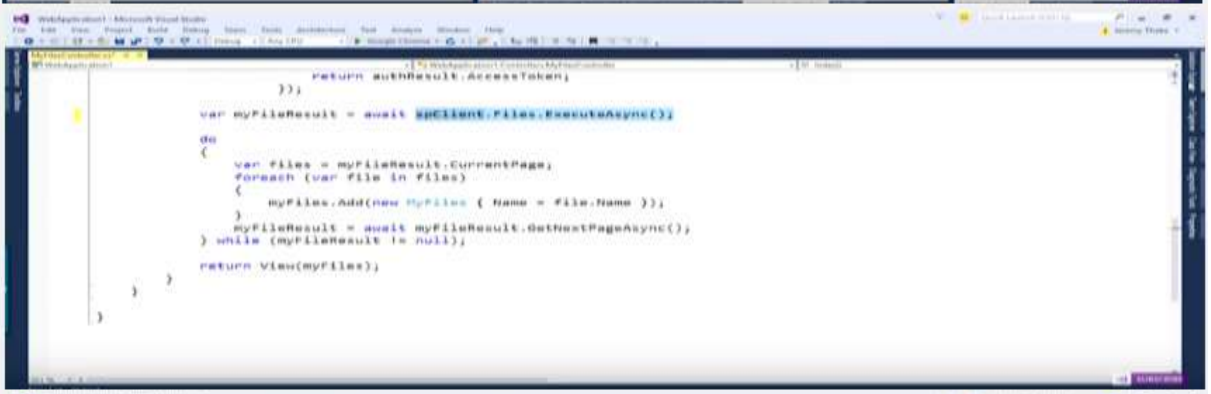
Name	
Shared with Everyone	Edit | Details | Delete
Matter Center Briefcase	Edit | Details | Delete
Notebooks	Edit | Details | Delete
Email attachments	Edit | Details | Delete
Jeremy @ Sites - Microsoft	Edit | Details | Delete
Document.docx	Edit | Details | Delete
Document1.docx	Edit | Details | Delete
Partnershipagreement.docx	Edit | Details | Delete
[illegible]	Edit | Details | Delete
Document2.docx	Edit | Details | Delete
[illegible]	Edit | Details | Delete
Book.xlsx	Edit | Details | Delete



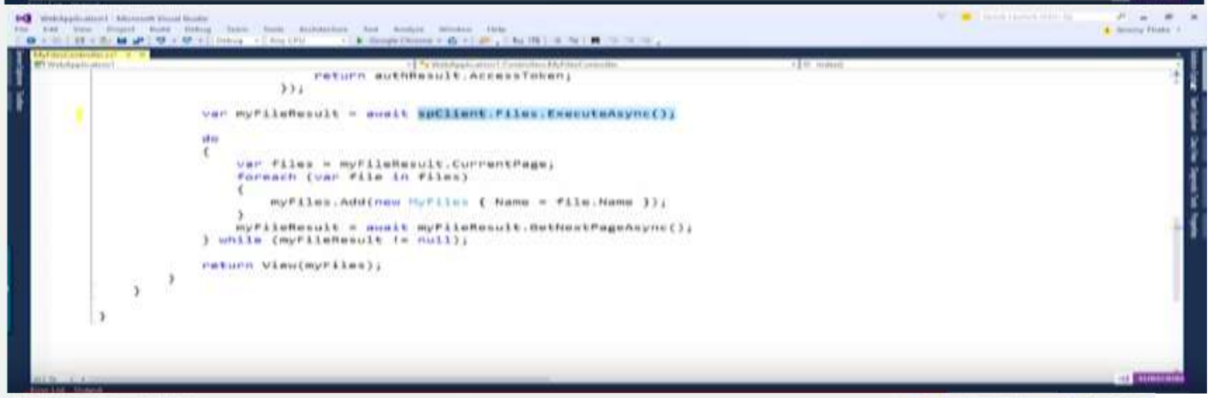
```
                return authResult.AccessToken;
            });

        var myFileResult = await spClient.Files.ExecuteAsync();

        do
        {
            var files = myFileResul
            foreach (var file in fi
            {
                myFiles.Add(new MyF
            }
            myFileResult = await my
        } while (myFileResult != nu

        return View(myFiles);
    }
```

Edit and Continue

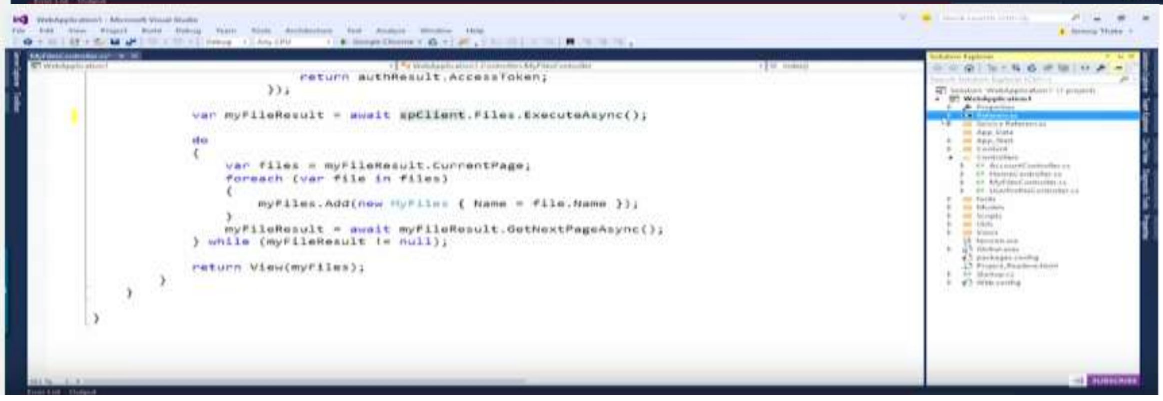
Changes are not allowed while code is running.



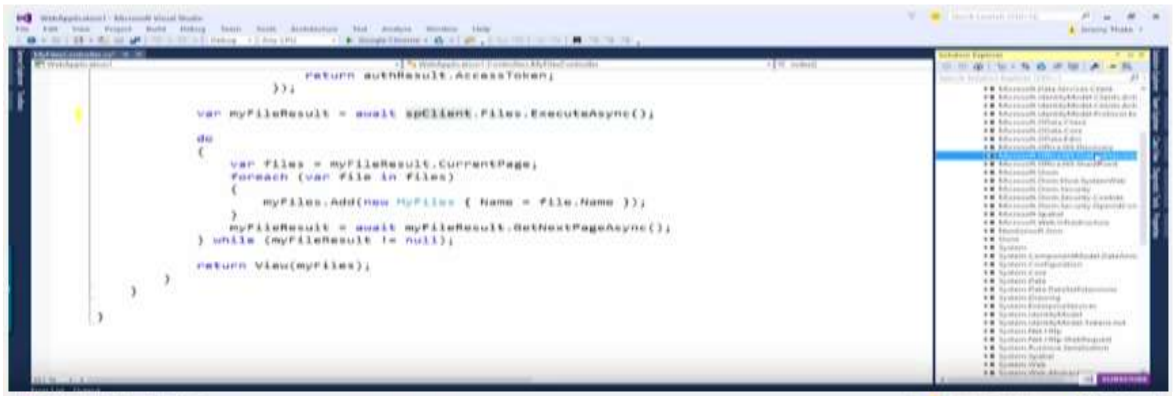
```
                return authResult.AccessToken;
            });

        var myFileResult = await spClient.Files.ExecuteAsync();

        do
        {
            var files = myFileResult.CurrentPage;
            foreach (var file in files)
            {
                myFiles.Add(new MyFiles { Name = file.Name });
            }
            myFileResult = await myFileResult.GetNextPageAsync();
        } while (myFileResult != null);

        return View(myFiles);
    }
  }
}
```



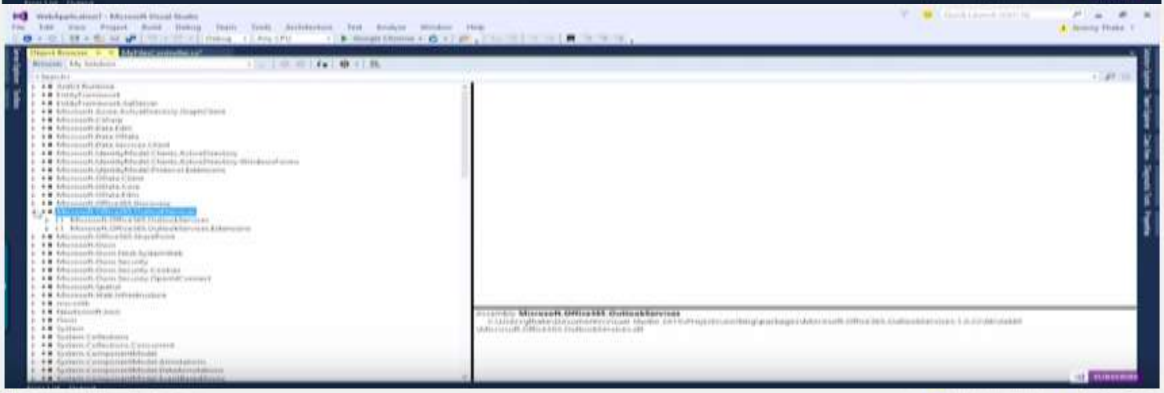
```
                return authResult.AccessToken;
            });

        var myFileResult = await spClient.Files.ExecuteAsync();

        do
        {
            var files = myFileResult.CurrentPage;
            foreach (var file in files)
            {
                myFiles.Add(new MyFiles { Name = file.Name });
            }
            myFileResult = await myFileResult.GetNextPageAsync();
        } while (myFileResult != null);

        return View(myFiles);
    }
  }
}
```



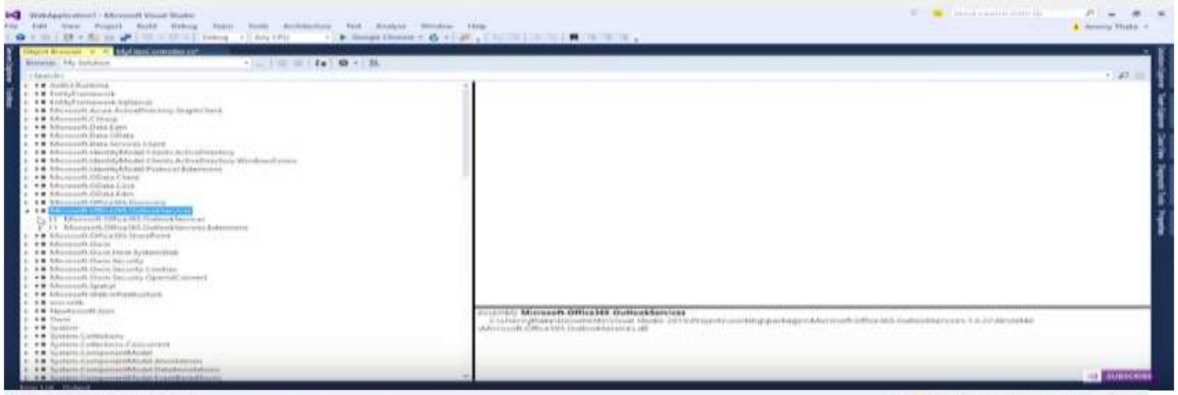
```
                return authResult.AccessToken;
            });

        var myFileResult = await spClient.Files.ExecuteAsync();

        do
        {
            var files = myFileResult.CurrentPage;
            foreach (var file in files)
            {
                myFiles.Add(new MyFiles { Name = file.Name });
            }
            myFileResult = await myFileResult.GetNextPageAsync();
        } while (myFileResult != null);

        return View(myFiles);
    }
  }
}
```



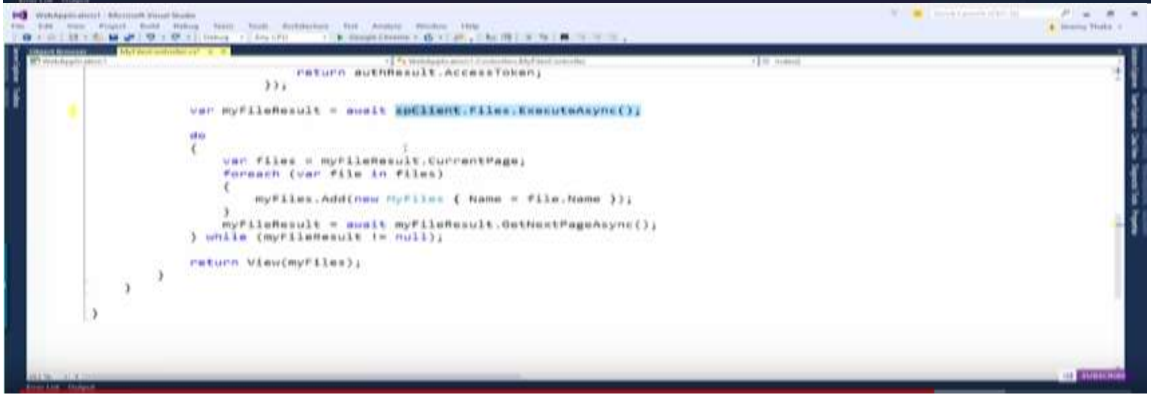
```
                return authResult.AccessToken;
            });

        var myFileResult = await spClient.Files.ExecuteAsync();

        do
        {
            var files = myFileResult.CurrentPage;
            foreach (var file in files)
            {
                myFiles.Add(new MyFiles { Name = file.Name });
            }
            myFileResult = await myFileResult.GetNextPageAsync();
        } while (myFileResult != null);

        return View(myFiles);
    }
  }
}
```



Assembly **Microsoft.Office365.OutlookServices**



Assembly **Microsoft.Office365.OutlookServices**



```
                return authResult.AccessToken;
            });

        var myFileResult = await spClient.Files.ExecuteAsync();

        do
        {
            var files = myFileResult.CurrentPage;
            foreach (var file in files)
            {
                myFiles.Add(new MyFiles { Name = file.Name });
            }
            myFileResult = await myFileResult.GetNextPageAsync();
        } while (myFileResult != null);

        return View(myFiles);
    }
  }
}
```

Experiment No:-10

Aim:- To create a web form application for building resume.

Objective:-

To understand Decide Which Type of Résumé You Want.
To understand List Your Experiences or Skills.
To understand List Your Education.
To understand List Any Awards You've Won and When You Won Them.
To understand List Your Personal Interests.

Theory:-

1. Decide Which Type of Résumé You Want.

There are three types of résumés: chronological, functional and combination. You might want to consider more than one format of résumé if you're applying for multiple jobs.

- **Chronological** is the most traditional format and lists experiences according to the order in which they took place. These résumés generally appeal to older readers and may be best suited for a conservative field.
- **Functional** is a type of résumé that lists your experiences according to skill. This is the format to use if you're changing career direction (and lack direct work experience). Because it displays your skills first, your work experience, or lack thereof, is not the main focus.
- **Combination** combines the best aspects of the **chronological** and **functional** styles. Be careful with length for this format; the résumé can quickly get long.

2. Create a Header.

A header should include your name, phone number and email address. You can also include your mailing address, but leave it out if you plan to post your résumé online.

- Use a phone number that you plan to answer and change your voicemail to a more professional message if necessary.
- Make sure your email address is professional. If your current email address, for example, is candygirl@mail.com or hotbod@inbox.com, it's time to set up a new email, such as janesmith@mail.com or jsmith99@inbox.com.

3. Write a Summary.

In one or two sentences, summarize your work experience and relevant skills. Keep this strong and simple.

- The summary can be useful to explain why you're applying for a role that is a departure from your career path.
- You don't have to include a summary, especially if your experience speaks for itself and is relevant to the jobs you're applying for.

4. List Your Experiences or Skills.

For Chronological/Combination Résumés, List Your Experiences

Starting with your most recent or current job, list your previous work experiences.

- This section shows where you have worked and when. It also states specific accomplishments for each position or job.
- This is where content can make your résumé run over a page, so be selective (if necessary) about what you include.
- Pick experiences that seem most relevant to the position you seek. For inspiration, think of your full-time or part-time work, summer jobs, occasional jobs, internships, fieldwork and special projects.
- Don't worry whether your experiences are "good enough." Employers admire people who have worked hard in a variety of positions.
- Always start each achievement with an accomplishment verb, like *accelerated, achieved, expanded, influenced, solved, maintained, generated, effected, advised, controlled, trained or utilized.*
- Don't worry if there are gaps in the timeline, but keep everything in chronological order, with most recent jobs at the top.

Examples:

Southwestern Writing Center, Peer Writing Tutor, Yuma, AZ

April 2014–Present- Tutored students in writing for all disciplines.- Critiqued peers' writing.

Camp Granite Falls, Area Director, Mountainville, TNJune 2013–September 2017- Directed staff of four while supervising 20 campers.- Taught crafts, sports and cooking.

For Functional/Combination Résumés, List Your Skills

The "skills" section of your résumé is a place where you can show your strengths and individuality. Start by stating each skill. Then back it up with a two- to three-line explanation of how you learned that skill or why you believe you have it. Make these entries short, clear and to the point.

- List skills that are most relevant to the job you seek. Think about what the employer is looking for in relation to what you've done and who you are as a person.
- Don't forget to list computer programs you've had experience with; proficiency can be seen as added value.

Examples:

Self-Motivated: Proactively organized volunteers to assist with distribution at the community food bank.

Bookkeeping: Maintained accurate, detailed inventory reports at school library and subsequently won Top Librarian Assistant award three months straight for Brown County.

5. List Your Activities.

List activities in which you have participated and include what your specific role was in each.

- This is the place to note membership or leadership positions in clubs, organizations of any kind, athletic teams, community organizations and so on.
- If you've had an interesting job unrelated to the field you're pursuing—such as reading to blind children or teaching English as a second language (ESL)—add it here. Employers are always looking for people with diverse backgrounds to work for them.

Examples:

Track Team: Team Captain, Senior Year. Fall 2016–Spring 2017.

Drama Club: "Crazy for You" and "West Side Story." Fall 2017 and 2018.

6. List Your Education.

- List the schools you've attended, starting with the most recent one. Include details such as GPA, class rank or special awards.
- Add any other educational experiences, such as training programs, community college or summer courses, seminars and so on.

Examples:

Oldham County High School, Oldham, PA. 3.8 GPA. Anticipated Graduation: June 2019.

Bellville Adult Education, Bellville, NY. Introduction to Web Design. September 2017.

7. List Any Awards You've Won and When You Won Them.

When you've been recognized by someone else, you should let potential employers know about it. But you shouldn't worry if you haven't received any awards; just skip this section.

Examples:
Richmond County National Essay Contest, Honorable Mention, May 2006.
Honor Roll, South Satchewan High School, Junior and Senior Years, 2008–2010.

8. List Your Personal Interests.

- This section shows you're a well-rounded person who people would want to know and work with.
- Employers often use this section at the start of an interview to break the ice.
- Casual interests are better not to list (e.g., napping, watching reality TV, gossiping). This is really about highlighting hobbies that have helped you grow as a person.
- This résumé step is considered optional. If you're having trouble coming up with interests, or feel your résumé is already too long, feel free to leave it off.

4. Appendix

5. Quiz on the subject

1) What determines the connection pool that a connection should use?

1. A connection string
2. The identity or credentials of the user opening the connection
3. The database being connected to
4. The connection object used to connect to the database

a. 1, 2

b. 1, 2, 3

c. 1, 3

d. 1, 4

ANSWER: 1, 2

Explanation:

Connecting with the database is time consuming and resource intensive task. Connection pool is cache that stores connection string. These connection strings can be reused for future requirements.Only connections with the same configuration can be stored in a pool. Connections are divided into pools by connection string and by identity when integrated security is used. Connection pooling removes the overhead of making a new connection.

2) What are the recommended techniques for enabling connection pooling on for a SQL Server 2000 or SQL Server 2005 database?

1. Setting the OLE DB Services connection string keyword to -4

2. Opening a connection and not explicitly disabling pooling

3. Setting the connection string keyword Pooling = True in the connection string

4. Using the Connection Pooling tab of the ODBC Data Source Administrator dialog Box

a. 1, 2

b. 1, 2, 3

c. 2, 3

d. 1, 4

ANSWER: 2, 3

Explanation:

For enabling connection pooling on for a SQL Server 2000 or SQL Server 2005 database set pooling = true and should not explicitly disable it.

3) How do I explicitly turn on connection pooling for an OLE DB data source?

a. By setting the OLE DB Services connection string keyword to 0

b. By setting the OLE DB Services connection string keyword to -4

c. By setting the OLE DB Services connection string keyword to -1

d. By setting the OLE DB Services connection string keyword to -7

ANSWER: By setting the OLE DB Services connection string keyword to -1

Explanation:

No explanation is available for this question!

4) What property contains the actual error message returned by SQL Server?

1. SqlException.Source

2. SqlException.Message

3. SqlError.Class

4. SqlError.Message

a. 1, 2

b. 1, 2, 3

c. 1, 3

d. 2, 4

ANSWER: 2, 4

Explanation:

SqlException.Message and SqlError.Messageproperty contains the actual error message returned by SQL Server.

5) What is the connection string's key / value pair for using WindowsAuthentication in SQLServer 2000 and SQL Server 2005?

a. 1, 2

b. 1, 2, 3

c. 2, 3

d. 1, 4

ANSWER: 2, 3

Explanation:

You can provide yes, no, true, SSPI and false value to Integrated Security property of connection string according to the requirement. If Integrated Security= truethen current Windows account credentials are used for authentication. SSPI is equivalent to specifying True.

6. How many Bytes are stored by 'Long' Data type in C# .net?
a) 8
b) 4
c) 2

d) 1
View Answer
Answer: a
Explanation: ‘Long’ is the data type keyword used for storing data of unlimited length so by definition its size is always maximum i.e 8.

7. Choose “.NET class” name from which data type “UInt” is derived ?
a) System.Int16
b) System.UInt32
c) System.UInt64
d) System.UInt16
View Answer
Answer: b
Explanation: By Definition class assigned to
a) System.Int16 = short.
b) System.UInt32 = UInt.
c) System.UInt64 = ULong.
d) System.UInt16 = UShort.

8. Correct Declaration of Values to variables ‘a’ and ‘b’?
a) int a = 32, b = 40.6;
b) int a = 42; b = 40;
c) int a = 32; int b = 40;
d) int a = b = 42;
View Answer
Answer: c
Explanation: a) Although,declaration of ‘b’ and ‘a’ are correct but initialization of value to ‘b’ should be ‘int’ data type not float.
b) Missing declaration type of ‘b’.
c) correctly declared data types ‘a’ and ‘b’.
d) ‘b’ can’t be assigned values before declaration.

9. Select error in the given program :

```
    Static Void Main(String[] args)
    {

        const int m = 100;

        int n = 10;

        const int k = n / 5 * 100 * n ;

        Console.WriteLine(m * k);

        Console.ReadLine();

    }
```

a) ‘k’ should not be declared constant
b) Expression assigned to ‘k’ should be constant in nature
c) Expression (m * k) is invalid
d) ‘m ‘ is declared in invalid format
View Answer
Answer: b
Explanation:’k’ should be declared as const int k = 10/5 * 100*10 i.e only constant values should be assigned to a constant.
Output:

```
Error 1 - The expression being assigned to 'k' must be constant.
```

10. Select the output for the following set of code :

```
    static void Main(string[] args)

    {

        int i = 30;
        int j = 25 % 25;

        if (Convert.ToBoolean(Convert.ToInt32(i = j)))

        {

            Console.WriteLine("In if");

        }

        else

        {

            Console.WriteLine("In else");

        }

        Console.WriteLine("In main");

        Console.ReadLine();

    }
```

a) In if
b) In else
c) In if
In main
d) In else
In main
View Answer
Answer: d
Explanation: Usage of '=' operator instead of '==' operator .hence,the condition is not true.
Output:

```
        In else
        In main
```

11. Select the correct output for the given set of code?

```
class sample
{
    public  int i;
    void display()
    {
        Console.WriteLine(i);
    }
}
class sample1 : sample
{
    public  int j;
    public void display()
    {
        Console.WriteLine(j);
    }
}
class Program
{
    static void Main(string[] args)
```

```
        {
            sample1 obj = new sample1();
            obj.i = 1;
            obj.j = 2;
            obj.display();
            Console.ReadLine();
        }
    }
```

a) 1
b) 3
c) 2
d) Compile Time error
View Answer
Answer: c
Explanation: class sample & class sample1 both contain display() method, class sample1 inherits class sample, when display() method is called by object of class sample 1, display() method of class sample 1 is executed rather than that of Class sample.

6. Conduction of Viva-Voce Examinations

Q.1 What is .NET Framework.

Q.2 what different types of applications.

Q.3 What is C# .

Q.4 Explain control statements in C# .Net.

Q.5 Explain class object and properties in C# .Net.

Q.6 What is Inheritance and abstract class in C# .Net.

Q.7 What is Interface and Exception handling in C# .Net.

Q.8 What Is Control statement are used in C#.

Q.9 What is single inheritance.

Q.10 What is Multiple inheritance.

Q.11 What is Multilevel inheritance.

Q.12 What is Hybrid inheritance.

Q.13 What is polymorphism.

Q.14 What is Toolbar, properties

Q.15 Which are tools are used in login form.

Q.16 What is windows Application.

Q.17 What is meaning Of ***** operator.

Q.18 what is the syntax for Login form.

Q.19 What is procedure for next Form generated.

Q.20 What calculator application.

Q.21 What are different tools are used in calculator application.

Q.22 What traffic signal application.

Q.23 What are different tools are used in traffic signal application.

Q.24 What is Timer.

Q.25 What is validation control in ASP .Net.

Q.26 What is mean by Master page in Asp .Net.

Q.27 How are the database connectivity in ASP.NET

Q.28 What is mean by data set.

Q.29 What is data grid view.

Q.30 What is RequiredFieldValidator.

Q.31what is RangeValidator.

Q.32 What is CompareValidator.

Q.33 What is RegularExpressionValidator.

Q.34 What is CustomValidator.

Q.35 What is Validation Summary.

Q36 What is source code.

Q.37 What is .aspx code.

7. Evaluation and Marking System

Basic honesty in the evaluation and marking system is absolutely essential and in the process impartial nature of the evaluator is required in the examination system to become popular amongst the students. It is a wrong approach or concept to award the students by way of easy marking to get cheap popularity among the students to which they do not deserve. It is a primary responsibility of the teacher that right students who are really putting up lot of hard work with right kind of intelligence are correctly awarded.

The marking patterns should be justifiable to the students without any ambiguity and teacher should see that `students are faced with unjust circumstances.

The assessment is done according to the directives of the Principal/ Vice Principal/ Dean Academics

DOs and DON' Ts in Laboratory:

1. Do not handle any equipment before reading the instructions/Instruction manuals.

2. Read carefully the program of the equipment before it is switched on whether ratings 230 V/50Hz or 115V/60 Hz. For Indian equipments, the power ratings are normally 230V/50Hz. If you have equipment with 115/60 Hz ratings, do not insert power plug, as our normal supply is 230V/50 Hz, which will damage the equipment.

3. Observe type of sockets of equipment power to avoid mechanical damage

4. Do not forcefully place connectors to avoid the damage

5. Strictly observe the instructions given by the teacher/Lab Instructor

Instruction for Laboratory Teachers::

1. Submission related to whatever lab work has been completed should be done during the next lab session.

2. The promptness of submission should be encouraged by way of marking and evaluation patterns that will benefit the sincere students