3.1 Web.config

ASP.NET application is affected by different settings in the configuration files:

- machine.config
- web.config

The machine.config file contains default and the machine-specific value for all supported settings. The machine settings are controlled by the system administrator and applications are generally not given access to this file.

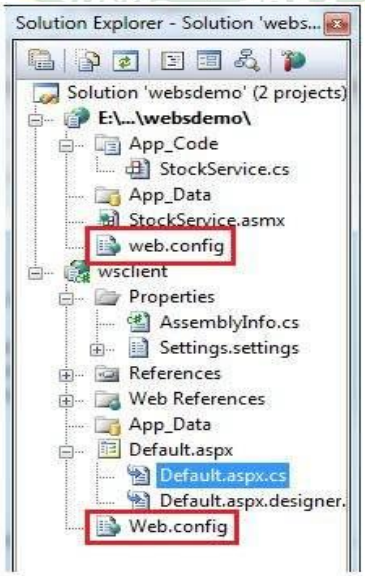
An application however, can override the default values by creating web.config files in its roots folder. The web.config file is a subset of the machine.config file.

If the application contains child directories, it can define a web.config file for each folder. Scope of each configuration file is determined in a hierarchical top-down manner.

Any web.config file can locally extend, restrict, or override any settings defined on the upper level.

Visual Studio generates a default web.config file for each project. An application can execute without a web.config file, however, you cannot debug an application without a web.config file.

The following figure shows the Solution Explorer for the sample example used in the web services tutorial:



In this application, there are two web.config files for two projects i.e., the web service and the web site calling the web service.

The web.config file has the configuration element as the root node. Information inside this element is grouped into two main areas: the configuration section-handler declaration area, and the configuration section settings area.

Web.config is the primary configuration file for ASP.NET web applications. It is an XML-based file that contains settings and configurations for various aspects of the application, including:

- **Connection Strings:** Database connection details.
- **Authentication and Authorization:** Security settings for user access.
- **Error Handling:** Custom error pages and logging.
- **Session State:** How session data is managed.
- **Application Settings:** Custom key-value pairs for application-specific configurations.
- **Compilation Settings:** Debugging, tracing, and language settings.

Example:

```
<?xml version="1.0"?>
<configuration>
  <connectionStrings>
    <add name="MyDatabase" connectionString="Data Source=server;Initial
    Catalog=database;Integrated Security=True" providerName="System.Data.SqlClient"/>
  </connectionStrings>
  <system.web>
    <authentication mode="Forms">
      <forms loginUrl="Login.aspx" timeout="2880"/>
    </authentication>
    <customErrors mode="On" defaultRedirect="Error.aspx"/>
  </system.web>
</configuration>
```

3.2 Sitemappath Server Control

The **SiteMapPath** control basically is used to access web pages of the website from one webpage to another. It is a navigation control and displays the map of the site related to its web pages. This map includes the pages in the particular website and displays the name of those pages. You can click on that particular page in the Site Map to navigate to that page. We can say that the SiteMapPath control displays links for connecting to URLs of other pages. The SiteMapPath control

uses a property called **SiteMapProvider** for accessing data from databases and it stores the information in a data source. The SiteMapProvider internally utilizes the SiteMapProvider abstract class defined under the **System.Web** namespace. The representation of the SiteMapPath control is as follows:

Root Node->Child Node

Public Properties of SiteMapPath class

ParentLevelsDisplayed: It specifies the number of levels of parent nodes and then displays the control accordingly related to the currently displayed node.

RenderCurrentNodeAsLink: It specifies whether or not the site navigation node that represents the currently displayed page is rendered as a hyperlink.

PathSeperator: It specifies the string that displays the SiteMapPath nodes in the rendered navigation path.

Style properties of the SiteMapPath class

CurrentNodeStyle: It specifies the style used for the display text for the current node.

RootNodeStyle: It specifies the style for the root node style text.

NodeStyle: It specifies the style used for the display text for all nodes in the site navigation path.

Creating the SiteMapPath Control

Now let's create an application by using the SiteMapPath control. In this application we design the following three pages:

- The Home Page (**Default.aspx**)
- The First Page (**myweb1.aspx**)
- The Second page (**myweb2.aspx**)

Now we can use the SiteMapPath control using the following steps:

Step 1: Open Microsoft Visual Studio 2010.

Step 2: Select File->New->Web Site.

Step 3: Select ASP.NET Web Site and name it as mywebsite.

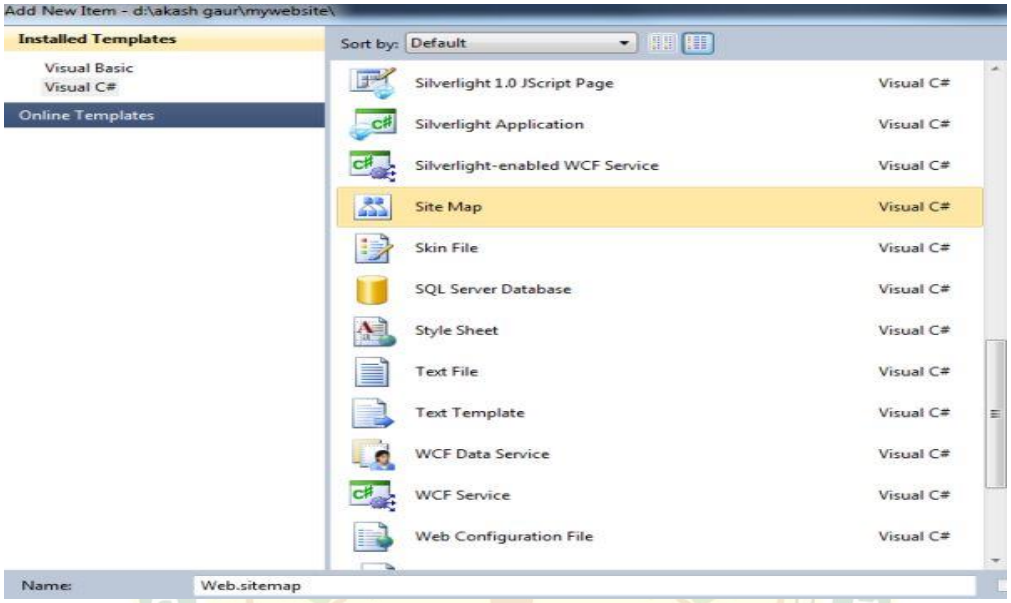
Step 4: Add two web forms to the application named myweb1.aspx and myweb2.aspx by performing the following steps:

- Move to the Solution Explorer window
- Right-click on the application name and select the Add New Item option from the context menu
- Name the web form as myweb1.aspx and click the Add button

Step 5: Similarly add the myweb2.aspx web form to the application. After that we have to add the Site Map file into the project. The Site Map file is the XML file and has the extension .sitemap. The steps are as follows.

Step 6: Right-click the application in the Solution Explorer window and then click the Add New Item option from the context menu.

Step 7: Select the Site Map Template from the Templates Pane. Note that, by default, the file has the name web.sitemap.

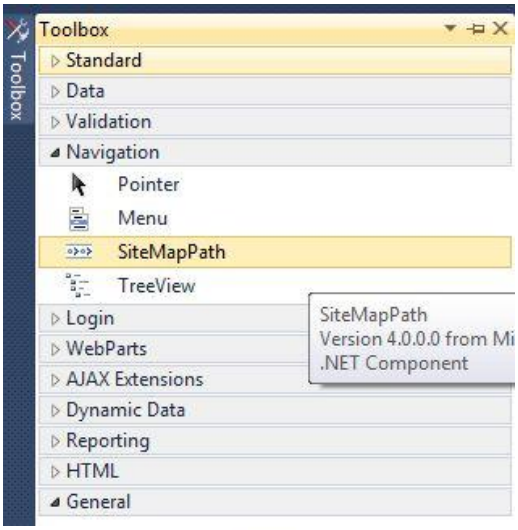


Step 8: Now click the Add button to add the sitemap file to our application.

Step 9: Now we can check that the sitemap file has been included in our project and we can see it in the Solution Explorer. And now we have to set the URL and title attributes in the sitemap file.

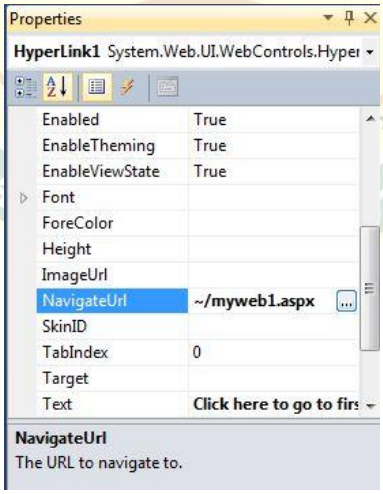
```
<?xml version="1.0" encoding="utf-8" ?>
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
    <siteMapNode url="Default.aspx" title="myhomepage"  description="">
        <siteMapNode url="myweb1.aspx" title="myfirstpage"  description="" />
        <siteMapNode url="myweb2.aspx" title="mysecondpage"  description="" />
    </siteMapNode>
</siteMap>
```

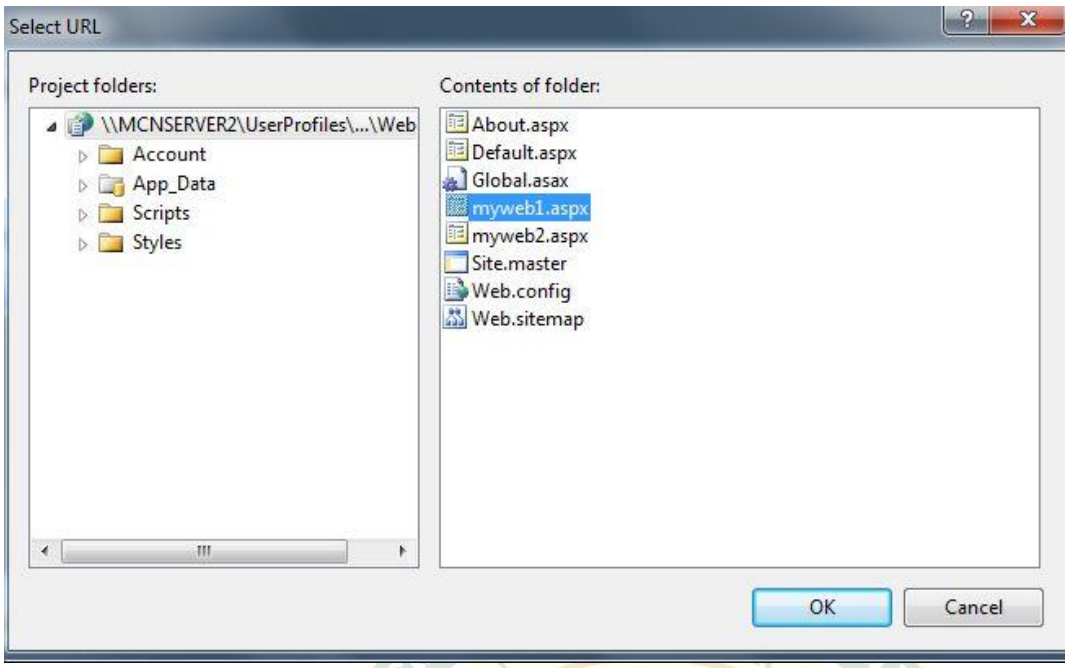
Step 10: Now Add one SiteMapPath control on the Default.aspx page from the navigation tab of the toolbox.



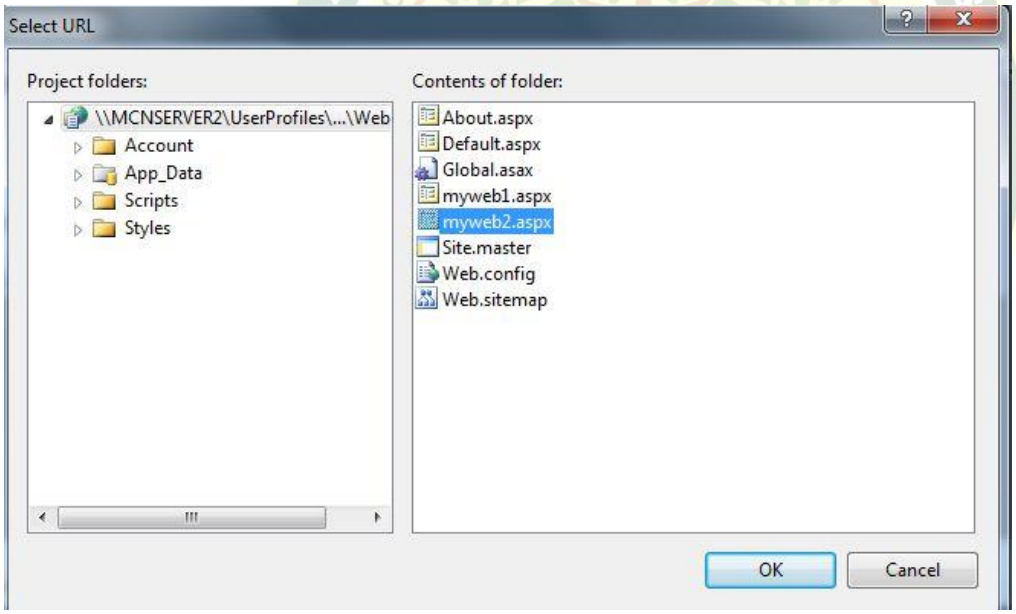
Step 11: Add three labels and two hyperlink controls from the toolbox to the Default.aspx page and set the text property of the control.

Step 12: Now set the NavigateUrl property of the first hyperlink control and select the myweb1.aspx file as follows.





Step 13: Now set the NavigateUrl Property of second hyperlink control and select myweb2.aspx file.



The source code of the Default.aspx page is as follows:

```
<%@ Page Title="Home Page" Language="C#" MasterPageFile="~/Site.master" AutoEventWire
up="true"
    CodeFile="Default.aspx.cs" Inherits="_Default" %>
<asp:Content ID="HeaderContent" runat="server" ContentPlaceHolderID="HeadContent">
</asp:Content>
<asp:Content ID="BodyContent" runat="server" ContentPlaceHolderID="MainContent">
```

```
<asp:Label ID="Label1" runat="server"
    Text="I have designed my website by using SiteMapPath control in ASP.NET"
    Font-Bold="True" Font-Names="Arial Black" Font-Size="Large"></asp:Label>
 <asp:Label ID="Label2" runat="server" Text="sitemap:"></asp:Label>
 <asp:SiteMapPath ID="SiteMapPath1" runat="server">
</asp:SiteMapPath>
<asp:Label ID="Label3" runat="server"
    Text="Click any link below  to go to desired page....."></asp:Label>
<asp:HyperLink ID="HyperLink1" runat="server" NavigateUrl="~/myweb1.aspx">Click here
to go to first page </asp:HyperLink>
<asp:HyperLink ID="HyperLink2" runat="server" NavigateUrl="~/myweb2.aspx">click here to
go to second page</asp:HyperLink>
</asp:Content>
```

Step 14: Now in the design mode of the myweb1.aspx page, add the same control but add only one hyperlink control and set its NavigateUrl property to myweb2.aspx then click ok.

The source code of the myweb1.aspx page is as follows:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="myweb1.aspx.cs" Inherits="my
web1" %>
<!DOCTYPE html PUBLIC "-
//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
   <title></title>
</head>
<body>
   <form id="form1" runat="server">
   <div>
     <asp:Label ID="Label1" runat="server" Font-Bold="True" Font-Names="Arial Black"
Font-Size="Large"
         Text="Thank you for clicking.This is My First Webpage....."></asp:Label>
      <asp:Label ID="Label2" runat="server" Text="SiteMap:"></asp:Label>
```

```
        <asp:SiteMapPath ID="SiteMapPath1" runat="server">
        </asp:SiteMapPath>
<asp:HyperLink ID="HyperLink1" runat="server" NavigateUrl="~/myweb2.aspx">click here to
go to mywebpage2</asp:HyperLink>
    </div>
    </form>
</body>
</html>
```

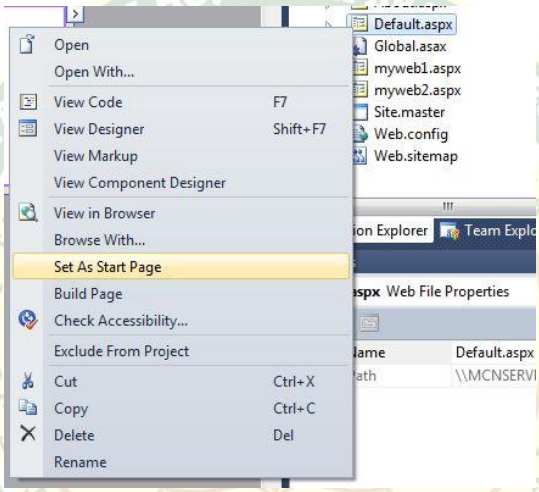
Step 15: Now in the design mode of the myweb2.aspx page, add the same controls as in myweb1.aspx and set the NavigateUrl property of the hyperlink as Default.aspx then click ok.

The source code of myweb2.aspx is as follows:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="myweb2.aspx.cs" Inherits="my
web2" %>
<!DOCTYPE html PUBLIC "-
//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:Label ID="Label1" runat="server" Font-Bold="True" Font-Names="Arial Black"
            Font-Size="Large"
            Text="Thanx for Clicking.This is my Second webpage by using SiteMapPath control..
.....">
        </asp:Label>
    </div>
        <asp:Label ID="Label2" runat="server" Text="SiteMap: "></asp:Label>
        <asp:SiteMapPath ID="SiteMapPath1" runat="server">
        </asp:SiteMapPath>
```

```
    </p>
    <asp:HyperLink ID="HyperLink1" runat="server" NavigateUrl="~/Default.aspx">Click here
to go to Home page</asp:HyperLink> 
    </p>
    </form>
</body>
</html>
```

Step 16: Now in the Solution Explorer window, right click on the Default.aspx page and select the Set As Start Page option.

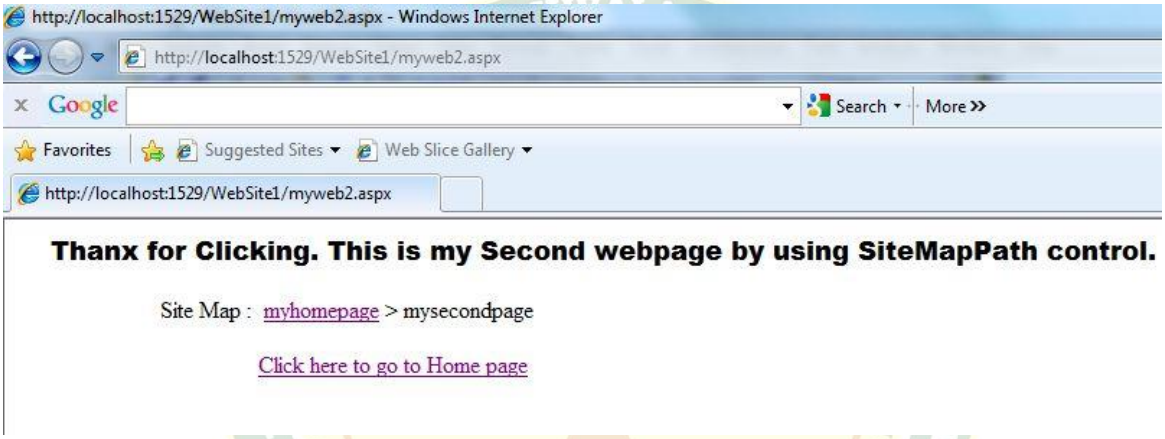


Step 17: Press the F5 key to run the application.

Output

After clicking on the first link the output is as follows.

After clicking the link, the output of the second web page is as follows:



3.3 Web Services

3.3.1 Basics of Web Services

- They are small unit of code.
- They are design to handle limited set of task.
- It used xml based communicating protocols.
- They are independent of operating system and programming languages.
- It connect people, system, devices.
- XML – standard for storing carrying and exchanging data
- Web service is a programmable application component accessible via standard web protocols. i.e. it is small code for limited task.
- .NET framework contains classes, attributes, protocol for realization of web services
- It is middleware for distributed application. It is used for remote procedure calls and data exchange. It is open standard based on XML.

Web services are software systems designed to support interoperable machine-to-machine interaction over a network. They use standard protocols like SOAP (Simple Object Access Protocol) or REST (Representational State Transfer) for communication, often employing XML or JSON for data exchange. Key characteristics include:

- **Interoperability:** Can be consumed by various platforms and programming languages.
- **Standard Protocols:** Rely on established protocols like HTTP, XML, SOAP, WSDL, UDDI.
- **Loose Coupling:** Services are independent and can evolve without directly impacting consumers.

3.3.2 Interacting with web services

- Web need to create two applications.
  - Web Service application.
  - Web application that consume web service
- Interacting with web services involves consuming the service by sending requests and processing the responses. This typically involves:
- **Creating a Service Reference:** In .NET, this involves adding a "Service Reference" to your project, which generates proxy classes based on the WSDL (Web Services Description Language) of the service.
- **Instantiating the Service Proxy:** Create an instance of the generated proxy class.
- **Calling Service Methods:** Invoke the methods exposed by the web service through the proxy object.
- **Handling Responses:** Process the data returned by the web service.

Example (Consuming a SOAP-based web service in C#):

```
// Assuming a service reference named 'MyServiceReference'
// and a web service with a method 'GetCustomerDetails'
try
{
    MyServiceReference.MyServiceClient client = new
MyServiceReference.MyServiceClient();
    string customerId = "12345";
    string customerDetails = client.GetCustomerDetails(customerId);
    Console.WriteLine("Customer Details: " + customerDetails);
    client.Close();
}
catch (Exception ex)
{
    Console.WriteLine("Error: " + ex.Message);
}
```