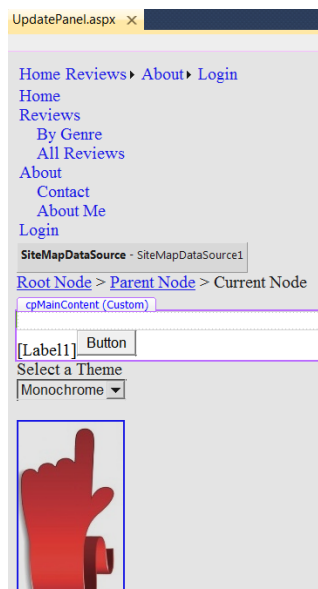


Adding an UpdatePanel to a Page

In this exercise, you add a `Label` and a `Button` control to a page. When you click the button in the browser, the `Text` property of the `Label` is updated with the current date and time at the server.

To avoid the page flicker typically associated with postbacks, you then wrap the controls in an `UpdatePanel` to see how that control affects the behavior.

1. Open the project in Visual Studio.
2. In the folder, create a new Web Form called `UpdatePanel.aspx` using your custom template. Give the page a Title of `UpdatePanel Demo`.
3. Switch the new page into Design View and drag a `Label` control and a `Button` control from the Toolbox into the `cpMainContent` placeholder. If the `ContentPlaceHolder` suddenly gets as small as the `Label`, simply drop the `Button` on top of the `Label`. The `Button` is then placed before the `Label` but if you now drag the `Label` on top of the `Button` again, the two change places.

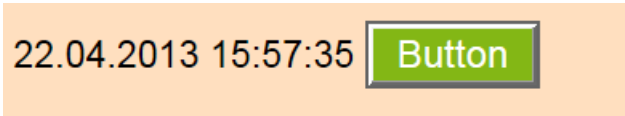


4. Use the Properties Grid to clear the `Text` property of the `Label` control. To do this, right-click the `Text` property label in the Properties Grid and choose `Reset`.

5. Double-click the grey and read-only area of the page in Design View to set up a handler for its `Load` event and add the following code to the handler that VS added for you:

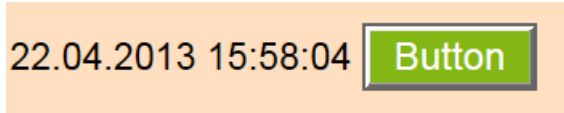
```
protected void Page_Load(object sender, EventArgs e)
{
    Label1.Text = System.DateTime.Now.ToString();
}
```

6. Save all your changes and press `Ctrl+F5` to open the page in your browser. The `Label` displays the current date and time.



22.04.2013 15:57:35 Button

Click the `Button` control a few times. **Note** that each time you click the button, the page flickers and is then redrawn, displaying the updated date and time.



22.04.2013 15:58:04 Button

Now take a look at the HTML that is used by the browser (right-click the page in the browser and choose View Source or View Page Source). Notice how the page contains a `span` element with the date and time that was sent from the server.

```
<span id="cpMainContent_Label1">22.04.2013 15:58:04</span>
```

7. Close your browser, go back into VS and switch the page `UpdatePanel.aspx` to Source View. Make some room right before the `Label` control, and then type `updatepanel` and press `Tab`. VS inserts the code for an `UpdatePanel` and a `<ContentTemplate>` for you.

```
<asp:UpdatePanel runat="server">
    <ContentTemplate>
    </ContentTemplate>
</asp:UpdatePanel>
```

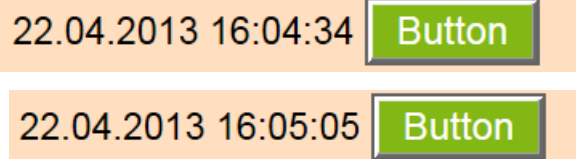
8. Next, cut both the closing `</ContentTemplate>` and the closing `</UpdatePanel>` tags and paste them below the button you created in step 3. You should end up with this markup (although your control may lack the ID attribute):

```
<asp:Content ID="Content2" ContentPlaceHolderID="cpMainContent" Runat="Server">
  <asp:UpdatePanel runat="server">
    <ContentTemplate>
      <asp:Label ID="Label1" runat="server"></asp:Label>
      <asp:Button ID="Button1" runat="server" Text="Button" />
    </ContentTemplate>
  </asp:UpdatePanel>
</asp:Content>
```

9. Right before the opening tag of the `UpdatePanel`, drag a `ScriptManager` from the AJAX Extensions category of the Toolbox. Alternatively, type `sm` followed by the Tab key to insert the `ScriptManager` using a code snippet. Your code should look similar to this (although your `ScriptManager` may lack the ID attribute and may use a self-closing element when you use a code snippet):

```
<asp:Content ID="Content2" ContentPlaceHolderID="cpMainContent" Runat="Server">
  <asp:ScriptManager runat="server" />
  <asp:UpdatePanel runat="server">
    <ContentTemplate>
```

10. Save your changes and request the page in the browser again. Click the button a few times to update the label with the current date and time. **Note** that there is no page flicker now. It's as if only the label is updated on the page. If you look at the source in the browser again you see the `span` element that contains the date and time of the very first request. The updates to the label that were added by clicking the button are not a part of the HTML source because they have been added dynamically by the Ajax framework to the browser's internal HTML.



```
<span id="cpMainContent_Label1">22.04.2013 16:04:34</span>
```

Flicker-free Pages – Putting It All Together

In this exercise, you modify the user control `ContactForm.ascx` that you created earlier, wrapping the entire control in an `UpdatePanel` so the page doesn't perform a full postback when you enter a message and click the `Send` button. To help users understand that the page is busy when the message is being sent, you add an `UpdateProgress` panel to the control. Inside this control you place an animated GIF image. Alternatively, you can go to www.ajaxload.info and create your own animated image.

1. Open the user control `ContactForm.ascx` from the `Controls` folder in `Source View` and wrap the entire `<table>` element and the `Label` at the bottom of the control in an `UpdatePanel` with a `<ContentTemplate>`. You can do this by typing the code directly in `Source View`, by using a code snippet, or by dragging the control from the `Toolbox`. Make sure the `ID` of the `UpdatePanel` is set to `UpdatePanel1`. You should end up with the following code:

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

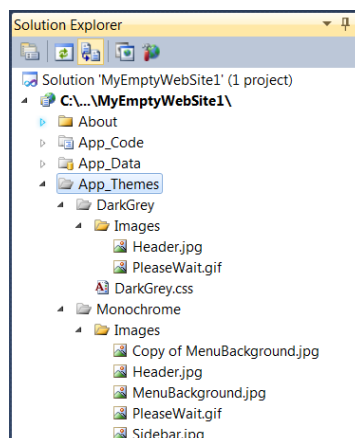
    <table class="style1" runat="server" id="FormTable">
      .....
    </table>
    <asp:Label ID="Message" runat="server" Text="Message Sent!" Visible="False"></asp:Label>

  </ContentTemplate>
</asp:UpdatePanel>
```

2. Save the changes to the control and then open the file `Frontend.master` from the `MasterPages` folder. Between the opening `<form>` tag and the `<div>` for the `PageWrapper`, add a `ScriptManager` control by dragging it from the `Toolbox` into the source of the page. You should end up with this code:

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

3. Save the changes to the master page and close it.
4. Open the `UpdatePanel.aspx` page you created in an earlier example and remove the `ScriptManager` element. Because this control is now declared in the master page, you can no longer redefine it in pages that are based on that master. Save and close the page.
5. Open the `Contact.aspx` page from the `About` folder in your browser and then fill in the contact form. Note that as soon as you click the `Send` button, the form disappears and is replaced with the label stating that the message is sent. Just as with the earlier example, you'll notice no page flicker when the page reloads and displays the text **Message Sent**.
6. To keep the user updated on the progress while the message is delivered to the mail server, you should add an `UpdateProgress` control to the page. Inside this control, you add an animated image and some text informing the user the message is being sent. To add the image, drag the file `PleaseWait.gif` from Windows Explorer into the `Images` folder of the `Monochrome` theme under `App_Themes`. Repeat this process, but now drag `PleaseWait.gif` from the `DarkGrey` folder into its respective theme's `Images` folder.



7. Open the `Monochrome.css` file, scroll all the way down to the end, and add the following rule:

```
.PleaseWait
{
    height: 32px;
    width: 500px;
    background-image: url(Images/PleaseWait.gif);
    background-repeat: no-repeat;
    padding-left: 40px;
    line-height: 32px;
}
```

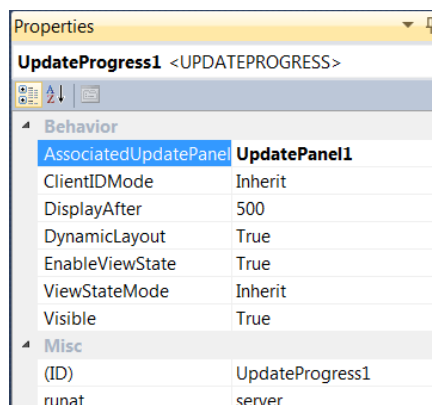
8. Copy the exact same rule into the `DarkGrey.css` file for the `DarkGrey` theme.
9. Switch back to the `ContactForm.ascx` user control and below the closing tag of the `UpdatePanel` at the end of the file, drag an `UpdateProgress` control from the `AJAX Extensions` category of the `Toolbox`. Set its `AssociatedUpdatePanelID` to `UpdatePanel1`, the ID of the `UpdatePanel`.

```

</ContentTemplate>
</asp:UpdatePanel>

<asp:UpdateProgress ID="UpdateProgress1" runat="server" AssociatedUpdatePanelID="UpdatePanel1">
</asp:UpdateProgress>

```



10. Between the `<UpdateProgress>` tags create a `<ProgressTemplate>`, and within this template, create a `<div>` element with its class attribute set to `PleaseWait`, the CSS class you created in step 7. Inside the `<div>` element, type some text (**Please wait...**) to inform your users that they should hold on for a while. You should end up with this code:

```

<asp:UpdateProgress ID="UpdateProgress1" runat="server" AssociatedUpdatePanelID="UpdatePanel1">
  <ProgressTemplate>
    <div class="PleaseWait">
      Please Wait...
    </div>
  </ProgressTemplate>
</asp:UpdateProgress>

```

11. To emulate a long delay while sending out the message so you can see the `UpdateProgress` control, add the following line of code to the Code Behind of the control, just after the lines that change the visibility of the controls in the method that sends out the e-mail:

```
Message.Visible = true;
FormTable.Visible = false;
➡ System.Threading.Thread.Sleep(5000);
```

12. Save all your changes and open the page `Contact.aspx` from the `About` folder once again. Fill in the required details and click the `Send` button. Shortly after you click the button, you should see the `UpdateProgress` control appear that displays text and an animated image below the form, shown in the figure. Shortly after that, the `UpdateProgress` and the entire form should disappear and you should be presented with the **Message Sent** text.

Visitors can use this form to get in touch with me.

Name	WBWP
Email Address	wbwp@gmail.com
Email Address(Repeat)	wbwp@gmail.com
Home Phone	3000000
Business Phone	3000000
Comments	<div>Bla bla bla</div>
<div>Send</div>	

Please Wait...

Visitors can use this form to get in touch with me.

Name

wbwp

Email Address

wbwp@gmail.com

Email Address(Repeat)

wbwp@gmail.com

Home Phone

3000000

Business Phone

3000000

Comments

bla bla

Send



Please Wait...

Adding Timer Control

1. Create a new Web Form called TimerDemo.aspx using your custom template. Give the page a Title of Timer Control Demo.
2. Switch the new page into Design View and drag an UpdatePanel into the cpMainContent area.
3. Drag a Label control into the UpdatePanel.
4. Switch to Source View, inside the UpdatePanel, drag a Timer control, below the Label control, from AJAX Extensions of Toolbox. Remove the Text property of the Label control.
5. Add the Interval and OnTick properties for the Timer control. You should end up with this code:

```
<asp:Content ID="Content2" ContentPlaceHolderID="cpMainContent" Runat="Server">
  <asp:UpdatePanel ID="UpdatePanel1" runat="server">
    <ContentTemplate>
      <asp:Label ID="Label1" runat="server"></asp:Label>
      <asp:Timer ID="Timer1" runat="server" Interval="1000" OnTick="Timer1_Tick"></asp:Timer>
    </ContentTemplate>
  </asp:UpdatePanel>
</asp:Content>
```

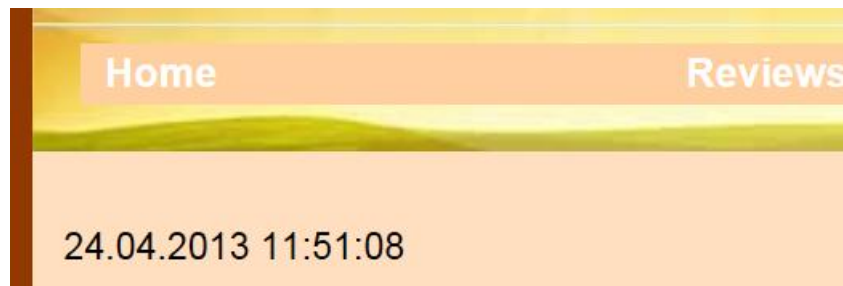
6. Switch to Design View and double click the Timer control. Add the following code to code behind:

```
protected void Timer1_Tick(object sender, EventArgs e)
{
    Label1.Text = System.DateTime.Now.ToString();
}
```

7. You can also add the same code to the Page_Load method to see the current date and time when the page first loads:

```
public partial class _TimerDemo : BasePage
{
    protected void Page_Load(object sender, EventArgs e)
    {
        Label1.Text = System.DateTime.Now.ToString();
    }
    protected void Timer1_Tick(object sender, EventArgs e)
    {
        Label1.Text = System.DateTime.Now.ToString();
    }
}
```

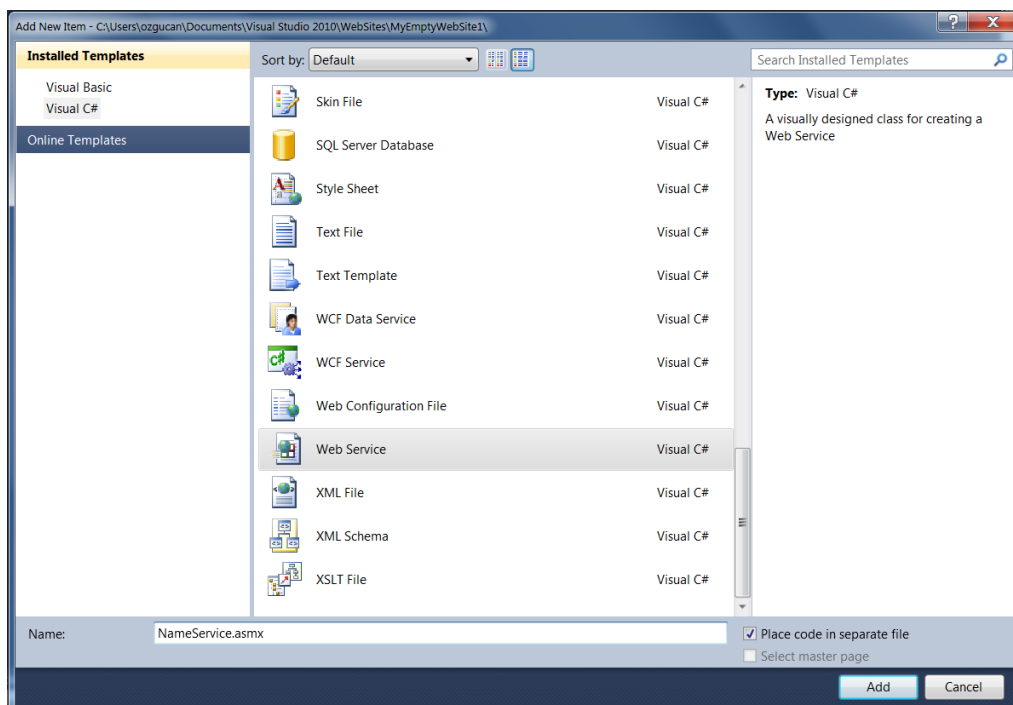
8. Save and view the page in the browser. When this code is run in the browser, the `Label` will be updated with the current date and time in every second.



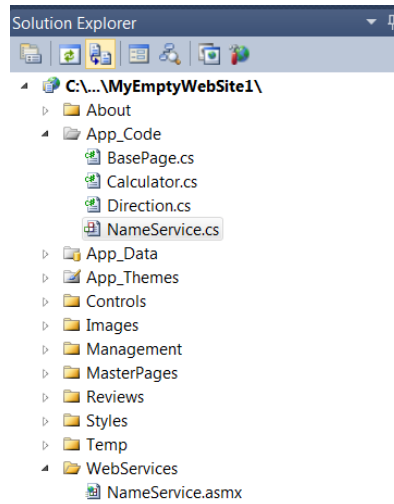
Creating a Web Service

In this exercise you create a simple “*Hello World*” web service. This service accepts your name as an input parameter and returns a friendly, personalized greeting. There’s not much real-world usage for this exact web service, but because of the simplicity in the service itself, it’s easy for you to focus on the underlying concepts.

1. Create a new folder called `WebServices` in the root of your site to group all web services in the site in a single folder. This is not required, but helps in organizing your site.
2. Next, right-click this new folder and choose Add New Item. Click the `Web Service` item (don’t accidentally choose `WCF Service` as that results in a different type of service), make sure that your preferred programming language and Place Code in Separate File are selected, and call the service **NameService**.



3. Click Add to add the service to the site. Notice how the .asmx file is added to the WebServices folder and the Code Behind file is placed in the site's App_Code folder.



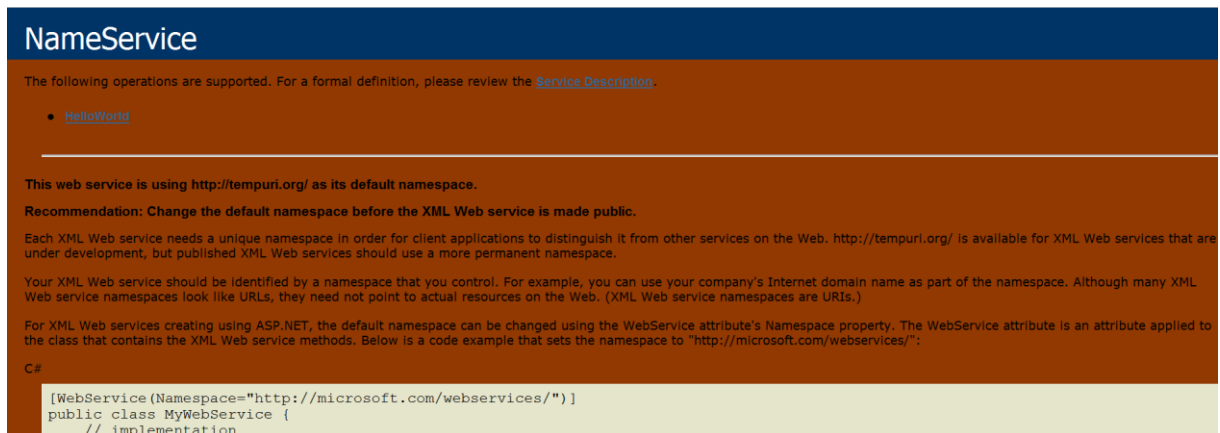
4. Open the NameService Code Behind file from the App_Code folder and change the code for the HelloWorld method so it accepts a string and returns a personalized greeting.

```
[WebMethod]
public string HelloWorld() {
    return "Hello World";
}
```

You should end up with code like this:

```
[WebMethod]
public string HelloWorld(string yourName) {
    return string.Format("Hello {0}", yourName);
}
```

5. Save all your changes, right-click `NameService.asmx` in the Solution Explorer, and choose View in Browser. Once the browser is done loading, you get a page that lists all the public web services defined in the `NameService.asmx` service. In this exercise, you should only see `HelloWorld` as shown in the figure:



NameService

The following operations are supported. For a formal definition, please review the [Service Description](#).

- [HelloWorld](#)

This web service is using `http://tempuri.org/` as its default namespace.

Recommendation: Change the default namespace before the XML Web service is made public.

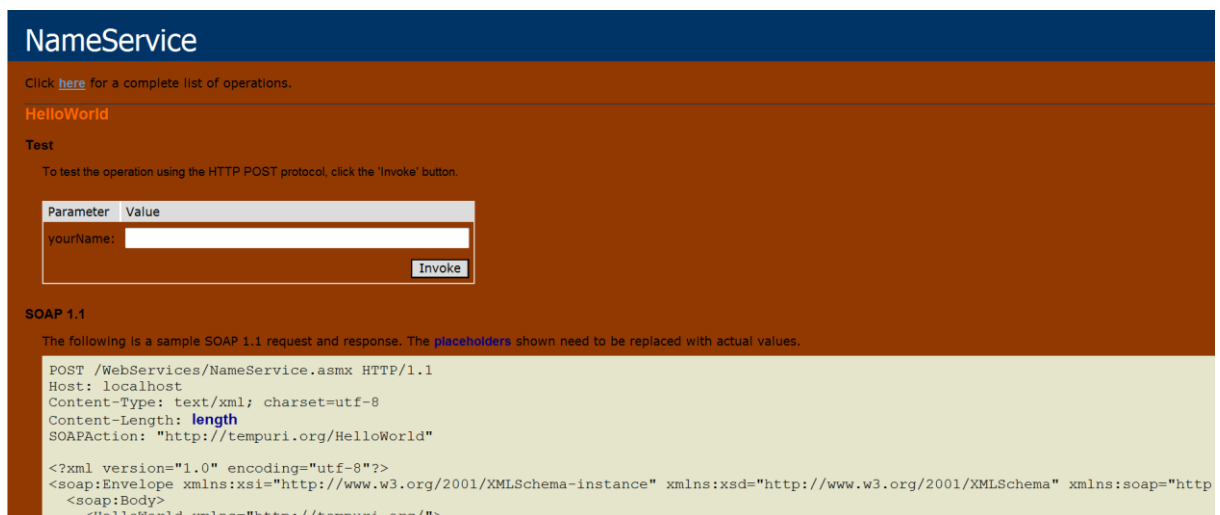
Each XML Web service needs a unique namespace in order for client applications to distinguish it from other services on the Web. `http://tempuri.org/` is available for XML Web services that are under development, but published XML Web services should use a more permanent namespace.

Your XML Web service should be identified by a namespace that you control. For example, you can use your company's Internet domain name as part of the namespace. Although many XML Web service namespaces look like URLs, they need not point to actual resources on the Web. (XML Web service namespaces are URIs.)

For XML Web services creating using ASP.NET, the default namespace can be changed using the `WebService` attribute's `Namespace` property. The `WebService` attribute is an attribute applied to the class that contains the XML Web service methods. Below is a code example that sets the namespace to `"http://microsoft.com/webservices/"`:

```
C#
[WebService(Namespace="http://microsoft.com/webservices/")]
public class MyWebService {
    // implementation
}
```

6. Click the `HelloWorld` link and you are taken to a page where you can test out the service.



NameService

Click [here](#) for a complete list of operations.

HelloWorld

Test

To test the operation using the HTTP POST protocol, click the 'Invoke' button.

Parameter	Value
yourName:	<input type="text"/>

SOAP 1.1

The following is a sample SOAP 1.1 request and response. The [placeholders](#) shown need to be replaced with actual values.

```
POST /WebServices/NameService.asmx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://tempuri.org/HelloWorld"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <HelloWorld xmlns="http://tempuri.org/">
```

Type your name in the `yourName` field and click `Invoke`. A new window opens, showing the XML that has been returned by the web service.

```
<?xml version="1.0" encoding="UTF-8"?>
<string xmlns="http://tempuri.org/">Hello Özgü</string>
```

Calling Web Services from Client-Side Code

In this exercise you register your web service in a `ScriptManagerProxy` control so it becomes available in one page only. In addition, you modify the service so its methods are accessible by script. Finally, you write some client-side JavaScript code that accesses the service and then displays its return value.

1. The first thing you need to do is add the `ScriptService` attribute to your service class to mark it as callable by client-side script. To do this, open the file `NameService.cs` from the `App_Code` folder and uncomment the line that defines the attribute. You should end up with this code:

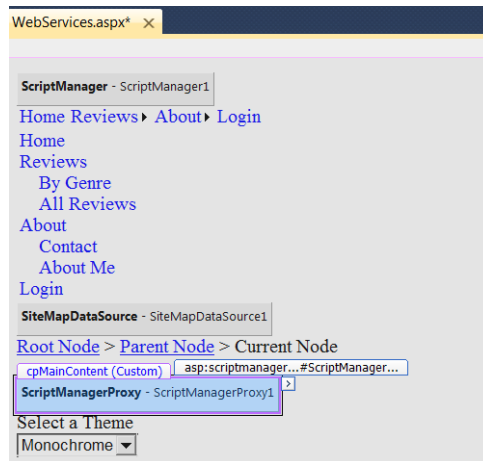
```
[System.Web.Script.Services.ScriptService]
public class NameService : System.Web.Services.WebService {
```

2. While you're at it, change the `Namespace` property of the `WebService` attribute. By default, the namespace looks like this:

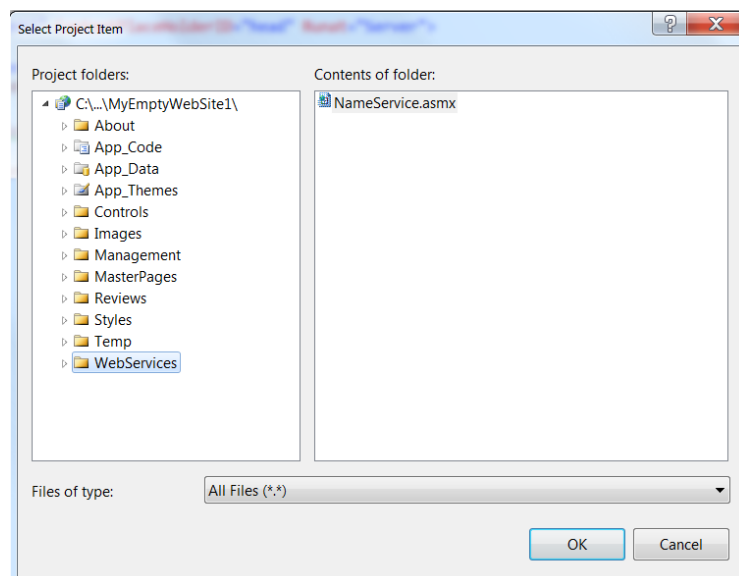
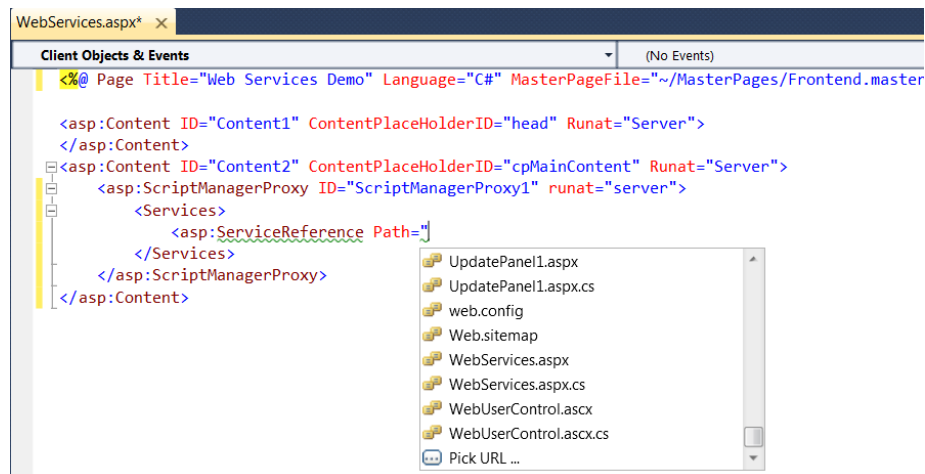
```
[WebService(Namespace = "http://tempuri.org/")]
```

Although this name is fine during development of your web services, it should really reflect the unique name of your service when you put it in a production environment in order to avoid possible conflicts with other services carrying the same names or types. If you have your own domain name, you can change the namespace to something like `http://www.yourdomain.com/`. If you don't have your own domain, don't worry about it. Even with the `Namespace` set to the default value of `http://tempuri.org/`, things will work fine.

3. The next step is creating a page that uses the service and then registers it using a `ScriptManagerProxy` control. Add a new Web Form in the project folder and call it `WebServices.aspx`. Make sure you base this page on your custom template, so it has the correct master page set and inherits from the `BasePage` class, and then give it a `Title` such as `Web Services Demo`. Once you've added the page, drag a `ScriptManagerProxy` control from the `AJAX Extensions` category of the `Toolbox` into the markup of the `cpMainContent` placeholder.



4. Within the ScriptManagerProxy element, add a `<Services>` element that in turn contains a `ServiceReference` with its `Path` set to the `NameService` you created earlier. Note that IntelliSense helps you pick the right file as soon as you type `Path=` by showing you a list with files. Click `Pick URL` at the bottom of the list and browse to the service file in the `WebServices` folder.



You should end up with this code in the `WebServices.aspx` page:

```
<asp:Content ID="Content2" ContentPlaceHolderID="cpMainContent" Runat="Server">
  <asp:ScriptManagerProxy ID="ScriptManagerProxy1" runat="server">
    <Services>
      <asp:ServiceReference Path="~/WebServices/NameService.asmx" />
    </Services>
  </asp:ScriptManagerProxy>
</asp:Content>
```

5. Right below the closing tag of the `<ScriptManagerProxy>`, add an `Input (Text)` and an `Input (Button)` by dragging them from the HTML category of the Toolbox. By using plain HTML elements and not ASP.NET Server Controls, you can see that the code you are going to write really executes at the client. Set the `id` of the text box to `YourName` and the `id` of the button to `SayHello`. Set the value of the button to `Say Hello`. You should end up with this markup:

```
</asp:ScriptManagerProxy>
<input id="YourName" type="text" />
<input id="SayHello" type="button" value="Say Hello" />
```

6. Below these two lines, add a client-side JavaScript block with the following code:

```
<input id="SayHello" type="button" value="Say Hello" />
<script type="text/javascript">
  function HelloWorld()
  {
    var yourName = $get('YourName').value;
    NameService.HelloWorld(yourName, HelloWorldCallback);
  }
  function HelloWorldCallback(result)
  {
    alert(result);
  }
  $addHandler($get('SayHello'), 'click', HelloWorld);
</script>
</asp:Content>
```


7. Open the `NameService.cs` file at the `App_Code` directory. Change `HelloWorld` method to:

```
[WebMethod]
public string HelloWorld(string yourName) {
    return string.Format("Hello {0}", yourName);
}
```

8. Save all your changes by pressing `Ctrl+Shift+S`, and then request the page `WebServices.aspx` in your browser. Enter your name and click the `Say Hello` button. If everything turned out well, you should be greeted with a message from the web service, repeating your name.



COMMON MISTAKES If you get an error instead of this message box, or you see a small yellow triangle in the bottom-left corner of the screen, make sure you typed the JavaScript exactly as in the code snippet. JavaScript is *case sensitive*, so make sure you get all the capitalization right. Also make sure that the JavaScript block you added in step 6 comes after the input box and button that you defined earlier. Finally, make sure that the path to your web service matches the actual path of your `.asmx` service file and that you have applied the `ScriptService` attribute to the service class.

Calling Page Methods from Client-Side Code

In this exercise, you modify the `WebServices.aspx` page and add a second button that calls a page method. To make it easy to compare the two techniques of calling code on the server, the page method you create is similar to the web service you called earlier.

1. Open up the master page `Frontend.master` in Source View and set the `EnablePageMethods` attribute of the `ScriptManager` control to `True`:

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

2. Open the Code Behind of `WebServices.aspx` in the project folder and add the following serverside method within the `WebServices` class:

```
public partial class _WebServices : BasePage
{
    protected void Page_Load(object sender, EventArgs e)
    {

    }

    [WebMethod]
    public static string HelloWorld(string yourName)
    {
        return string.Format("Hello {0}", yourName);
    }
}
```

Notice how this code is almost identical to what you defined in the web service, including the `WebMethod` attribute. The only difference is the inclusion of the `static`.

The `Webmethod` attribute won't be recognized directly. To fix this, type the following `using` statement at the top of the page, below the other statements:

```
using System.Web.Services;
```

```

[WebMethod]
public static string HelloWorld(string yourName)
{
    return string.Format("Hello {0}", yourName);
}

```

Alternatively, click the attribute once in the code and then press Ctrl+. (Ctrl+Dot) to bring up a list with suggested options and choose the first item to have the code inserted for you.

3. Switch to Source View and create a copy of the HTML button you created earlier, set its id to SayHelloPageMethod and change its value to better describe what the button does. You should end up with code like this:

```

<input id="SayHello" type="button" value="Say Hello" />
➡ <input id="SayHelloPageMethod" type="button" value="Say Hello wit a Page method" />
<script type="text/javascript">
    function HelloWorld()
    {

```

4. Set up a handler for the button's client click event, similar to the one you created earlier. Use HelloWorldPageMethod as the client method to call:

```

$.addHandler($get('SayHelloPageMethod'), 'click', HelloWorldPageMethod);

```

5. Implement the HelloWorldPageMethod method as follows:

```

function HelloWorldPageMethod()
{
    var yourName = $get('YourName').value;
    PageMethods.HelloWorld(yourName, HelloWorldCallback);
}

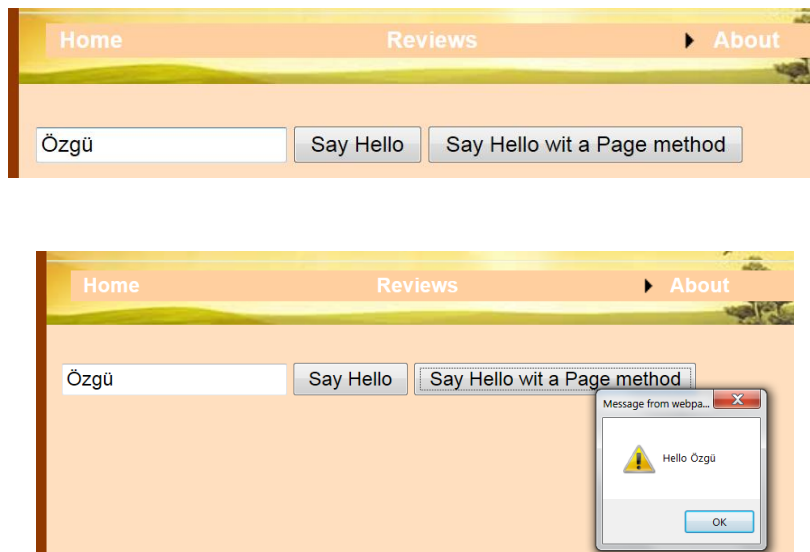
```

You can add the method directly below the HelloWorld function, in the same script block. Notice there's no need to write a new callback method to handle the return value of the call to HelloWorld. The one you created in the web service example can easily be reused because all it does is simply alert the return value.

You should end up with this code:

```
<script type="text/javascript">
    function HelloWorld()
    {
        var yourName = $get('YourName').value;
        NameService.HelloWorld(yourName, HelloWorldCallback);
    }
    function HelloWorldPageMethod() {
        var yourName = $get('YourName').value;
        PageMethods.HelloWorld(yourName, HelloWorldCallback);
    }
    function HelloWorldCallback(result)
    {
        alert(result);
    }
    $addHandler($get('SayHello'), 'click', HelloWorld);
    $addHandler($get('SayHelloPageMethod'), 'click', HelloWorldPageMethod);
</script>
```

6. Save all your changes and press Ctrl+F5 to run the page in the browser. Enter your name and click the Say Hello with a Page Method button. You should see the same message as you saw with the web service example.



COMMON MISTAKES If you get an error about PageMethods not being defined, make sure you added the `static` keyword to the method's signature and make sure you set `EnablePageMethods` to `True` on the `ScriptManager` control in the master page.