

ASP.NET AJAX

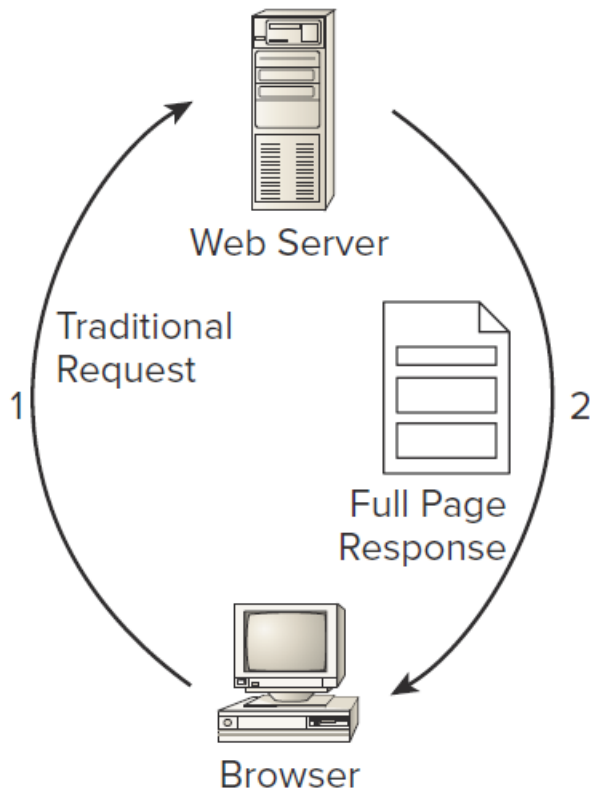
Asst. Prof. Dr. Özgü Can

ASP.NET AJAX

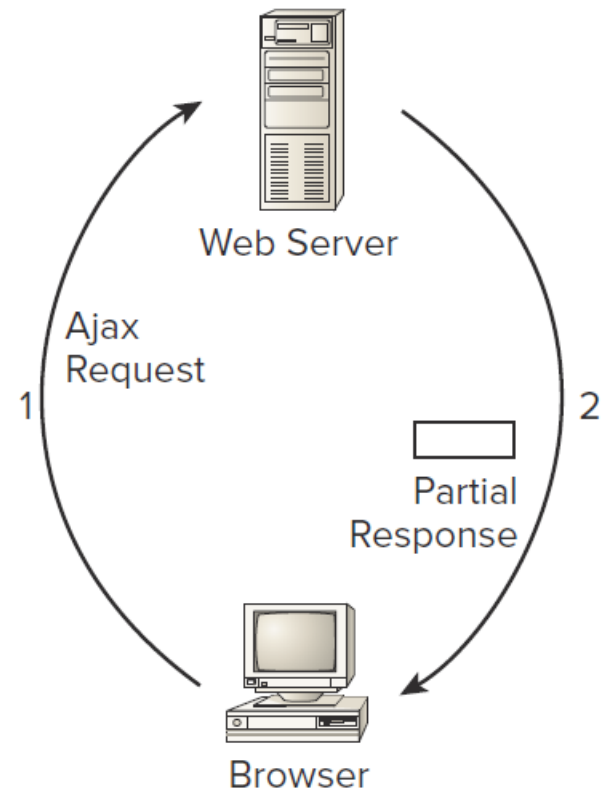
- Asynchronous JavaScript And XML
- Enables *client-side* web pages to exchange data with the *server* through asynchronous calls.
- **Flicker-free page**
 - Enables to **perform a postback** to the server **without refreshing the entire page.**
- Gives more server controls to create *rich*, *interactive*, and *responsive user interfaces*.

AJAX

Traditional Page Processing



Ajax Page Processing



ASP.NET AJAX

- Enables you to:
 - **Create flicker-free pages** that enable you to refresh portions of the page **without a full reload and without affecting other parts of the page**.
 - **Provide feedback** to your users **during these page refreshes**.
 - **Update** sections of a page and **call** server-side code on a scheduled basis **using a timer**.
 - **Access** server-side web services and page methods and **work with the data** they return.
 - **Use** the **rich, client-side programming** framework to access and modify elements in your page, and get access to a code model and type system that looks similar to that of the .NET Framework.

AJAX Extensions

▲ AJAX Extensions



Pointer



ScriptManager



ScriptManagerProxy



Timer



UpdatePanel



UpdateProgress

Creating Flicker-Free Pages

- `UpdatePanel` → To avoid **full postbacks** in ASPX pages and **update only** part of the page.
- *Whenever* one of the controls *within* the `UpdatePanel` causes a **postback** to the server, **only** the content *within* that `UpdatePanel` is refreshed.
- For this control to operate correctly;
 - **ScriptManager** → Bridge between the client page and the server

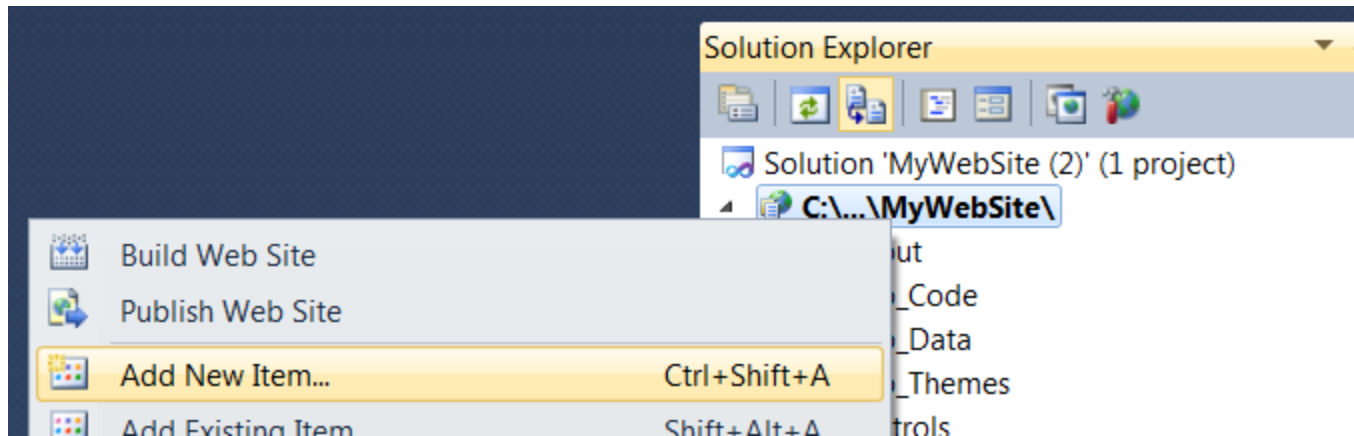
ScriptManager

- **Place** the **ScriptManager** in the **master** page
 - If you're going to use Ajax functionality in many of your ASPX pages → It's available in all pages that are based on this master.
- You can only have **one** ScriptManager per page
 - *If* you add **one** to a **master** page, you **can't** add another one to a **content** page.
- **ScriptManagerProxy** → In order to access a **ScriptManager** control that is defined in a **master** page *from* a **content** page.

Example

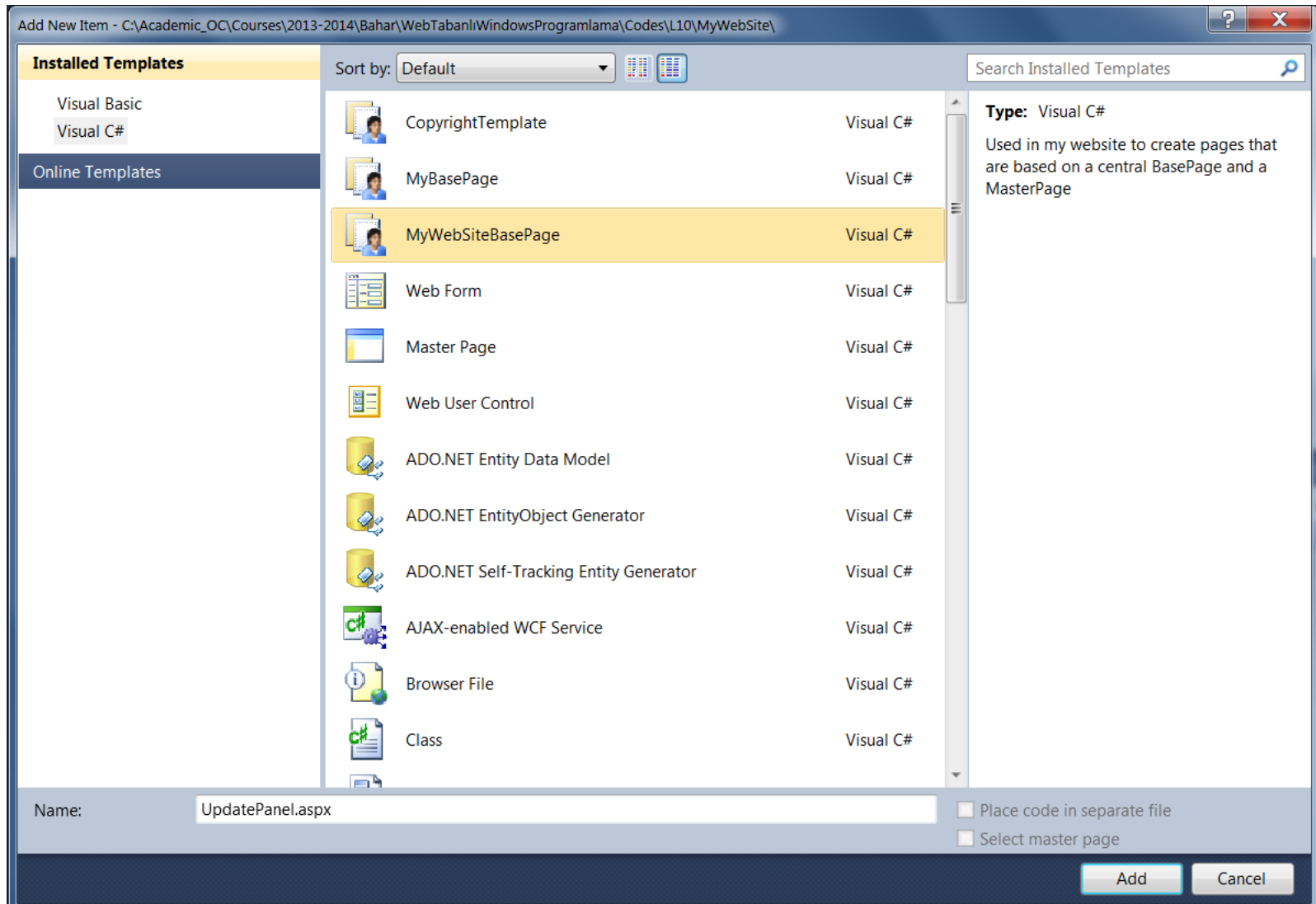
Adding UpdatePanel

- Add New Item



Example

Adding UpdatePanel



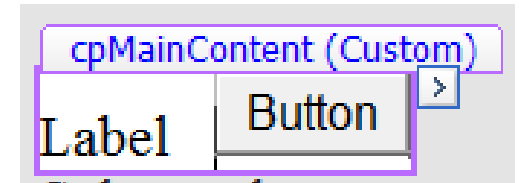
Example

Adding UpdatePanel

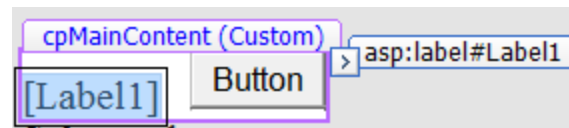
- **Title** → **Update Panel Demo**

- In Design View:

- Drag a **Label** and a **Button**



- Clear the **Text** property of the **Label**



Example

Adding UpdatePanel

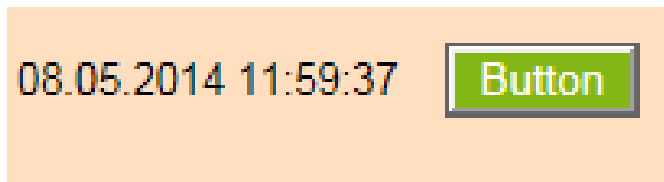
- Double-click the grey area:

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

Example

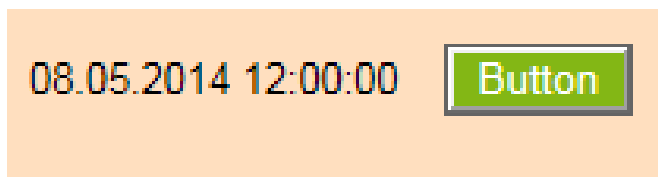
Adding UpdatePanel

- View in browser



- Click the **Button** control a few times.

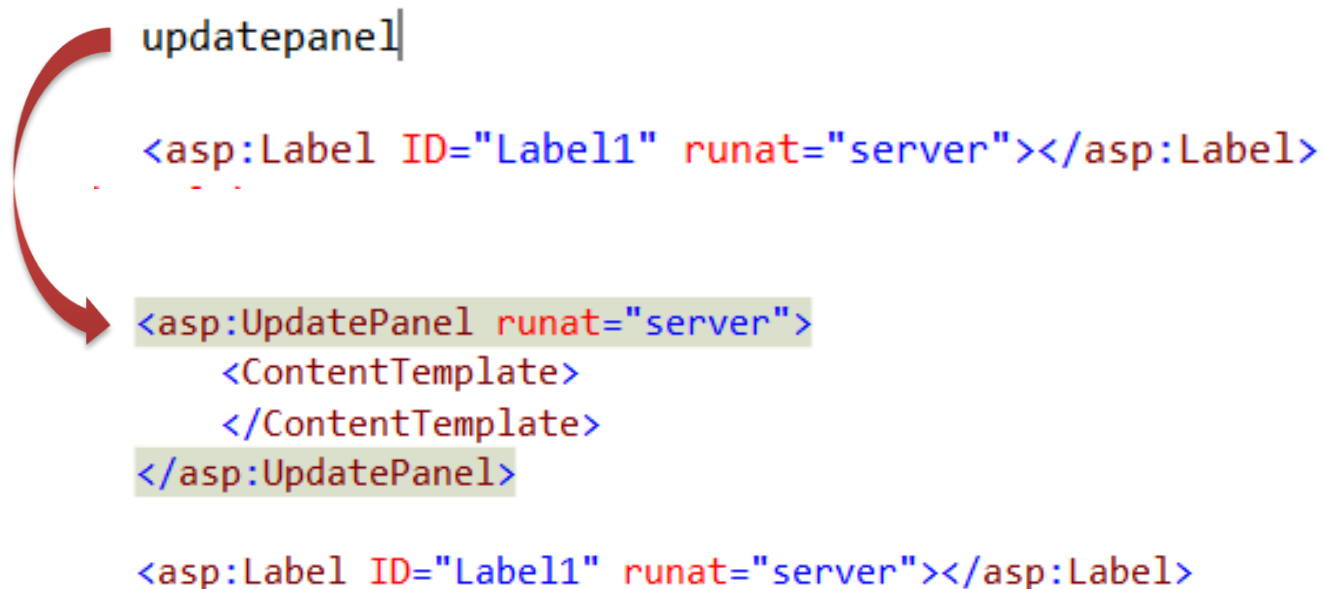
Note that each time you click the button, the page flickers and is then redrawn, displaying the updated date and time.



Example

Adding UpdatePanel

- In Source View:
 - Make some room right before the **Label** control, and then type **updatepanel** and press Tab.



The diagram illustrates the process of adding an `UpdatePanel` control to an ASP.NET page. A red curved arrow points from the text `updatepanel` to the `<asp:UpdatePanel runat="server">` tag in the code snippet below. The code snippet shows the `UpdatePanel` being inserted before the `<asp:Label ID="Label1" runat="server"></asp:Label>` control.

```
updatepanel|  
  
<asp:Label ID="Label1" runat="server"></asp:Label>  
.  
.  
.  
<asp:UpdatePanel runat="server">  
    <ContentTemplate>  
    </ContentTemplate>  
</asp:UpdatePanel>  
  
<asp:Label ID="Label1" runat="server"></asp:Label>
```

Example

Adding UpdatePanel

- Cut & paste **Label** and **Button** definitions between **<ContentTemplate>** and **</ContentTemplate>**

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

Example

Adding UpdatePanel

Drag a **ScriptManager** before **UpdatePanel**

or

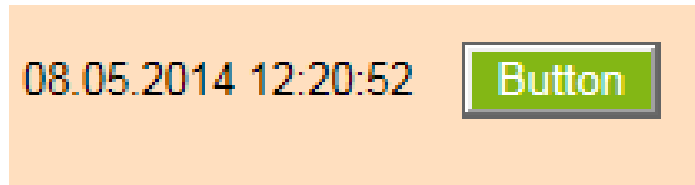
Type **sm** and press Tab

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

Example

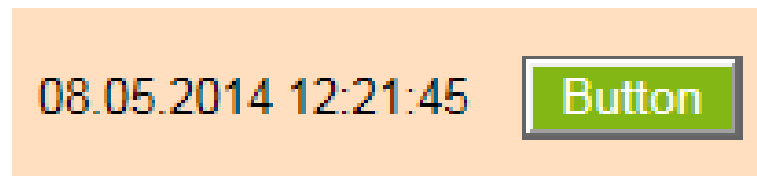
Adding UpdatePanel

- View in browser



- Click the button a few times to update the label with the current date and time.

Note that there is no page flicker now.



UpdateProgress

- Postback's advantage → the user can see *something is happening*
- **UpdatePanel** → Users have *no visual cue* that something is happening until it has happened.
- **UpdateProgress**
 - Tells users to hold on for a few seconds while their request is being processed.

UpdateProgress

- Connect the **UpdateProgress** control to an **UpdatePanel**:
 - **AssociatedUpdatePanelID** property
 - Text such as “**Please wait**” or an animated image to let the user know something is happening

Example

- Open **ContactForm.ascx** in **Controls** folder.
- In Source View → Wrap the entire **<table>** element & the **Label** at the bottom of the control in an **UpdatePanel** with a **<ContentTemplate>**.

Add ID

```
<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>
```

Example

- Open → **Frontend.master**
- Between the opening **<form>** tag and the **<div>** for the **PageWrapper**, add a **ScriptManager** control.



```
<body>
  <form id="form1" runat="server">
    <div id="PageWrapper">
      <div id="Header"><a href="/" runat="server"></a></div>

<body>
  <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server" />
    <div id="PageWrapper">
      <div id="Header"><a href="/" runat="server"></a></div>
```

Example

- Open → **UpdatePanel.aspx**
- Remove **ScriptManager** Control

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

[illegible]

Example

- View `Contact.aspx` in browser

The image shows a web browser displaying a contact form. The form is titled "Home > About > Contact" and includes a message: "Visitors can use this form to get in touch with me." The form fields are: Name, Email Address, Email Address (Repeat), Home Phone, and Business Phone. Below these is a large text area for comments. A green "Send" button is at the bottom left. A purple arrow points from the "Send" button to a confirmation message box on the right. The confirmation message box also displays "Home > About > Contact" and "Message Sent!"

Home > About > Contact

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

Name

Email Address

Email Address (Repeat)

Home Phone

Business Phone

Comments

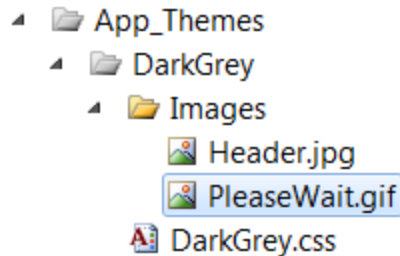
Send

Home > About > Contact

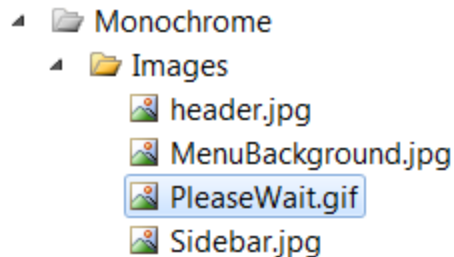
Message Sent!

Example

- In **App_Themes** folder:
 - Add **PleaseWait.gif** → **Monochrome** > **Images**



- Add **PleaseWait.gif** → **DarkGrey** > **Images**



Example

- In **Monochrome.css** and **DarkGrey.css**:

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


Example

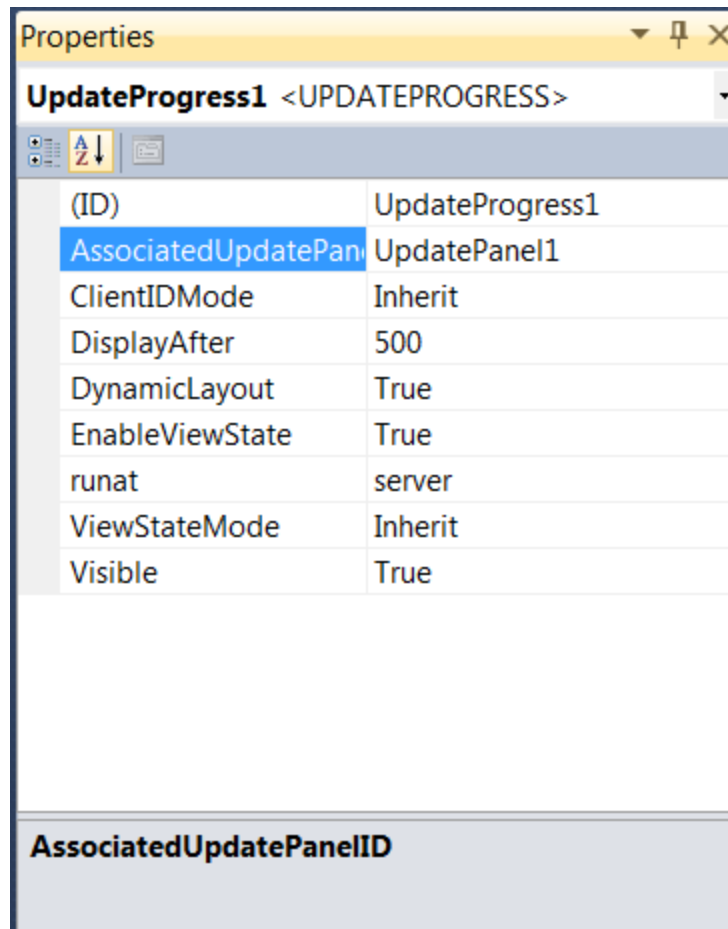
- Open → **ContactForm.ascx** in **Controls** folder
- Below the closing tag of the **UpdatePanel** at the end of the file, drag an **UpdateProgress** control.

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

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

Example

AssociatedUpdatePanelID = UpdatePanel1



The screenshot shows the Visual Studio Properties window for a control named **UpdateProgress1** of type **<UPDATEPROGRESS>**. The window has a title bar with a dropdown arrow, a pin icon, and a close icon. Below the title bar is a toolbar with icons for Show All Properties, Show Default Properties, and a Sort button (A-Z with a downward arrow). The main area is a table of properties and their values. The property **AssociatedUpdatePan** is highlighted in blue. At the bottom of the window, there is a section labeled **AssociatedUpdatePanelID**.

UpdateProgress1 <UPDATEPROGRESS>	
(ID)	UpdateProgress1
AssociatedUpdatePan	UpdatePanel1
ClientIDMode	Inherit
DisplayAfter	500
DynamicLayout	True
EnableViewState	True
runat	server
ViewStateMode	Inherit
Visible	True

AssociatedUpdatePanelID

Example

In Source View:


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

Example

- Open → `ControlForm.ascx.cs`

```
Message.Visible = true;
```

```
FormTable.Visible = false;
```

To emulate a long delay  `System.Threading.Thread.Sleep(5000);`

Example

- View `Contact.aspx` in browser

Home > About > Contact

Select a theme
DarkGrey ▼

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

Name


Email Address


Email Address (Repeat)

Home Phone

Business Phone

Comments



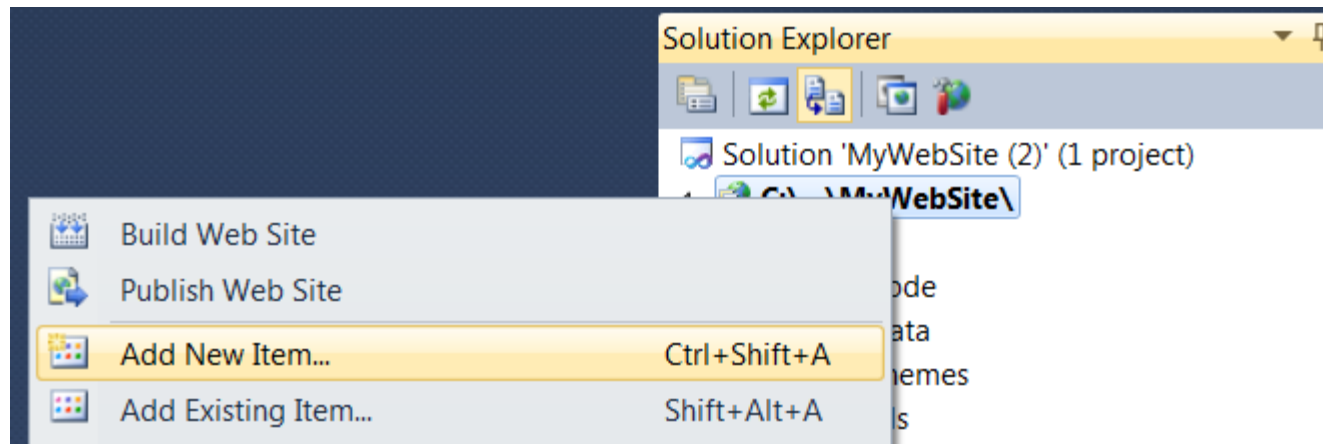
 Please Wait...

Timer

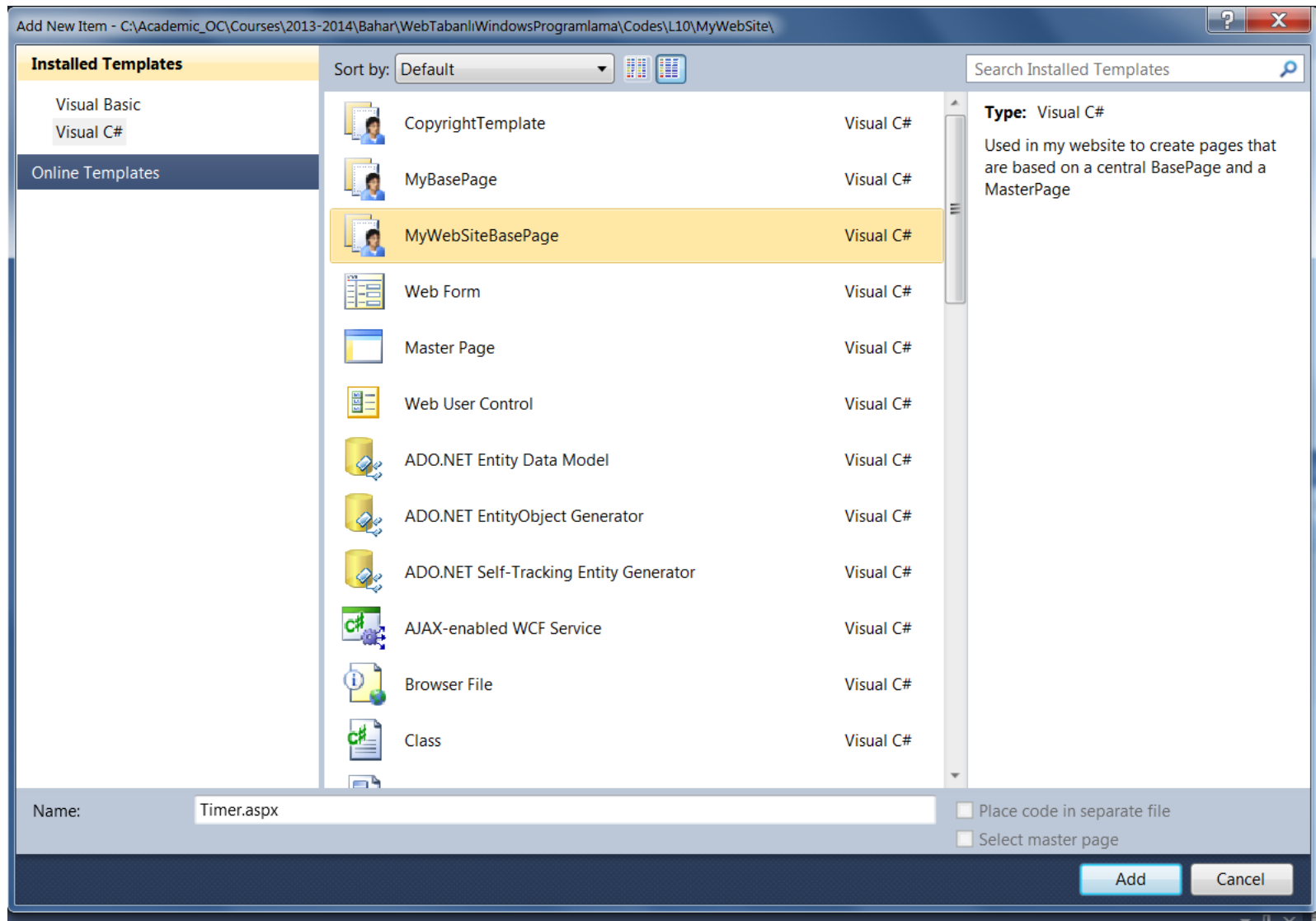
- Great for executing server-side code **on a repetitive basis**.
 - *For example*, you can use it to update the contents of an **UpdatePanel** every 5 seconds.
- For more information check out MSDN's web page → <http://tinyurl.com/TimerClass>

Example Timer

- Add New Item



Example Timer



Example Timer

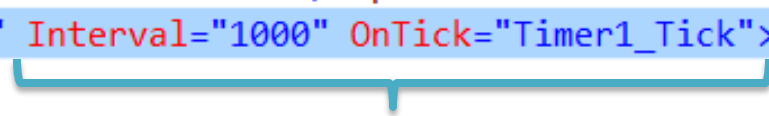
- **Title = Timer Demo**
- In Design View:
 - Drag an **UpdatePanel**
 - Drag a **Label** inside **UpdatePanel**

```
<asp:Content ID="Content2" ContentPlaceHolderID="cpMainContent" Runat="Server">  
  <asp:UpdatePanel ID="UpdatePanel1" runat="server">  
    <ContentTemplate>  
      <asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>  
    </ContentTemplate>  
  </asp:UpdatePanel>  
</asp:Content>
```

Example Timer

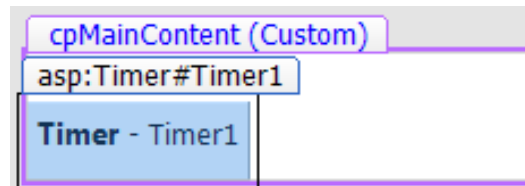
- Drag a **Timer** control below the **Label**

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

A blue bracket is drawn under the `<asp:Timer ID="Timer1" runat="server" Interval="1000" OnTick="Timer1_Tick">` line, indicating its placement relative to the `<asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>` line above it. The entire code block is highlighted with a light blue background.

Example Timer

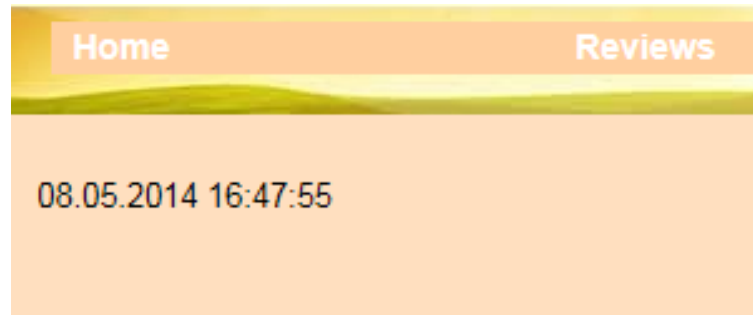
- In Design View → Double-click the **Timer** control



```
protected void Timer1_Tick(object sender, EventArgs e)
{
    Label11.Text = System.DateTime.Now.ToString();
}
```

Example Timer

- View in browser



Web Services

- Methods that you can call over the Internet and that can optionally return data to the calling code.
- Ideal for exchanging data between *between different* types of *platforms* and *systems*.
 - For example;
 - exchange data between an **ASP.NET web site** *running on Microsoft Windows* and a **PHP-based site** *running on Linux*.
 - exchange data between an **ASP.NET web site** and a **client browser** *using JavaScript*.

Web Services

- To expose a method as a service:
 - **WebMethod** attribute
- An **attribute** is like a little tag or label that you can stick on code elements, like methods, properties, and so on, to **mark *that piece of code as something special***.

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

Signal to the ASP.NET runtime that you really want to expose this method **as a web method** that can be **called from client-side** code.

Web Services

- This also enables you to create other methods in the same class that are **not** exposed as **web services** automatically, giving you *flexibility in determining what to open up for the outside world*.
- Place this method in a file with an **.asmx** extension and inside a class that inherits from **System.Web.Services.WebService**.

Web Services

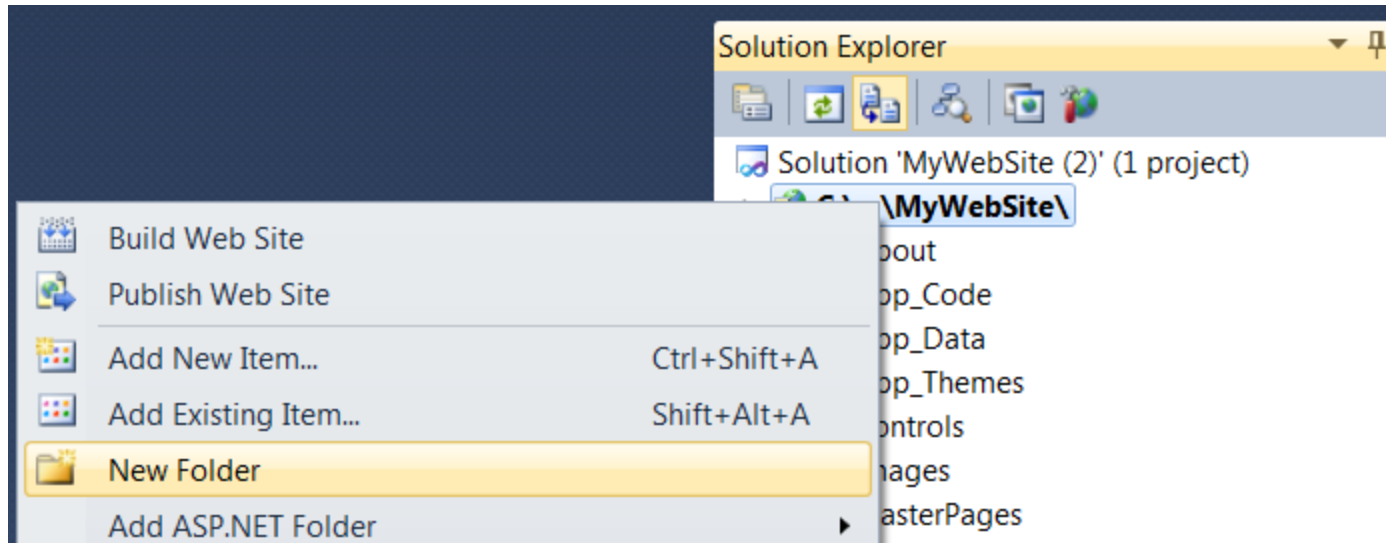
- You may want your **client-side** pages to **talk** to a **web service on a different domain**.
 - You need to *set up security* in the browser to allow this.
- You can also use **web services** to **have two servers or other applications communicate with each other**. In that case, one application interacts with an ASP.NET web service over the network to exchange data.

Outside the scope of this course!

Example

Creating a Web Service

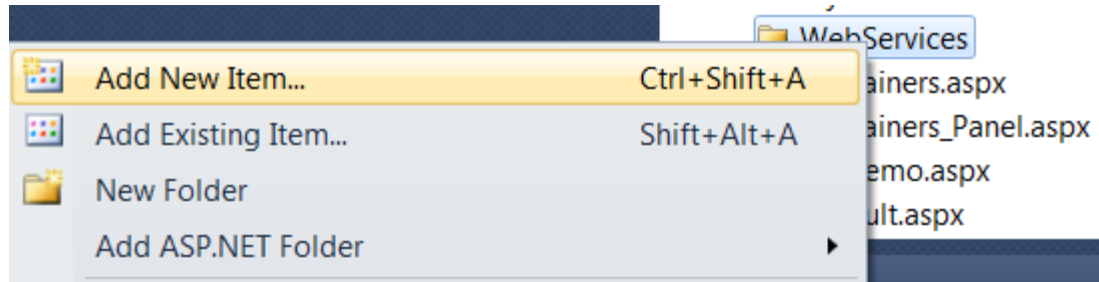
- Create a new folder → **WebServices**



Example

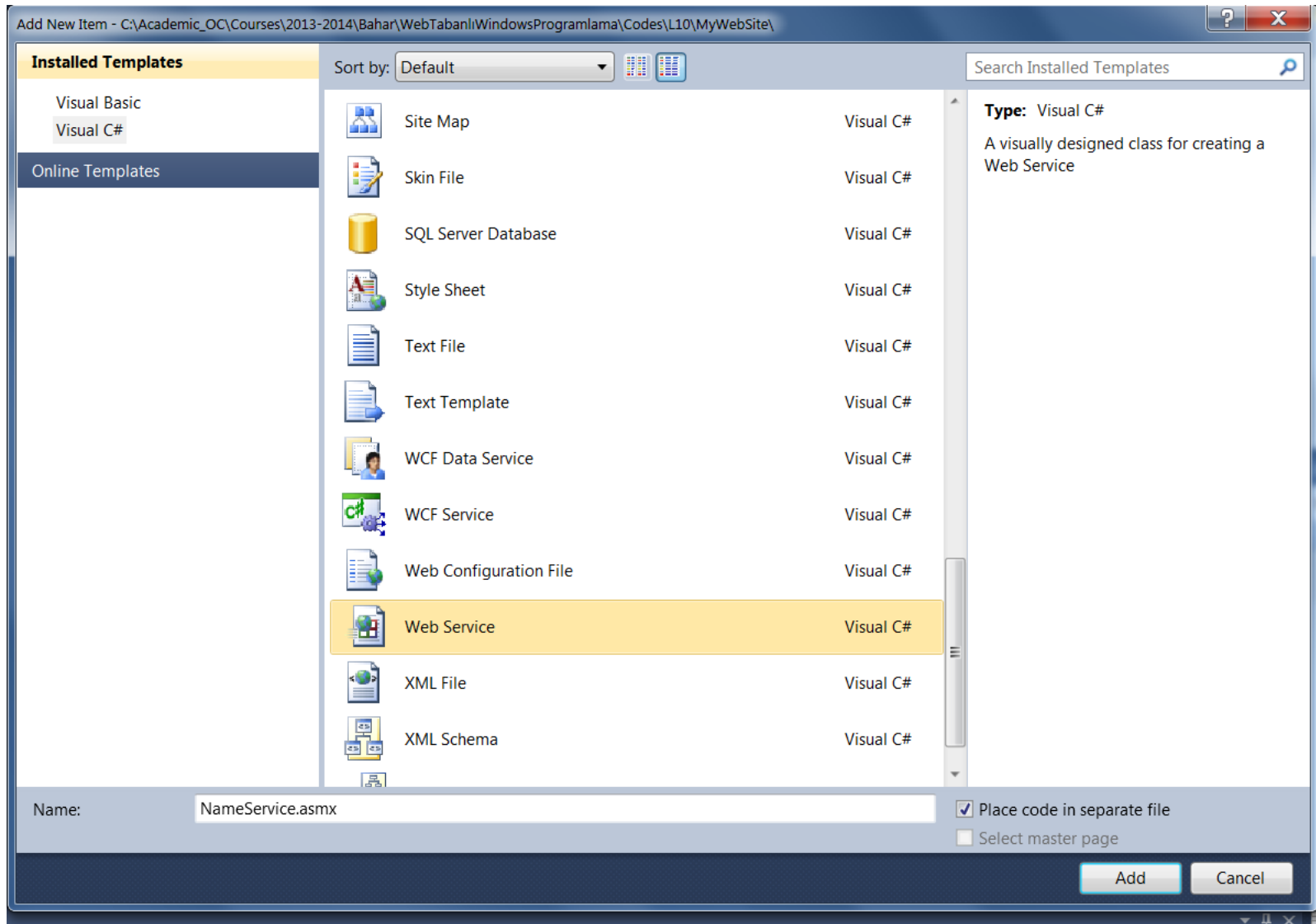
Creating a Web Service

- Add New Item



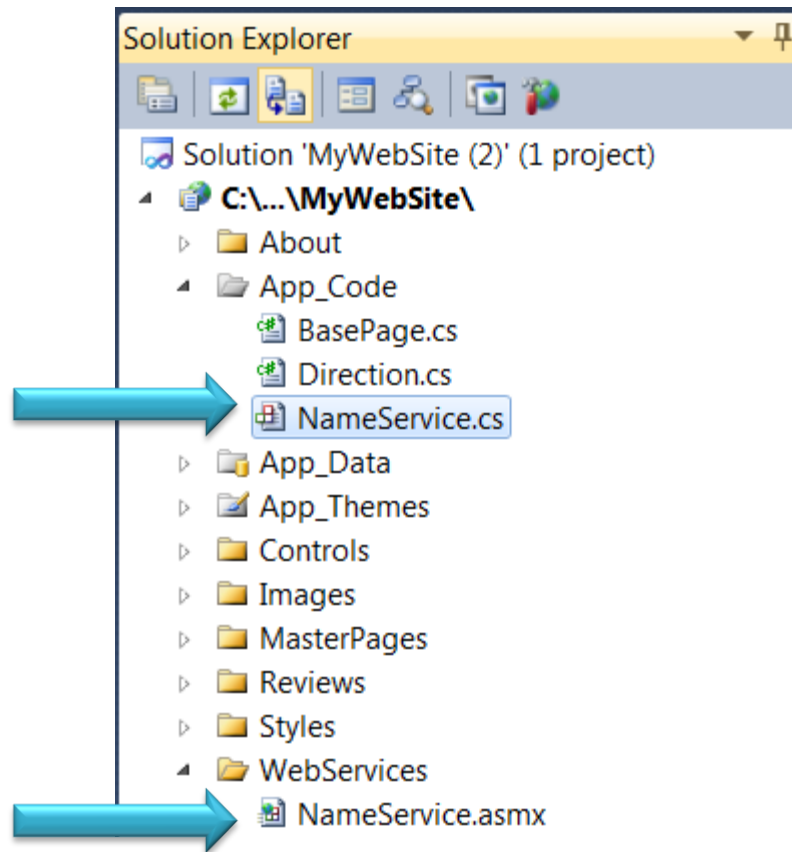
Example

Creating a Web Service



Example

Creating a Web Service



Example

Creating a Web Service

- Open → **NameService.cs**

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



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

Example

Creating a Web Service

- View **NameService.asmx** in browser.

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
}
```

Visual Basic

```
<WebService(Namespace:="http://microsoft.com/webservices/")> Public Class MyWebService
    ' implementation
End Class
```

C++

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

Example

Creating a Web 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" value="Özgü"/>

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 /MyWebSite/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/">
      <yourName>string</yourName>
    </HelloWorld>
  </soap:Body>
</soap:Envelope>
```

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

Configuring the Web Service

- To make a *web service* **visible by client-side script**:

- at the **NameService** class in the **App_Code** folder, you see that the template already added the attribute for you, but commented it out:

```
// To allow this Web Service to be called from script, using ASP.NET AJAX, uncomment the following line.  
// [System.Web.Script.Services.ScriptService]
```

- **Uncomment the line** to expose the entire service as a client-script service.

Configuring the ScriptManager

- A required component in almost all Ajax-related operations.
- To expose your web service to client script:
 - In the **ScriptManager** in the **master page**
 - or**
 - In a **content page** that uses the web service, using the **ScriptManagerProxy** class

Configuring the ScriptManager

*In the **ScriptManager** in the master page*

- Give the **ScriptManager** control a **<Services>** element that in turn contains one or more **ServiceReference** elements that point to your public services.

```
<asp:ScriptManager ID="ScriptManager1" runat="server">  
  <Services>  
    <asp:ServiceReference Path="~/WebServices/NameService.asmx" />  
  </Services>  
</asp:ScriptManager>
```

Configuring the ScriptManager

- On a normal page that **doesn't** use a master page with a **ScriptManager** you can simply **add** a **ScriptManager** to the Web Form **directly**.
- *However*, if you are **using** a **master page** that already has its **own ScriptManager** you **need** to use a **ScriptManagerProxy** control.

```
<asp:ScriptManagerProxy ID="ScriptManagerProxy1" runat="server">  
  <Services>  
    <asp:ServiceReference Path="~/WebServices/NameService.asmx" />  
  </Services>  
</asp:ScriptManagerProxy>
```

Example

Calling Web Services from Client-Side

- Open → **NameService.cs**
- To mark it as callable by client-side script:

Uncomment `// [System.Web.Script.Services.ScriptService]`

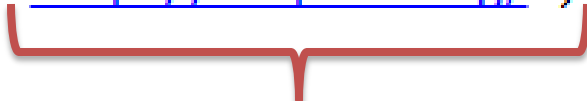


`[System.Web.Script.Services.ScriptService]`

NOTE

- If you have your own domain name:

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

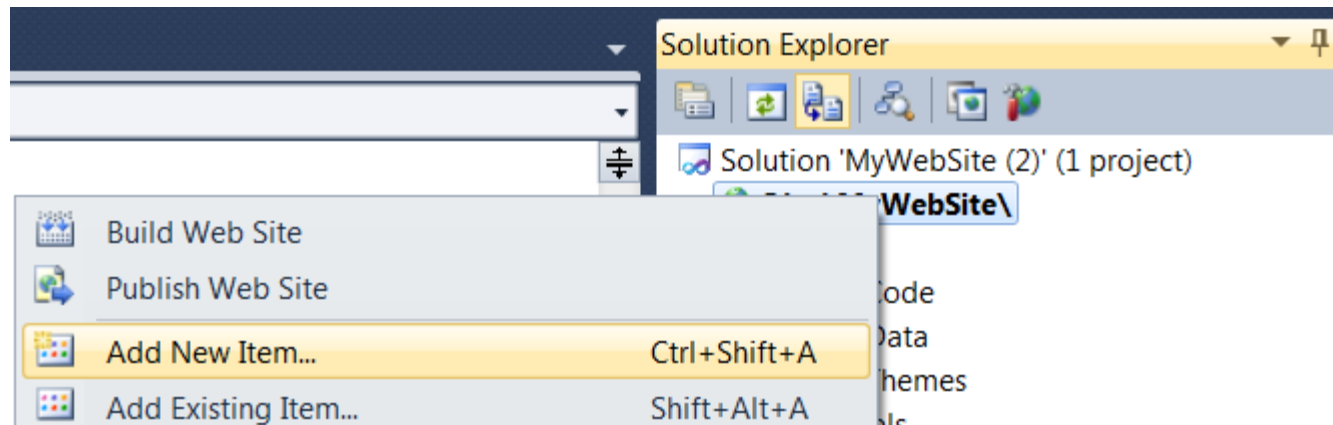


Change the namespace

Example

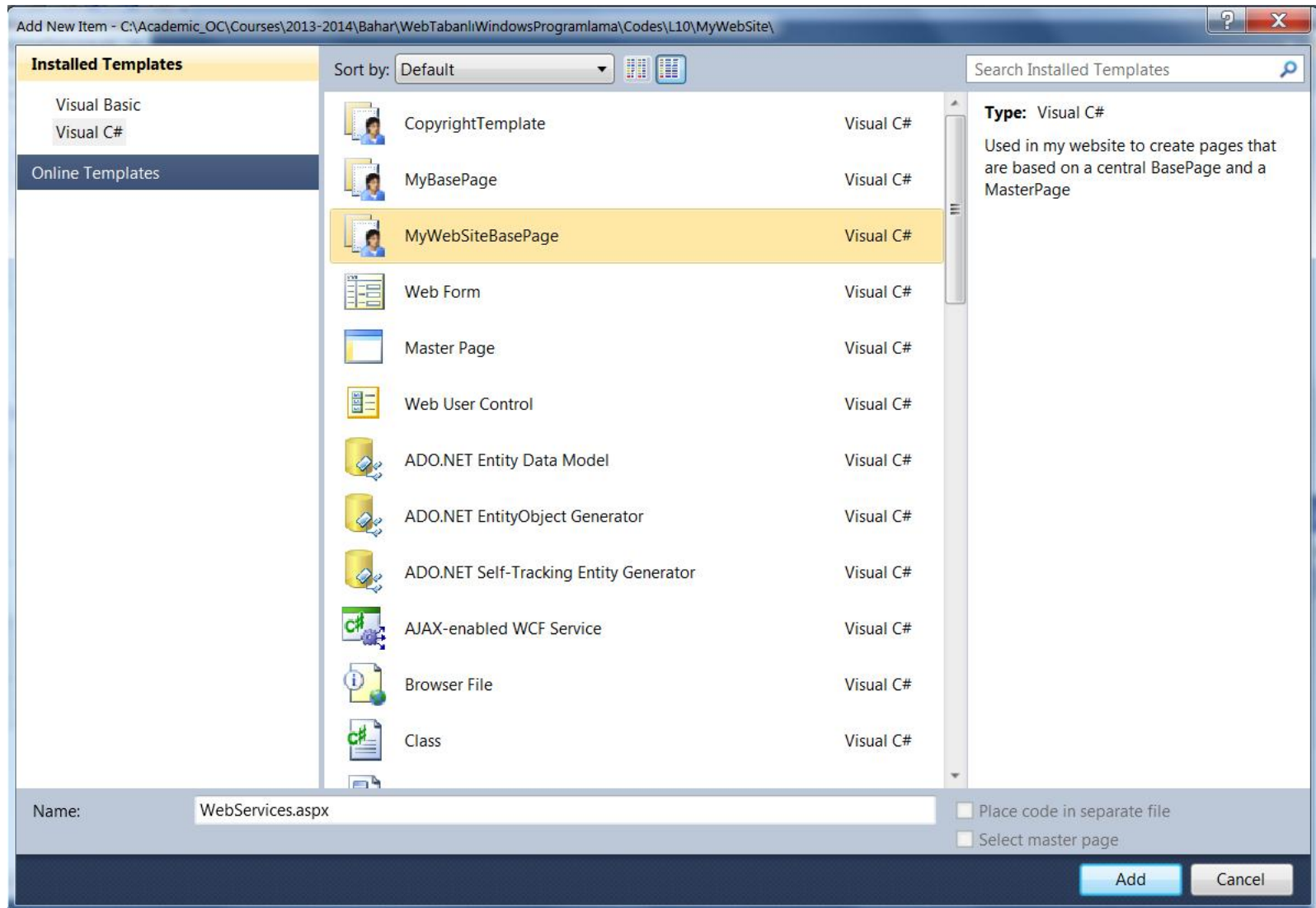
Calling Web Services from Client-Side

- Add New Item to the project



Example

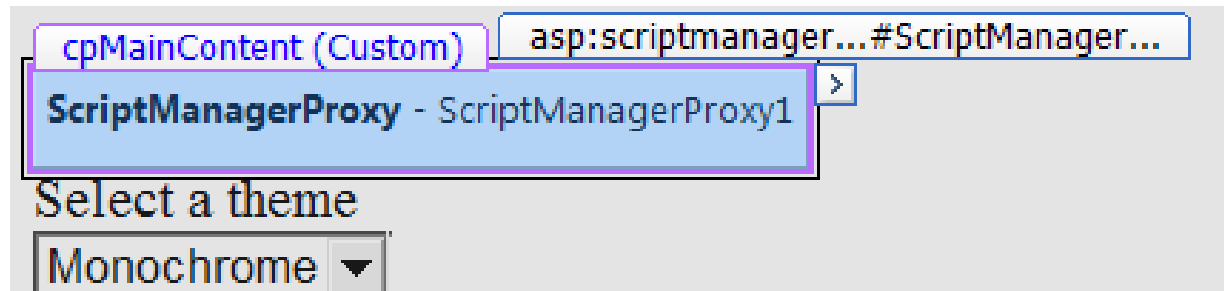
Calling Web Services from Client-Side



Example

Calling Web Services from Client-Side

- **Title = Web Services**
- In Design View → Drag a **ScriptManagerProxy**

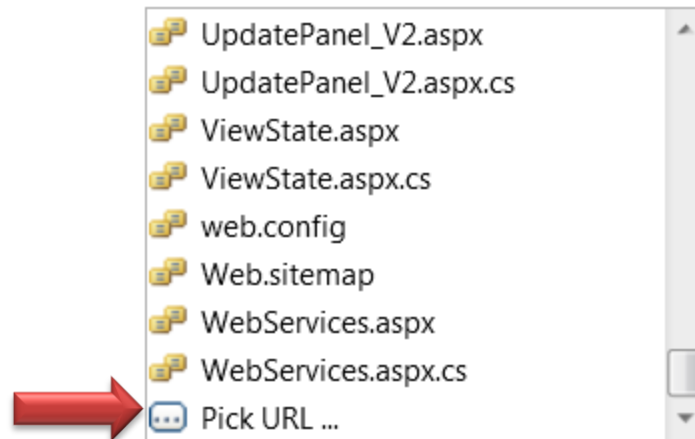


Example

Calling Web Services from Client-Side

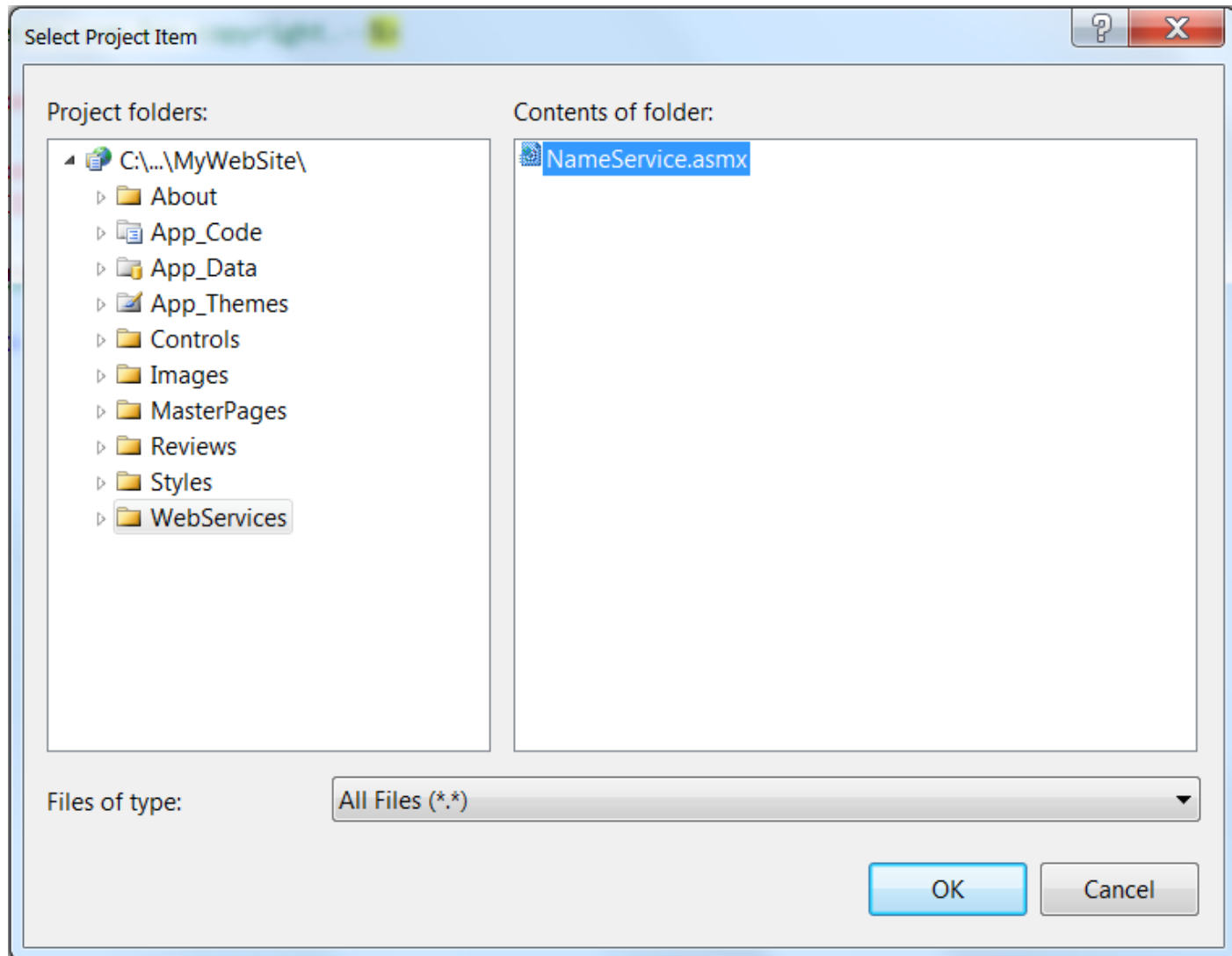
- In Source View:
 - Add **<Services>** element that contains **ServiceReference** with **Path**

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



Example

Calling Web Services from Client-Side



Example

Calling Web Services from Client-Side

- You should end up with this code:

```
<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>
```

Example

Calling Web Services from Client-Side

- After `</ScriptManagerProxy>`, drag:
 - Input (Text)
 - Input (Button)

from HTML category of Toolbox.

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

Example

Calling Web Services from Client-Side

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

NOTE → JavaScript is case-sensitive.

Example

Calling Web Services from Client-Side

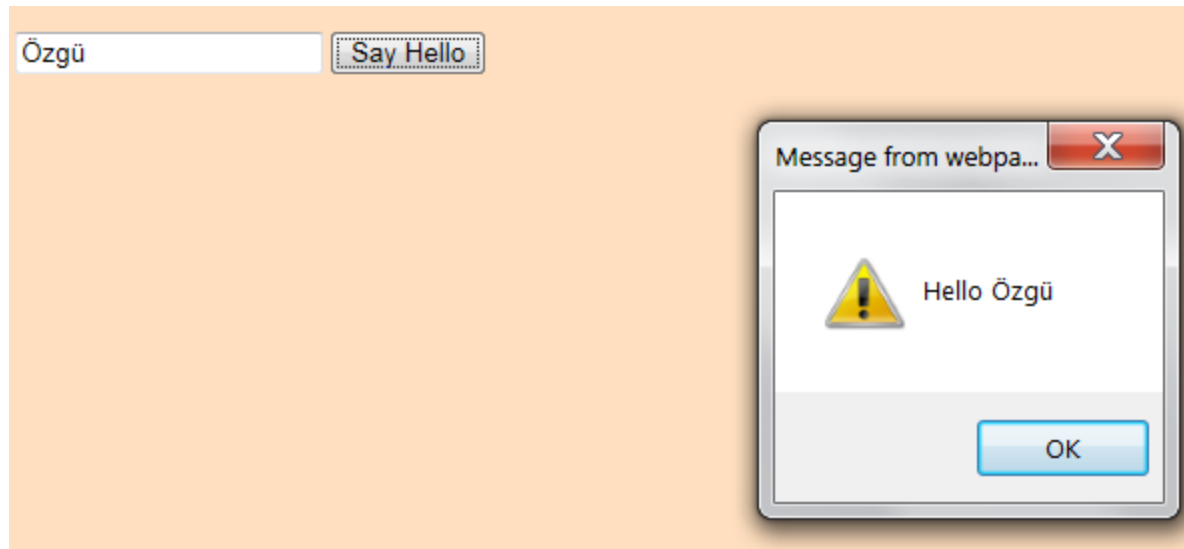
- View `WebServices.aspx` in browser.



A screenshot of a web service interface. It features a light orange background. On the left, there is a white text input field. To its right is a grey button with the text "Say Hello".



A screenshot of the same web service interface. The text input field now contains the name "Özgü". The "Say Hello" button remains to its right.



Page Methods

- Similar to web services.
- What's **different** is that page methods are defined **directly** in an *existing* ASPX page **instead of** a *separate* ASMX service file.
- You can only call them from script running within that page.
- Ideal for **small, simple functionality that is limited in scope to the current page.**

Page Methods

- To enable page methods you need to set the property **EnablePageMethods** of the **ScriptManager** control to **True**.

Page Methods

- Setting them up and using them is a twostep process:
 1. Create a **public** and **static** method in the Code Behind of the page you're working with. You need to apply the `[WebMethod]` attribute to this method. The method can optionally receive data through its parameters and optionally return some data.
 2. Write the necessary JavaScript to call the page method and work with its result.

Example

Calling Page Methods

- Open → `Frontend.master`
- `ScriptManager` Properties:

– `EnablePageMethods = True`

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

Example

Calling Page Methods

- Open → **WebServices.aspx.cs**
- Add the following code:

```
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);
    }
}
```

Example

Calling Page Methods

- Add → **using System.Web.Services**



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

Example

Calling Page Methods

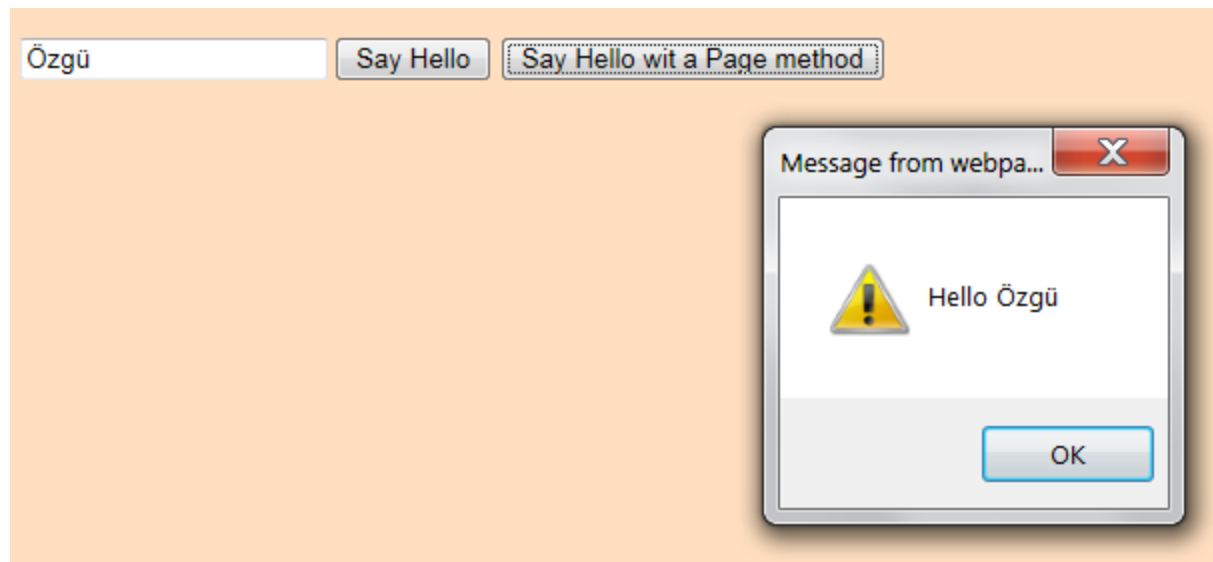
- In Source View:
 - Add **Input (Button)** from HTML category of Toolbox

```
<input id="yourNameText" type="text" />
<input id="sayHelloButton" type="button" value="Say Hello" />
➡ <input id="sayHelloPageMethod" type="button" value="Say Hello with a Page Method" />
<script type="text/javascript">
    function HelloWorld() {
        var yourName = $get('yourNameText').value;
        NameService.HelloWorld(yourName, HelloWorldCallback);
    }
    {
        function HelloWorldPageMethod() {
            var yourName = $get('yourNameText').value;
            PageMethods.HelloWorld(yourName, HelloWorldCallback);
        }
    }
    function HelloWorldCallback(result) {
        alert(result);
    }
    $addHandler($get('sayHelloButton'), 'click', HelloWorld);
    ➡ $addHandler($get('sayHelloPageMethod'), 'click', HelloWorldPageMethod);
</script>
```

Example

Calling Page Methods

- View in browser



ASP.NET AJAX

- For more information please check out MSDN's web site:

<http://tinyurl.com/AboutAjax>