Validation

Asst. Prof. Dr. Özgü Can

Validation

- Data usually flows in two directions:
  1. It either comes from the **server** and is sent to the **end user's browser**.
     - Files and databases

or

  2. The data is entered **by the user** and sent to the **server** to be processed or stored.
     - Page requests, contact forms, shopping cart scenarios

Validation

- To prevent your system from receiving **invalid data**, it's important to **validate this data** before you allow your system to work with it.
- Gathering data from the user:
  - **GET**: Data is appended to the actual address of the page being requested.
  - **POST**: Data is sent in the body of the request for the page.

GET

- Data is added to the requested address for a page.
- You can retrieve it using the **QueryString** property of the **Request** object.

http://www.MyWebSite.com/Reviews/ViewDetails.aspx?ReviewId=34&CategoryId=3

- **To access individual items in the query string**, you can use the **Get** method of the **QueryString** collection:

```
// Assigns the value 34 to the reviewId variable
int reviewId = Convert.ToInt32(Request.QueryString.Get("ReviewId"));
// Assigns the value 3 to the categoryId variable
int categoryId = Convert.ToInt32(Request.QueryString.Get("CategoryId"));
```

POST

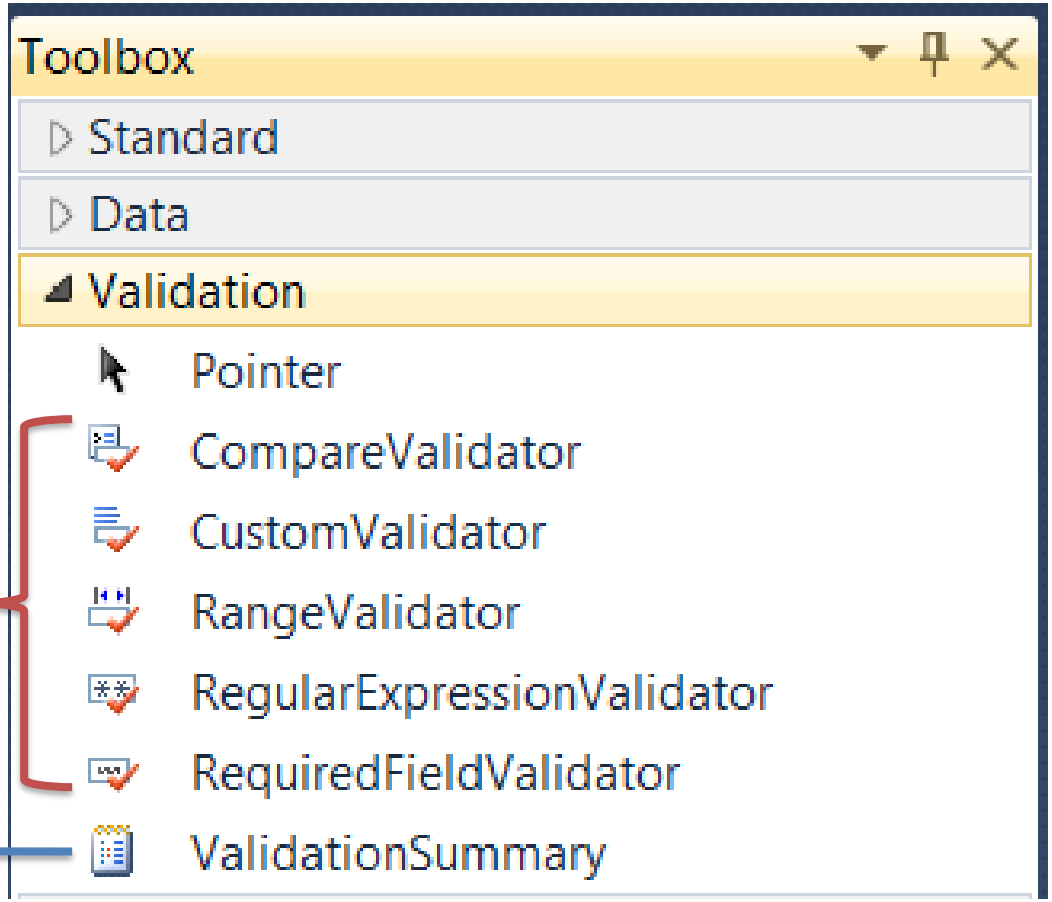
- Gets its data from a form with controls that have been submitted to the server.
- A **`TextBox`** called **`Age`** to hold the user's age and a **`Button`** to submit that age to the server.
- In the **`Button`** control's **`Click`** event:

```
int age = Convert.ToInt32(Age.Text);          // age now holds the user's age
```

- What happens when a user submits invalid data, either deliberately or by accident?
  - **Code will crash & throw an exception**
  - **Validate** all the data that is being sent to server.

ASP.NET Validation Controls

Never trust user input!



Perform the actual validation

Provides feedback to the user about any errors made in the page

Validation Controls

- Check the input at the **client** and at the **server**.

- ***Never*** rely on **client-side validation** as the only solution to validation.

- **Server-side validation** is the ***only*** real means of validation.

PROPERTY	DESCRIPTION
`Display`	This property determines whether or not the hidden error message takes up space. With the `Display` set to `Static`, the error message takes up screen estate, even when it is hidden. This is similar to the CSS setting `visibility: hidden` you saw in earlier chapters. The `Dynamic` setting hides the error message using `display: none` until it needs to be displayed. With a setting of `None`, the error message is not visible at all. This is useful if you are using a `ValidationSummary`
`CssClass`	This property enables you to set the CSS `class` attribute that is applied to the error message text.
`ErrorMessage`	This property holds the error message used in the `ValidationSummary` control. When the `Text` property is empty, the `ErrorMessage` value is also used as the text that appears on the page.
`Text`	The `Text` property is used as the text that the validation control displays on the page. This could be an asterisk (*) to indicate an error, or text like "Enter your name."
`ControlToValidate`	This property contains the server ID of the control that needs to be validated.
`EnableClientScript`	This property determines whether the control provides validation at the client. The default is `True`.
`SetFocusOnError`	This property determines whether client-side script gives the focus to the first control that generated an error. This setting is `False` by default.
`ValidationGroup`	Validation controls can be grouped together, enabling you to perform validation against a selection of controls. All controls with the same `ValidationGroup` are checked at the same time, which means that controls that are not part of that group are not checked. Consider, for example, a login page with a Login button and fields for a user name and password. The same page may also contain a search box that enables you to search the site. With the `ValidationGroup`, you can have the Login button validate the user name and password boxes, whereas the Search button triggers validation for just the search box.
`IsValid`	You don't typically set this property at design time, but at runtime it provides information about whether the validation test has passed.

RangeValidator

- Enables you to check whether a value falls within a certain range.

PROPERTY	DESCRIPTION
MinimumValue	This property determines the lowest acceptable value. For example, when checking an integer number between 1 and 10, you set this property to 1.
MaximumValue	This property determines the highest acceptable value. For example, when checking an integer number between 1 and 10, you set this property to 10.
Type	This property determines the data type that the validation control checks. This value can be set to String, Integer, Double, Date, or Currency to check the respective data types.

```
<asp:RangeValidator ID="RangeValidator1" runat="server"
    ControlToValidate="Rate" ErrorMessage="Enter a number between 1 and 10"
    MaximumValue="10" MinimumValue="1" Type="Integer" />
```

RegularExpressionValidator

- Enables you to check a value against a regular expression that you set in the **ValidationExpression** property of the control.

```
<asp:RegularExpressionValidator ID="RegularExpressionValidator1" runat="server"
    ControlToValidate="Email" ErrorMessage="Enter a valid e-mail address"
    ValidationExpression="\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*" />
```

CompareValidator

- Enables you to compare **the value of one control to another value**.
  - Sign-up forms where users have to enter a password twice to make sure they type the same password both times.
- You can also compare against a **constant value**.

CompareValidator

PROPERTY	DESCRIPTION
`ControlToCompare`	This property contains the ID of the control that the validator compares against. When this property is set, `ValueToCompare` has no effect.
`Operator`	This property determines the type of compare operation. For example, when `Operator` is set to `Equal` both controls must contain the same value for the validator to be considered valid. Similarly, you have options like `NotEqual`, `GreaterThan`, and `GreaterThanEqual` to perform different validation operations.
`Type`	This property determines the data type that the validation control checks. This value can be set to `String`, `Integer`, `Double`, `Date`, or `Currency` to check the respective data types.
`ValueToCompare`	This property enables you to define a constant value to compare against. This is often used in agreements where you have to enter a word like Yes to indicate you agree to some condition. Simply set the `ValueToCompare` to the word `Yes` and the `ControlToValidate` to the control you want to validate and you're done. When this property is set, make sure that the `ControlToCompare` property is empty because that will otherwise take precedence.

```
<asp:CompareValidator ID="CompareValidator1" runat="server"
    ControlToCompare="ConfirmPassword" ControlToValidate="Password"
    ErrorMessage="Your passwords don't match" />
```

CustomValidator

- Enables you to write custom validation functions for both the **client** (in JavaScript) and the **server** (using VB.NET or C#).

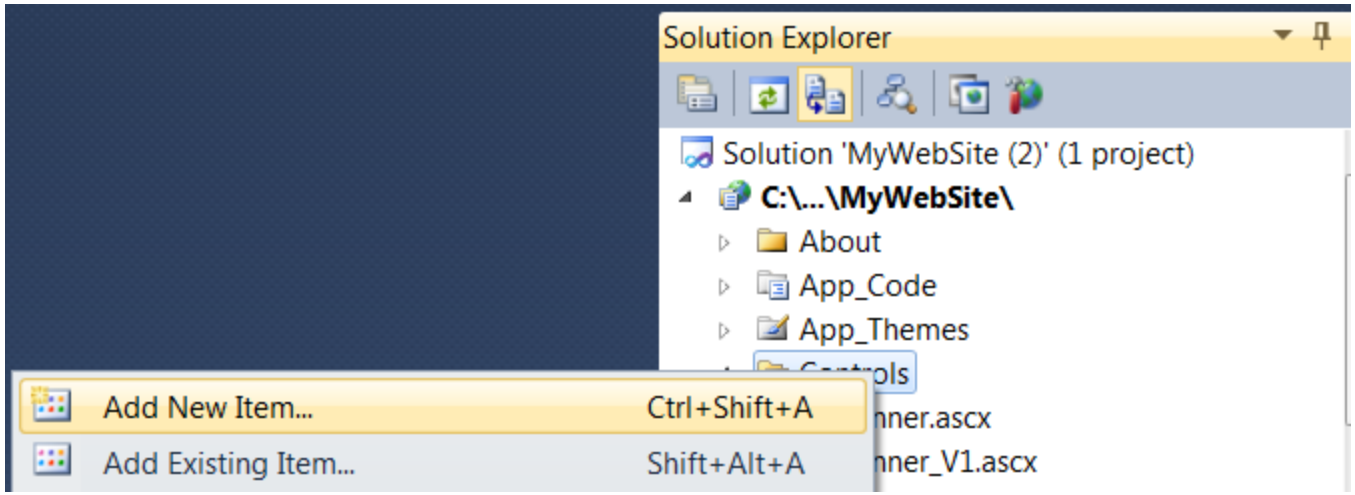
ValidationSummary

- Display errors.

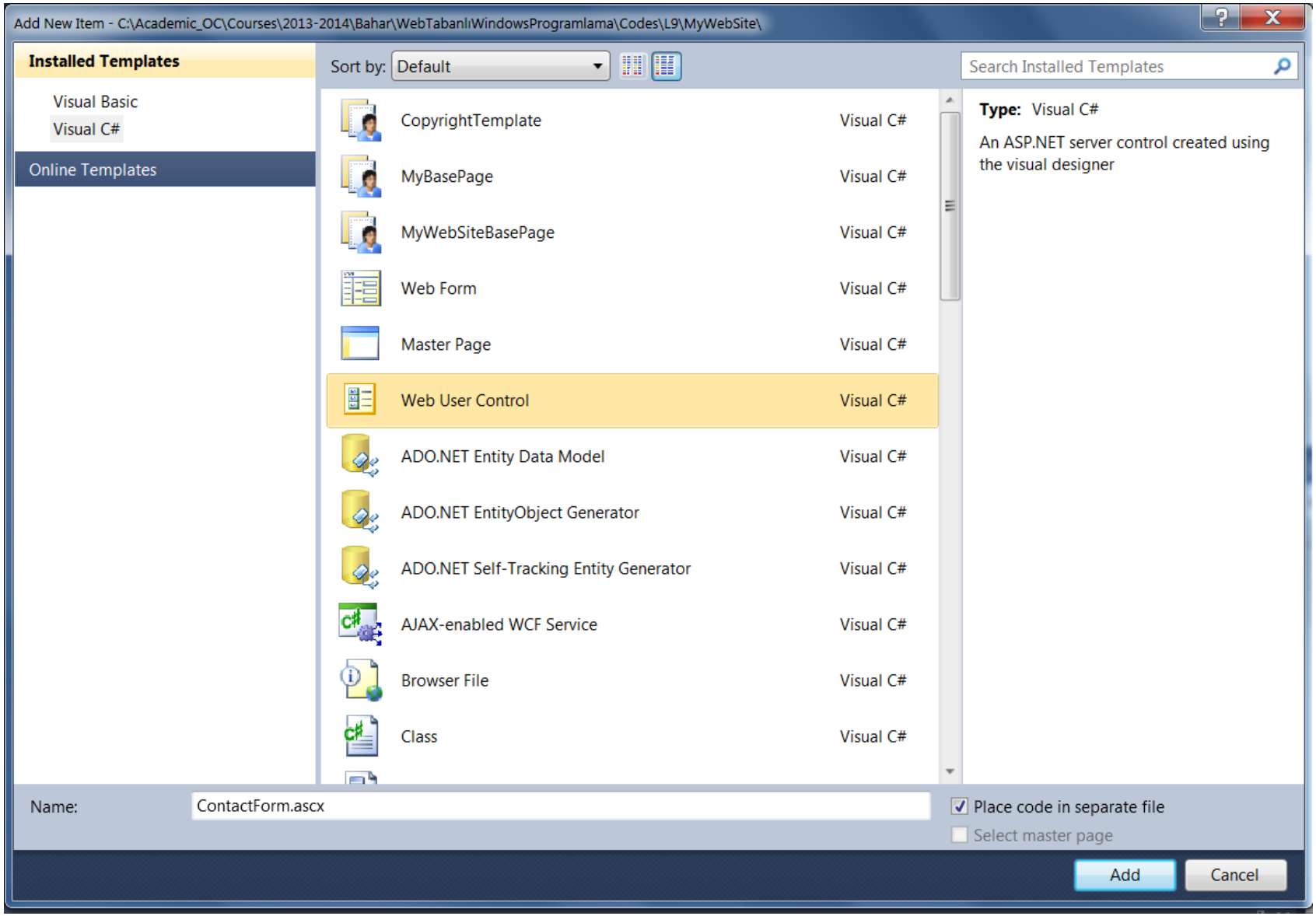
- Provides the user with a list of errors that it retrieves from the individual validation control's **`ErrorMessage`** properties.

Example

- In **`Controls`** folder → **`Add New Item`**

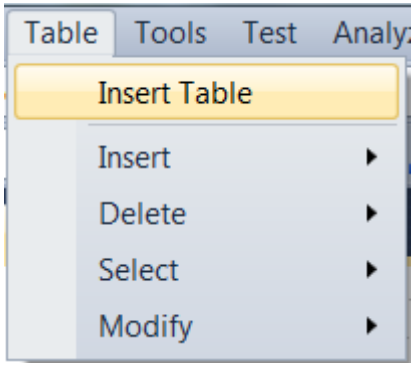


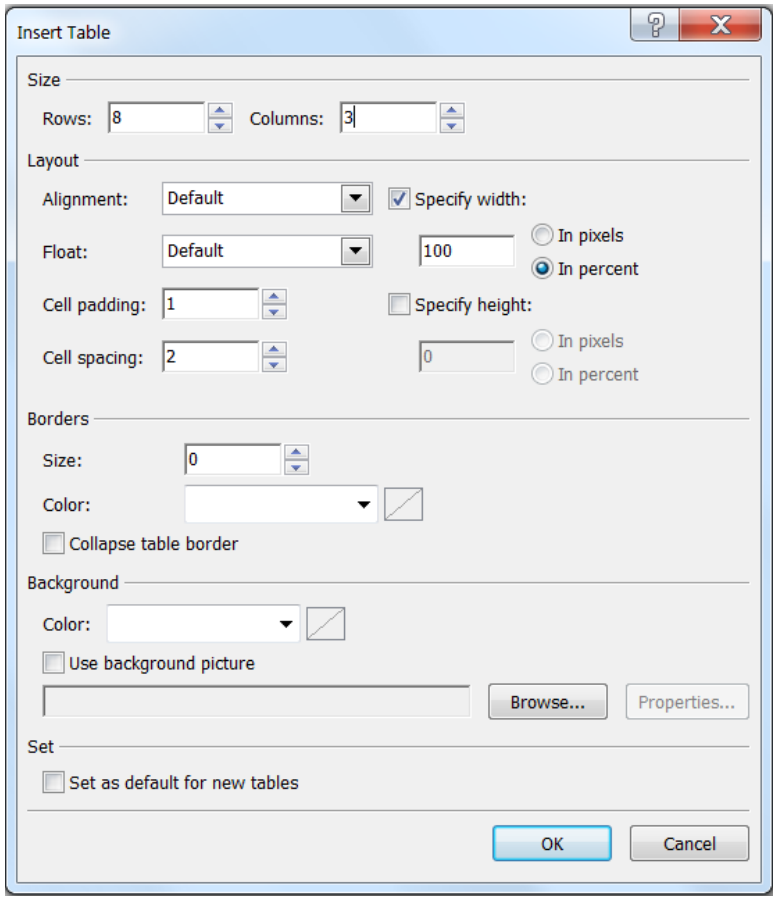
Example


Add New Item - C:\Academic_OC\Courses\2013-2014\Bahar\WebTabanliWindowsProgramlama\Codes\L9\MyWebSite\
Installed Templates
Visual Basic
Visual C#
Online Templates
Sort by: Default
Search Installed Templates
CopyrightTemplate Visual C#
MyBasePage Visual C#
MyWebSiteBasePage Visual C#
Web Form Visual C#
Master Page Visual C#
Web User Control Visual C#
ADO.NET Entity Data Model Visual C#
ADO.NET EntityObject Generator Visual C#
ADO.NET Self-Tracking Entity Generator Visual C#
AJAX-enabled WCF Service Visual C#
Browser File Visual C#
Class Visual C#
Type: Visual C#
An ASP.NET server control created using the visual designer
Name: ContactForm.ascx
Place code in separate file
Select master page
Add
Cancel

Example

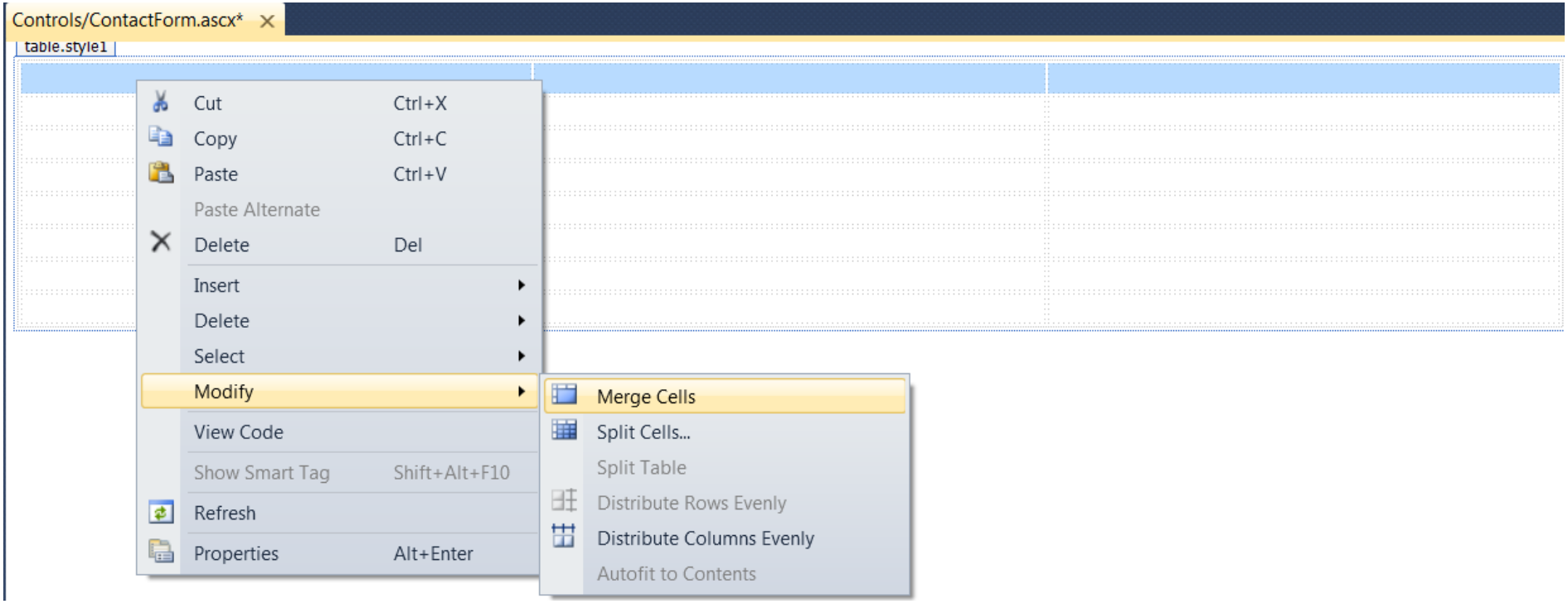
- In Design View → Insert a table





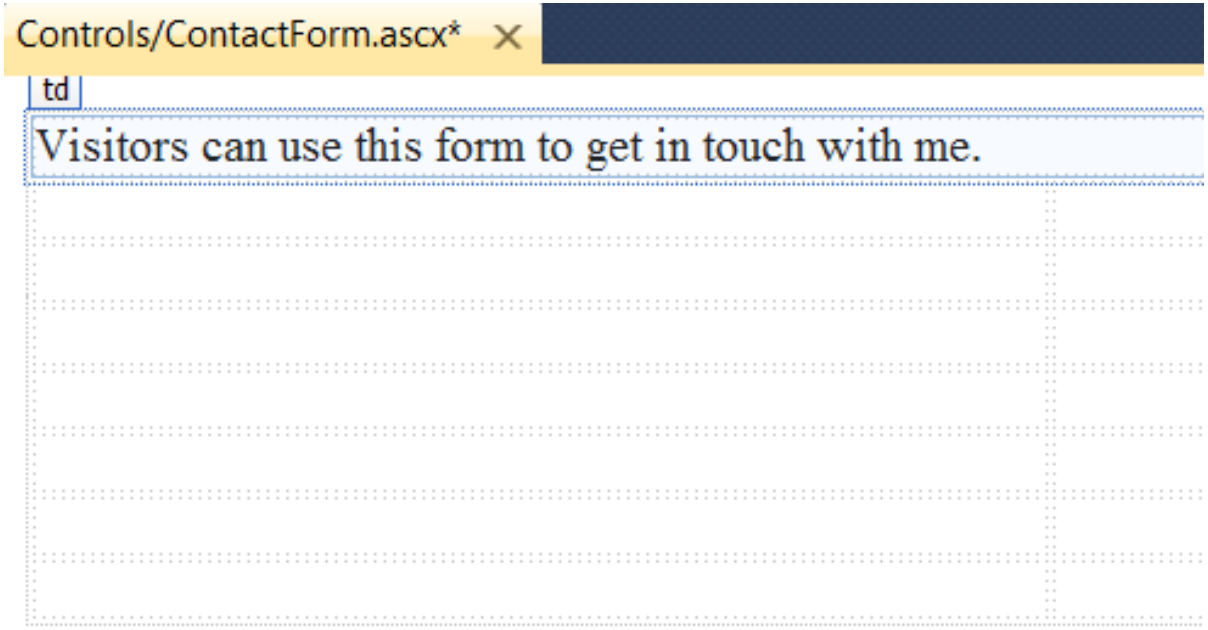
Example

- Merge the 3 cells of the 1st row.



Example

- Type some text.



Example

- Type **`Name`** → 1st cell of the 2nd row
- Drag a **`TextBox`** → 2nd cell of the 2nd row
- Drag a **`RequiredFieldValidator`** → 3rd cell of the 2nd row
- Drag a **`Button`** → 2nd cell of the last row

Example

- TextBox → ID = nameTextBox
- Button:
  - ID = sendButton
  - Text = Send

Example

- Properties of **`RequiredFieldValidator`**:
  - **`CssClass = ErrorMessage`**
  - **`ErrorMessage = Enter your name`**
  - **`Text = * Enter your name`**
  - **`ControlToValidate = nameTextBox`**

Example

- Add the following declaration to **`Monochrome.css`** and **`DarkGrey.css`**.

```
.ErrorMessage
{
    color:Red;
}
```

Example

- Open **`Contact.aspx`** from the **`About`** folder.
- In Design View:
  - From the Solution Explorer, drag the user control **`ContactForm.ascx`** into the main content area of the page, identified by the purple border.

Home Reviews About Login

Home
Reviews
By Genre
All Reviews
About
Contact
About Me
Login

SiteMapDataSource - SiteMapDataSource1

Root Node > Parent Node > Current Node

cpMainContent (Custom)

Select a theme

Monochrome

Home Reviews About Login

Home
Reviews
By Genre
All Reviews
About
Contact
About Me
Login

SiteMapDataSource - SiteMapDataSource1

Root Node > Parent Node > Current Node

uc1:contactform#ContactForm1

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

Name * Enter your name

Send

Select a theme

Monochrome

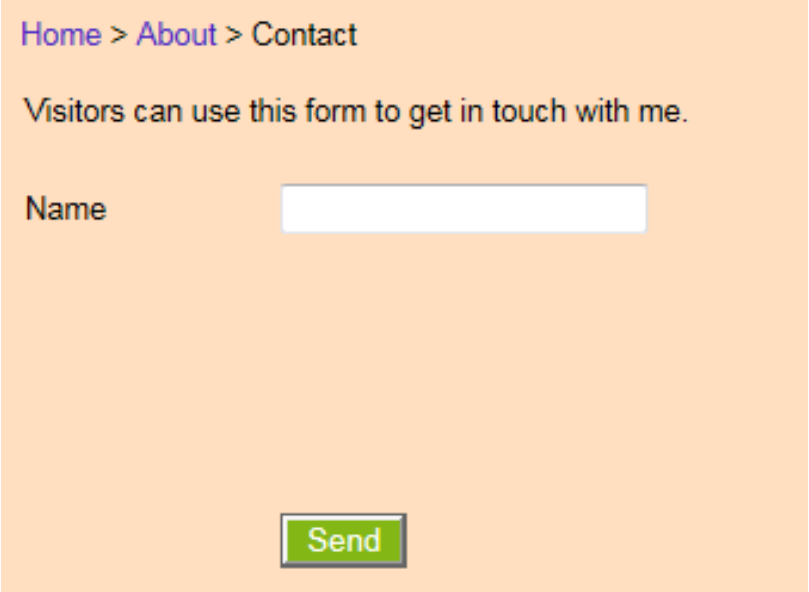
Example

- In Source View:

```
<asp:Content ID="Content2" ContentPlaceHolderID="cpMainContent" Runat="Server">
    <uc1:ContactForm ID="ContactForm1" runat="server" />
</asp:Content>
```

Example

- View in browser

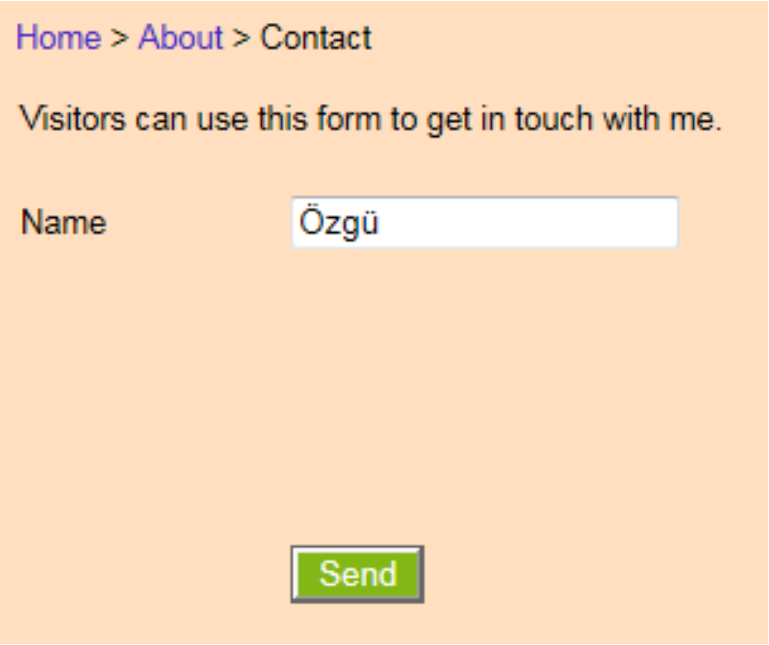


Example

- Leave the **`TextBox`** empty and click **`Send`**

Example

- Enter your name & click Send
  - Page posts back to the server



Example
Extending the Contact Form

- Open **`ContactForm.ascx`** in **`Controls`** folder.

- In Design View → In the 2nd column, drag 5 **`TextBox`** in the empty table cells between the text box for the **`Name`** and the **`Send`** button.

Example
Extending the Contact Form

- Name the **`IDs`**:
  - **`emailAddressTextBox`**
  - **`confirmEmailAddressTextBox`**
  - **`phoneHomeTextBox`**
  - **`phoneBusinessTextBox`**
  - **`commentsTextBox`**
    - **`TextMode = MultiLine`**

Example
Extending the Contact Form

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

Name		* Enter your name
	Send	

Example
Extending the Contact Form

- Add the text:

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

Name		* Enter your name
Email Address		
Email Address (Repeat)		
Home Phone		
Business Phone		
Comments		
	Send	

Example
Extending the Contact Form

- In the last cell of the row for the first e-mail address, drag a **`RequiredFieldValidator`** and a **`RegularExpressionValidator`**.
- In the last cell of the row for the second e-mail address, drag a **`RequiredFieldValidator`** and a **`CompareValidator`**.
- In the last cell for the comments row, drag a **`RequiredFieldValidator`**.

Example
Extending the Contact Form

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

Name		* Enter your name
Email Address		RequiredFieldValidatorRegularExpressionValidator
Email Address (Repeat)		RequiredFieldValidator CompareValidator
Home Phone		
Business Phone		
Comments		RequiredFieldValidator
	Send	

Example
Extending the Contact Form

- For each of the 5 validation controls set the properties:
  - **`Text = *`**
  - **`Display = Dynamic`**
  - **`CssClass = ErrorMessage`**

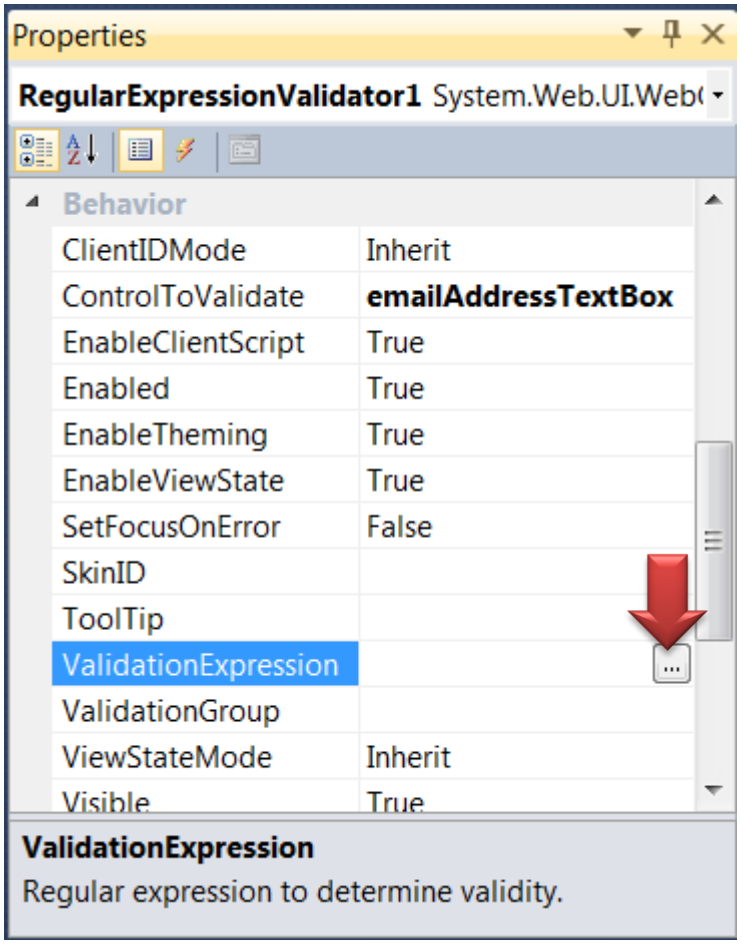
Example
Extending the Contact Form

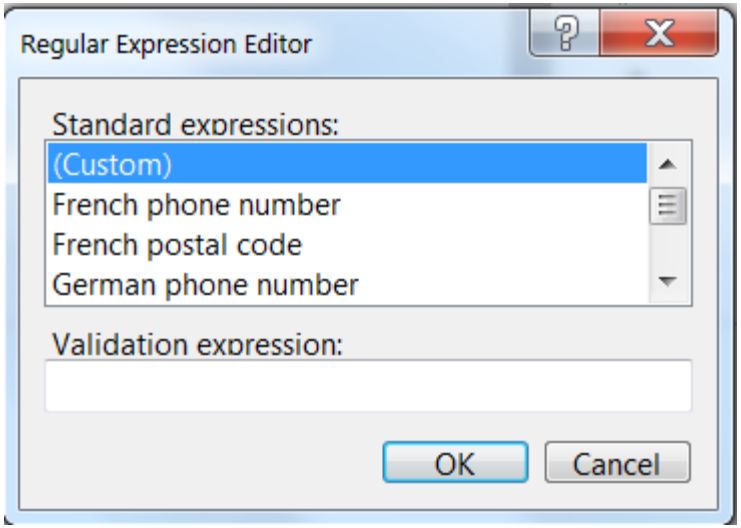
- Set the remaining properties for the controls as shown:

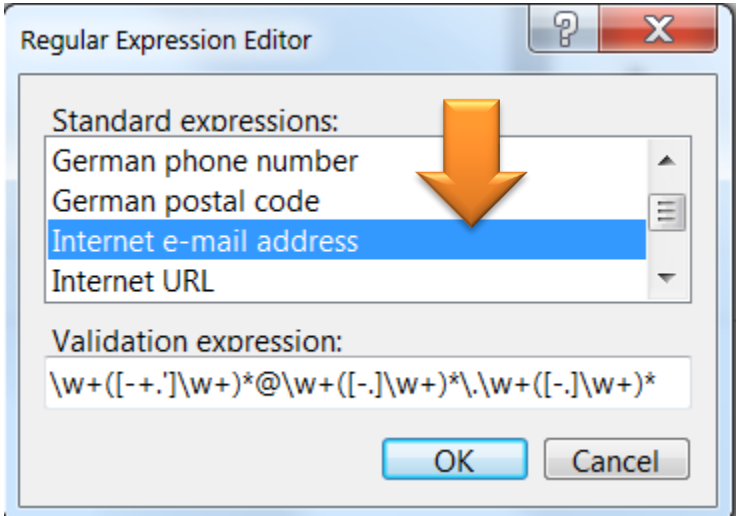
CONTROL	PROPERTIES YOU NEED TO SET	
RequiredFieldValidator (for the first e-mail address)	ErrorMessage:	Enter an e-mail address
	ControlToValidate:	emailAddressTextBox
RegularExpressionValidator	ErrorMessage:	Enter a valid e-mail address
	ControlToValidate:	emailAddressTextBox
RequiredFieldValidator (for the second e-mail address)	ErrorMessage:	Confirm the e-mail address
	ControlToValidate:	confirmEmailAddressTextBox
CompareValidator	ErrorMessage:	Retype the e-mail address
	ControlToCompare:	emailAddressTextBox
	ControlToValidate:	confirmEmailAddressTextBox
RequiredFieldValidator (for the Comments field)	ErrorMessage:	Enter a comment
	ControlToValidate:	commentsTextBox

Example
Extending the Contact Form

- Click **`RegularExpressionValidator`**

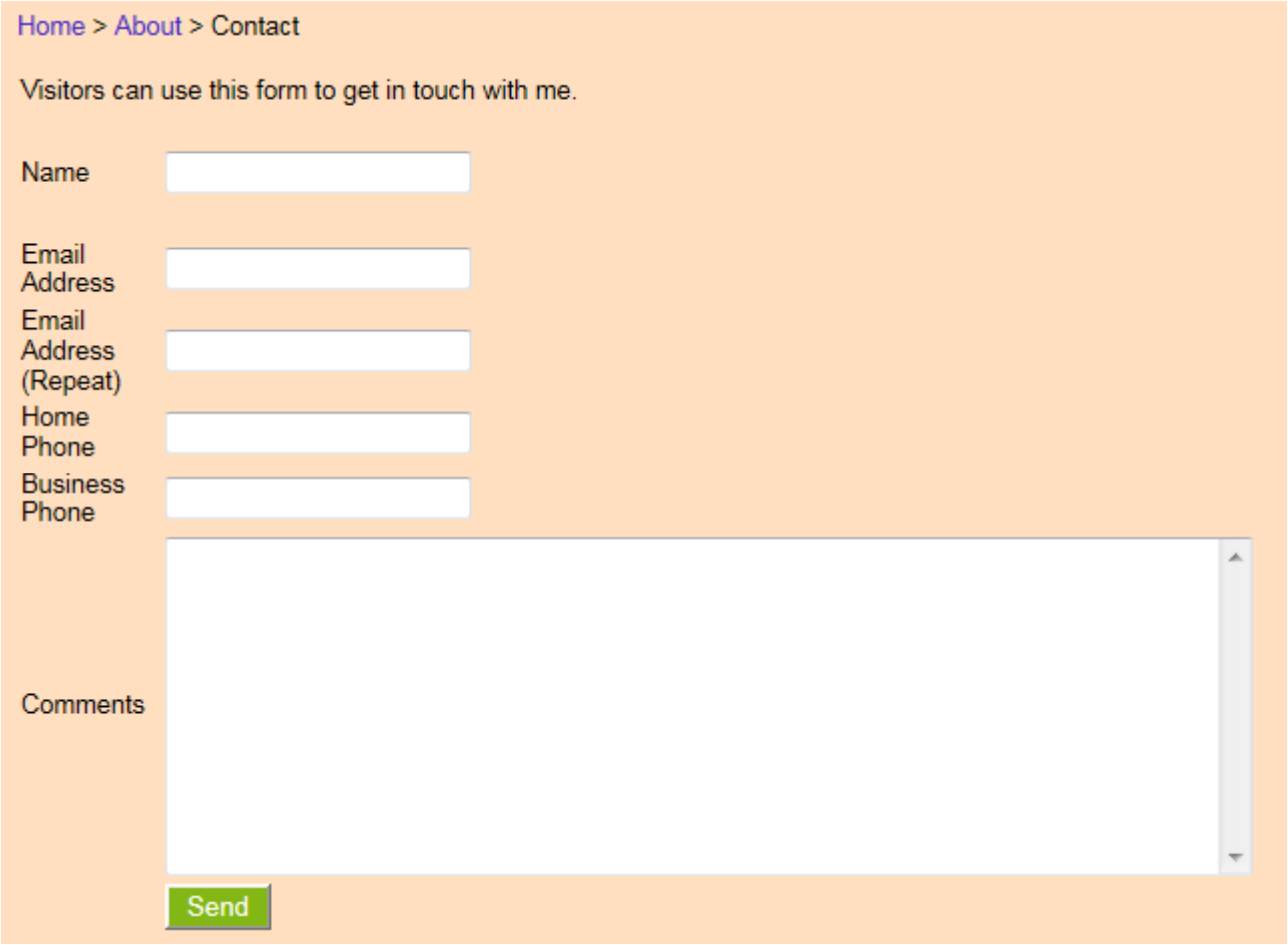






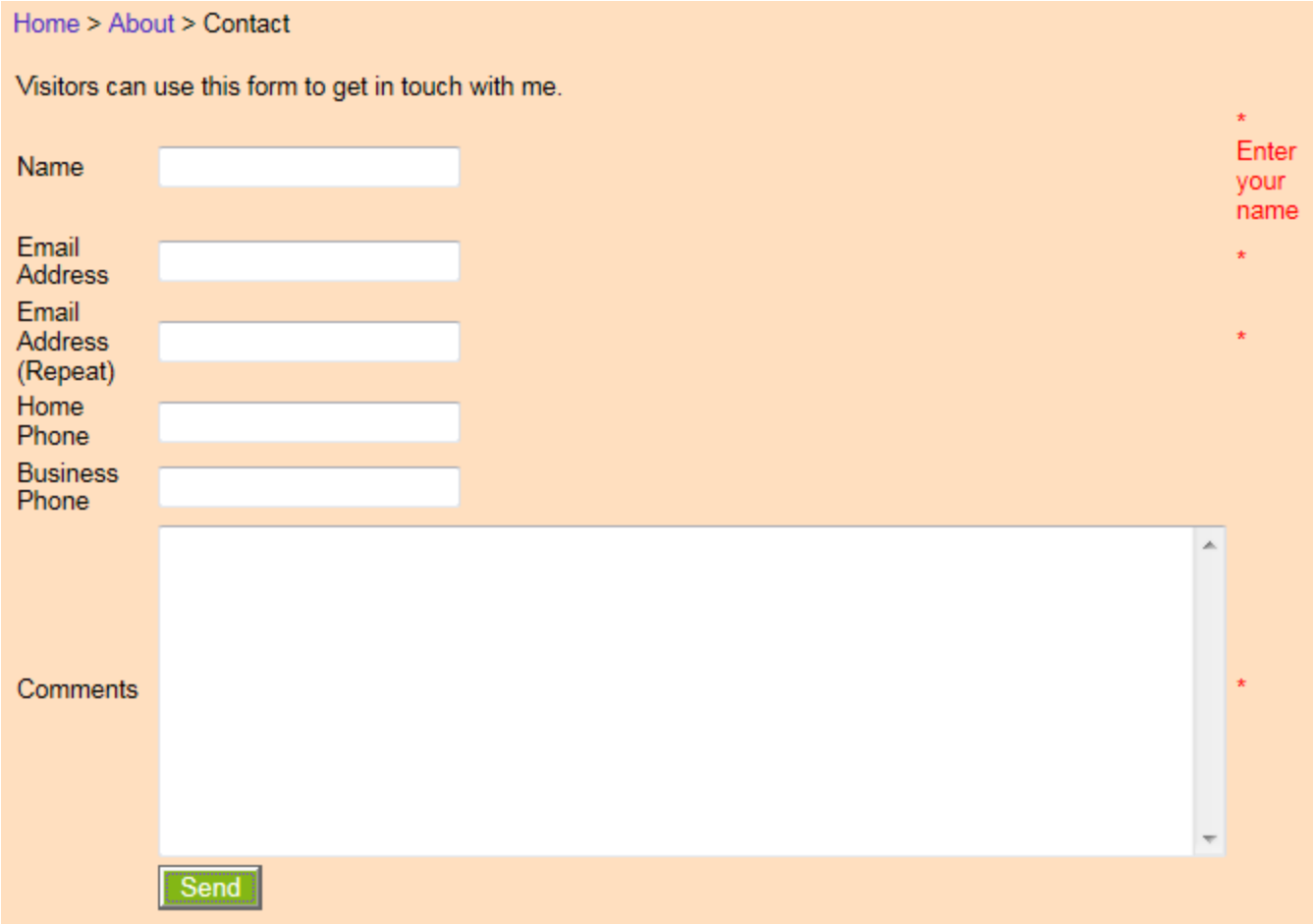
Example
Extending the Contact Form

- View **`Contact.aspx`** in browser



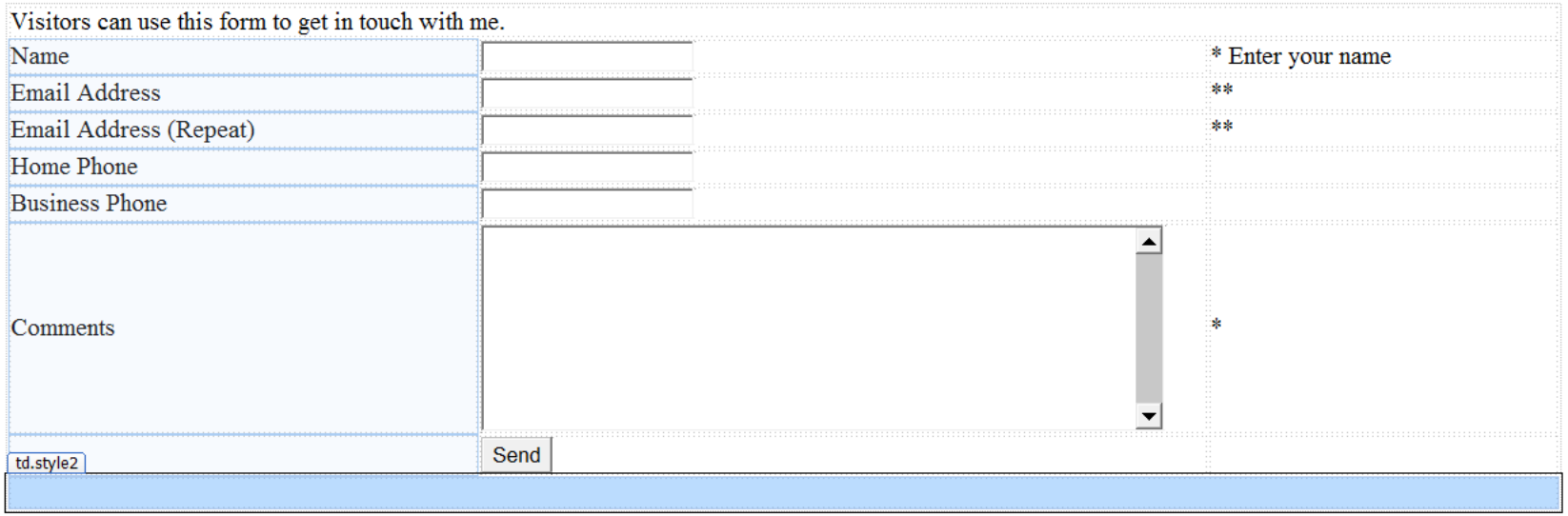
Example
Extending the Contact Form

- Leave textboxes empty and click **`Send`**



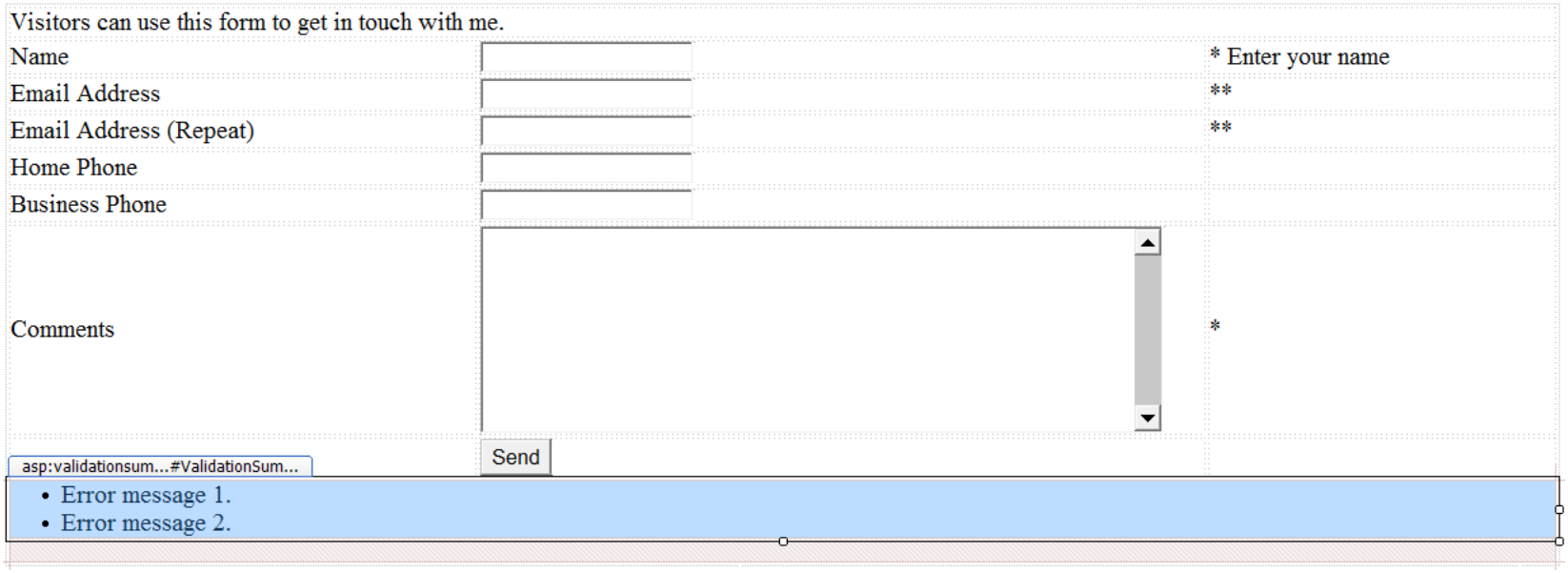
Example
Validation Methods

- Open → **`ContactForm.ascx`**
- In Design View → Insert a row below the last row.
  - **Modify** > **Merge Cells**



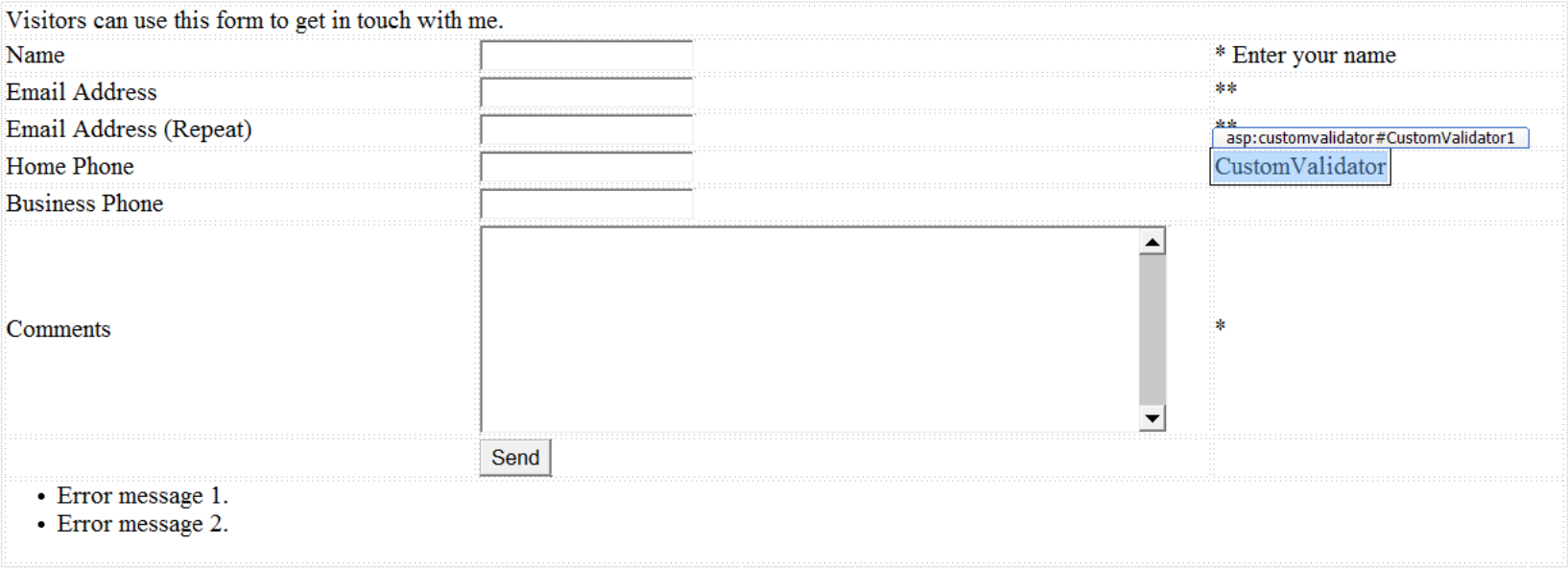
Example
Validation Methods

- Drag a **`ValidationSummary`** control into this new cell
  - **`CssClass = ErrorMessage`**



Example
Validation Methods

- In the empty cell after the text box for the **`Home Phone`**, drag a **`CustomValidator`** control.



Example
Validation Methods

- Set the following properties:
  - **CssClass = ErrorMessage**
  - **Display = Dynamic**
  - **ErrorMessage = Enter your home or business number**
  - **Text = ***
  - **ClientValidationFunction = ValidatePhoneNumbers**

Example
Validation Methods

- Double-click **`CustomValidator`**

```
protected void CustomValidator1_ServerValidate(object source, ServerValidateEventArgs args)
{
    if (!string.IsNullOrEmpty(phoneHomeTextBox.Text) || !string.IsNullOrEmpty(phoneBusinessTextBox.Text))
    {
        args.IsValid = true;
    }
    else
    {
        args.IsValid = false;
    }
}
```

Server-side Validation

Example
Validation Methods

- In Source View:

```
<script type="text/javascript">
    function ValidatePhoneNumbers(source, args)
    {
        var phoneHome = document.getElementById('<%= phoneHomeTextBox.ClientID %>');
        var phoneBusiness = document.getElementById('<%= phoneBusinessTextBox.ClientID %>');
        if (phoneHome.value != '' || phoneBusiness.value != '')
        {
            args.IsValid = true;
        }
        else
        {
            args.IsValid = false;
        }
    }
</script>
```

Client-side Validation

```
<table class="style1">
    <tr>
        <td colspan="3">
            Visitors can use this form to get in touch with me.</td>
```

Example
Validation Methods

View
`Contact.aspx`
in
browser.

Home > About > Contact

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

Name Özgü

Email Address ochotmail.com *

Email Address (Repeat) *

Home Phone *

Business Phone

Comments *

Send

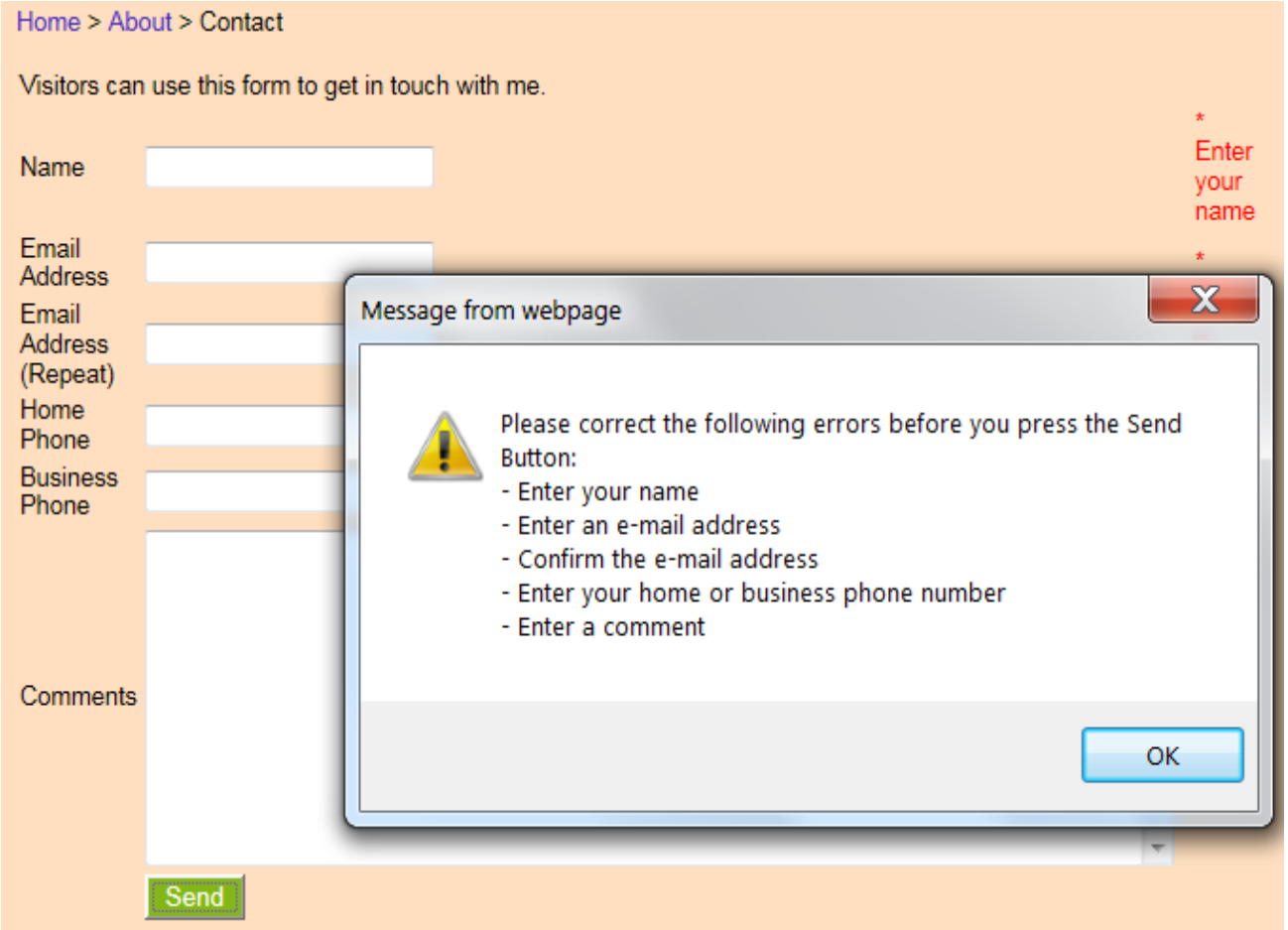
- Enter a valid e-mail address
- Confirm the e-mail address
- Enter your home or business phone number
- Enter a comment

Example
Validation Methods

- In Design View:
  - **`ValidationSummary`** control's Properties Grid
    - **`ShowMessageBox = True`**
    - **`ShowSummary = False`**
    - **`HeaderText = Please correct the following errors before you press the Send Button:`**

Example
Validation Methods

View
`Contact.aspx`
in
browser.



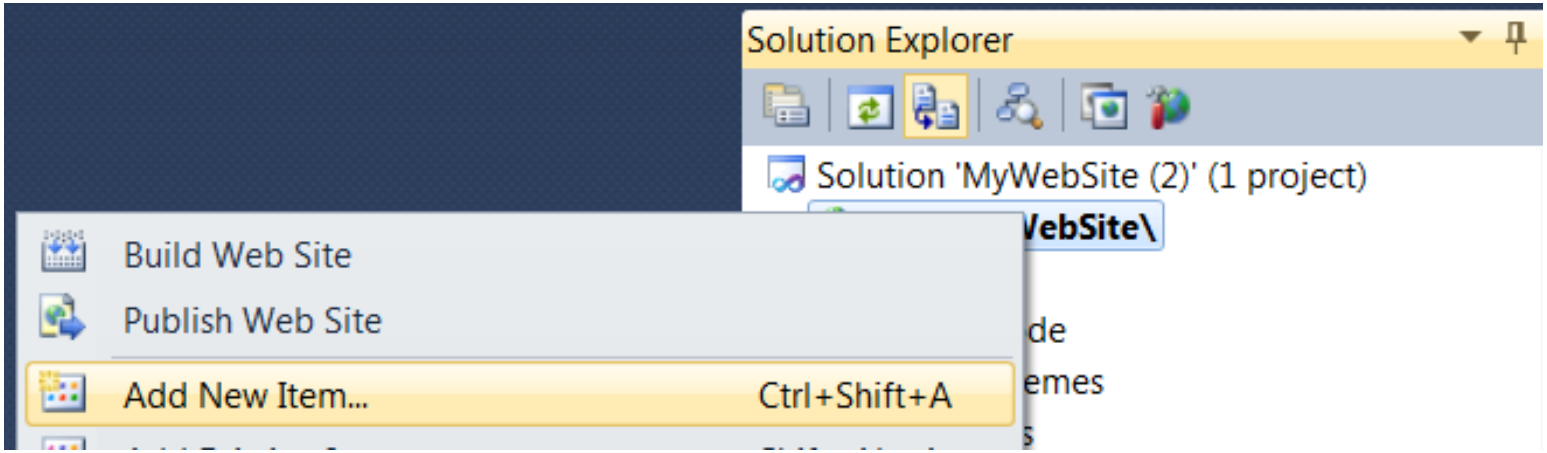
Sending E-Mail

- Sending e-mail from .NET application:

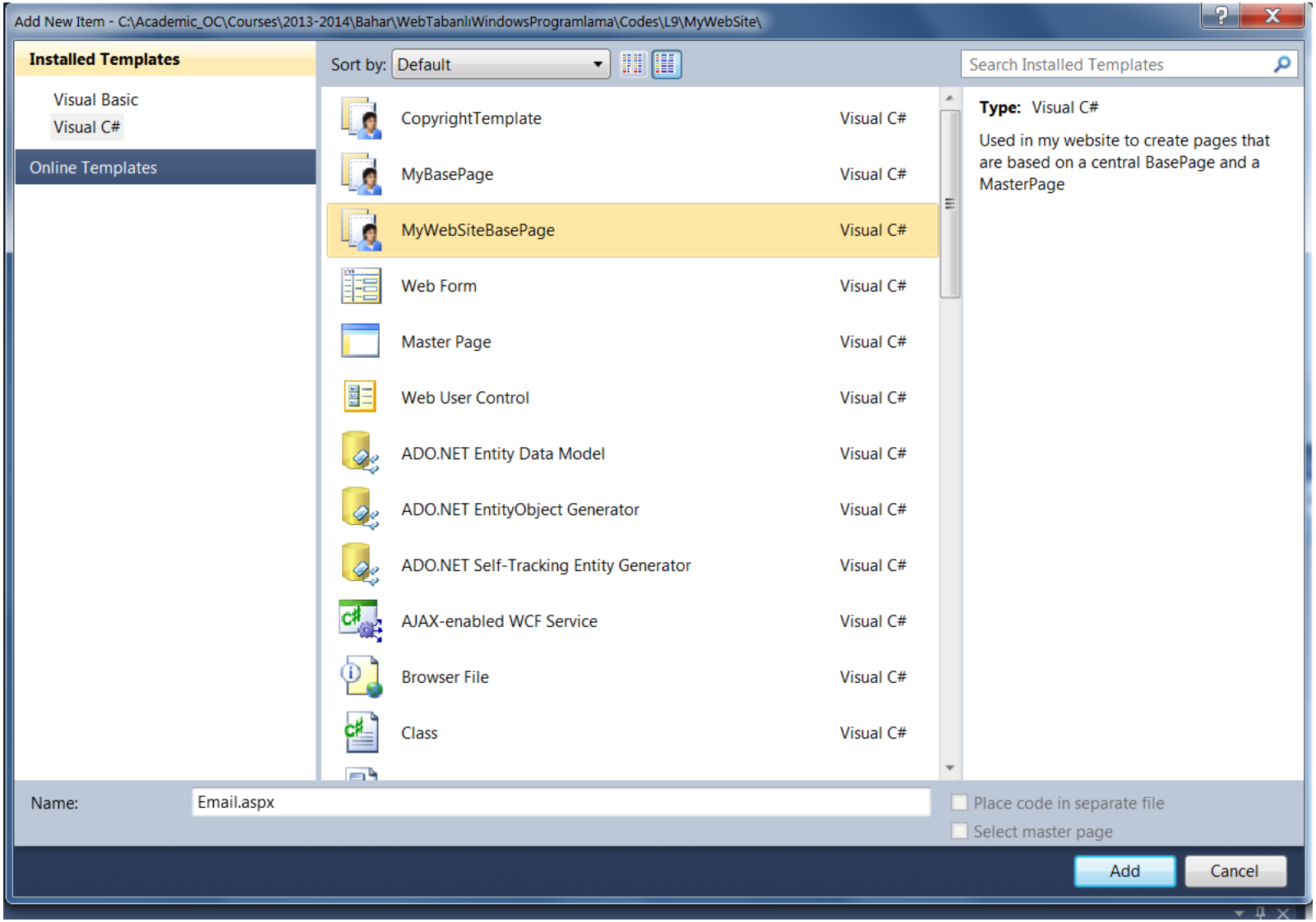
CLASS	DESCRIPTION
`MailMessage`	This class represents the message you're going to send. It has properties such as `Subject` and `Body` to set the message contents; `To`, `CC`, and `Bcc` properties to set the addressees; and an `Attachments` collection to attach files to the message.
`MailAddress`	This class represents a sender or receiver address used in the e-mail. It has a few constructor overloads that enable you to set the e-mail address and display name.
`Attachment`	This class represents files you can attach to a `MailMessage`. When you construct an `Attachment` instance, you can pass in the name of the file you want to send. You then add the attachment to the `MailMessage` using the `Add` method of its `Attachments` collection.
`SmtpClient`	This class is used to send the actual message. By default, an instance of this class checks the `web.config` file for settings such as the SMTP server (which stands for Simple Mail Transfer Protocol) to send the mail to and an optional user name and password that is used for sending e-mail.

Example
Sending E-Mail

- Add New Item



Example
Sending E-Mail



Example
Sending E-Mail

- Change the **`Title`** → **`Email Demo`**
- Open Code Behind file
  - Add:

```
using System.Net.Mail;
```

Example
Sending E-Mail

- Add the following code to **`Page_Load`** handler:

```
protected void Page_Load(object sender, EventArgs e)
{
    MailMessage myMessage = new MailMessage();
    myMessage.Subject = "Test Message";
    myMessage.Body = "Hello world! This is my web site.";
    myMessage.From = new MailAddress("wbwpcourse@gmail.com", "Sender_WBWP");
    myMessage.To.Add(new MailAddress("wbwpcourse@gmail.com", "Receiver_Ozgu Can"));
    myMessage.CC.Add(new MailAddress("ozgucan@gmail.com", "Receiver_Ozgu Can"));

    SmtpClient mySmtpClient = new SmtpClient();
    mySmtpClient.Host = "smtp.gmail.com";
    mySmtpClient.EnableSsl = true;
    mySmtpClient.Port = 587;
    mySmtpClient.Send(myMessage);
}
```

Example
Sending E-Mail

- Open **`web.config`**
- Right before the closing **`</configuration>`** tag, add the following settings:

```
  <system.net>
    <mailSettings>
      <smtp deliveryMethod="Network" from="WBWP &lt; wbwpcourse@gmail.com &gt;">
        <network enableSsl="true" userName="wbwpcourse@gmail.com" password="blablabla" host="smtp.google.com" />
      </smtp>
    </mailSettings>
  </system.net>

</configuration>
```

Example
Sending E-Mail

- View **`Email.aspx`** in browser
- Check your e-mail box!

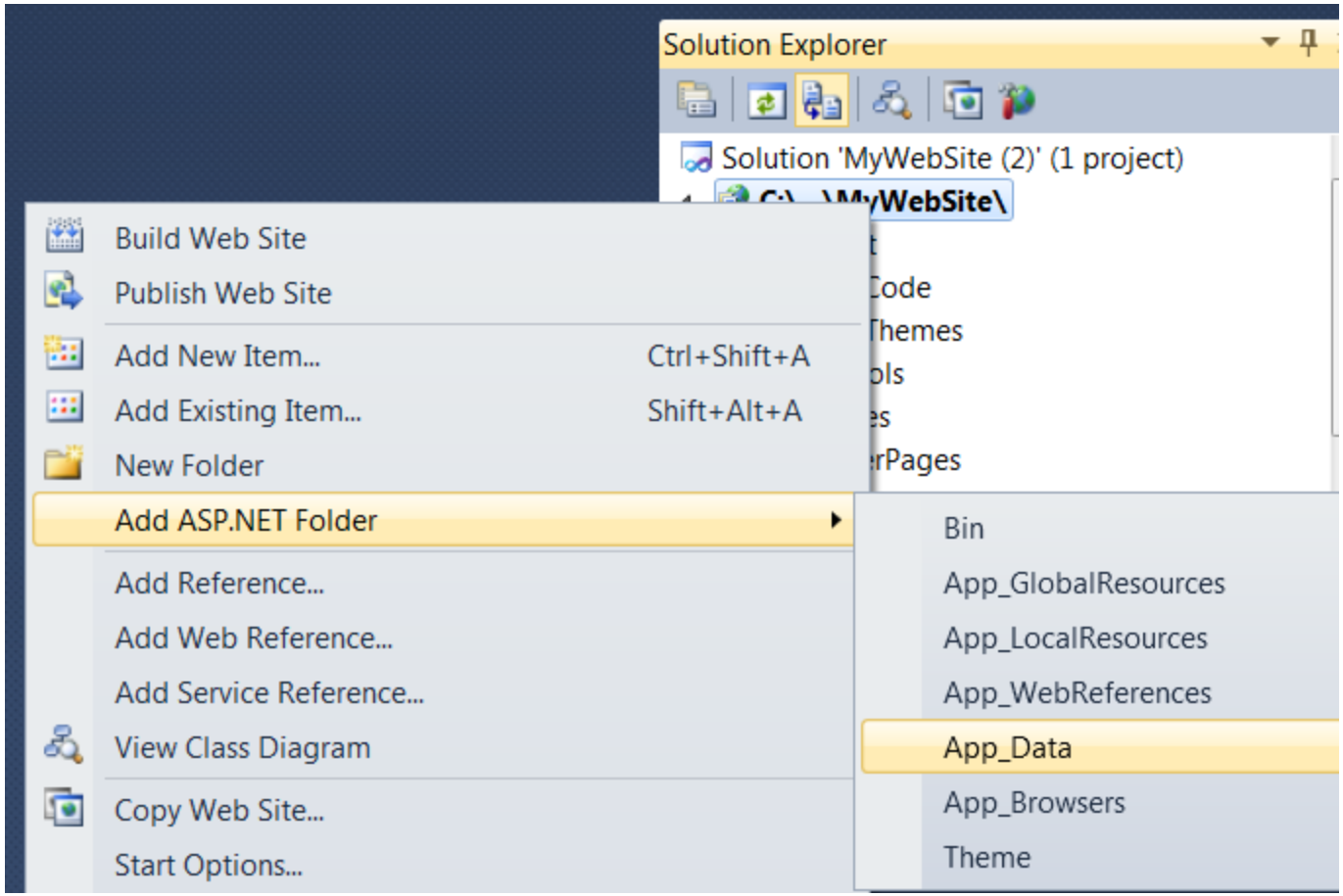
Reading from Text Files

- Common methods of the **`File`** class

METHOD	VALUE
`AppendAllText`	Appends a specified string to a text file. If the file does not exist, it's created first.
`Copy`	Copies a file from one location to another.
`Delete`	Deletes the specified file from disk.
`Exists`	Checks if the specified file exists on disk.
`Move`	Moves the specified file to a different location.
`ReadAllText`	Reads the contents of a text file.
`WriteAllText`	Writes the contents of a string to a new file and overwrites the target file if it already exists.

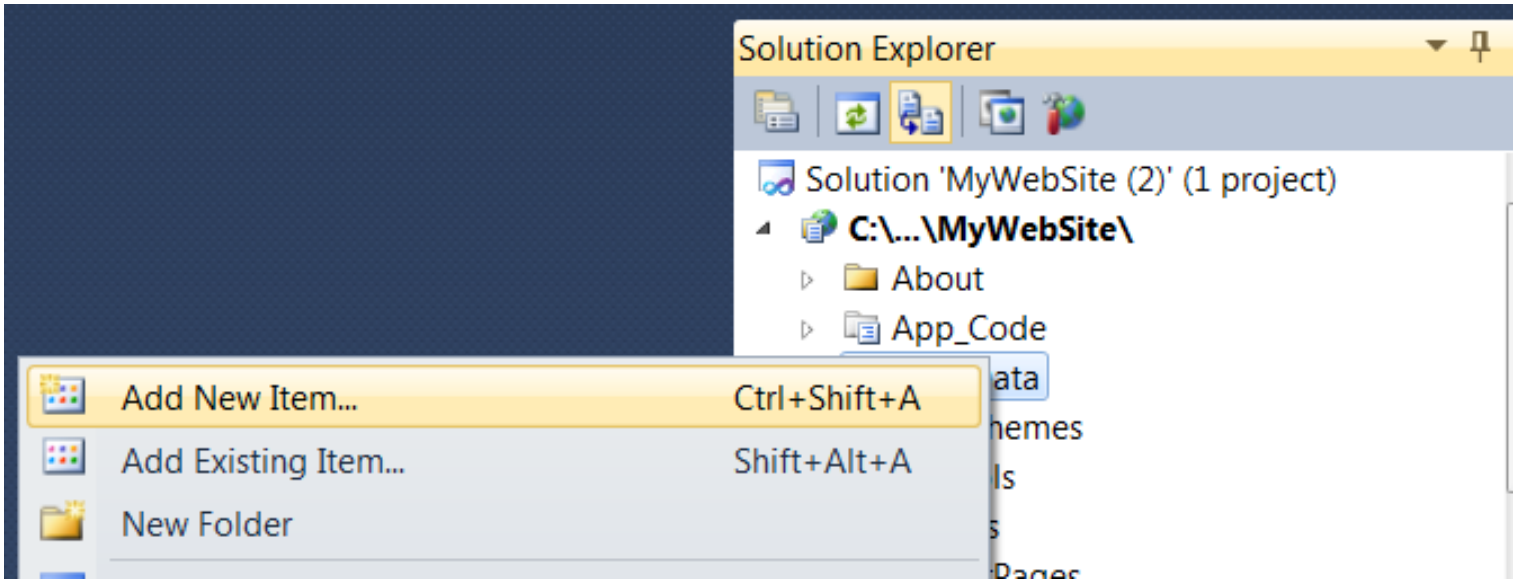
Example
Sending E-Mail from the ContactForm

- Add new folder → **`App_Data`**

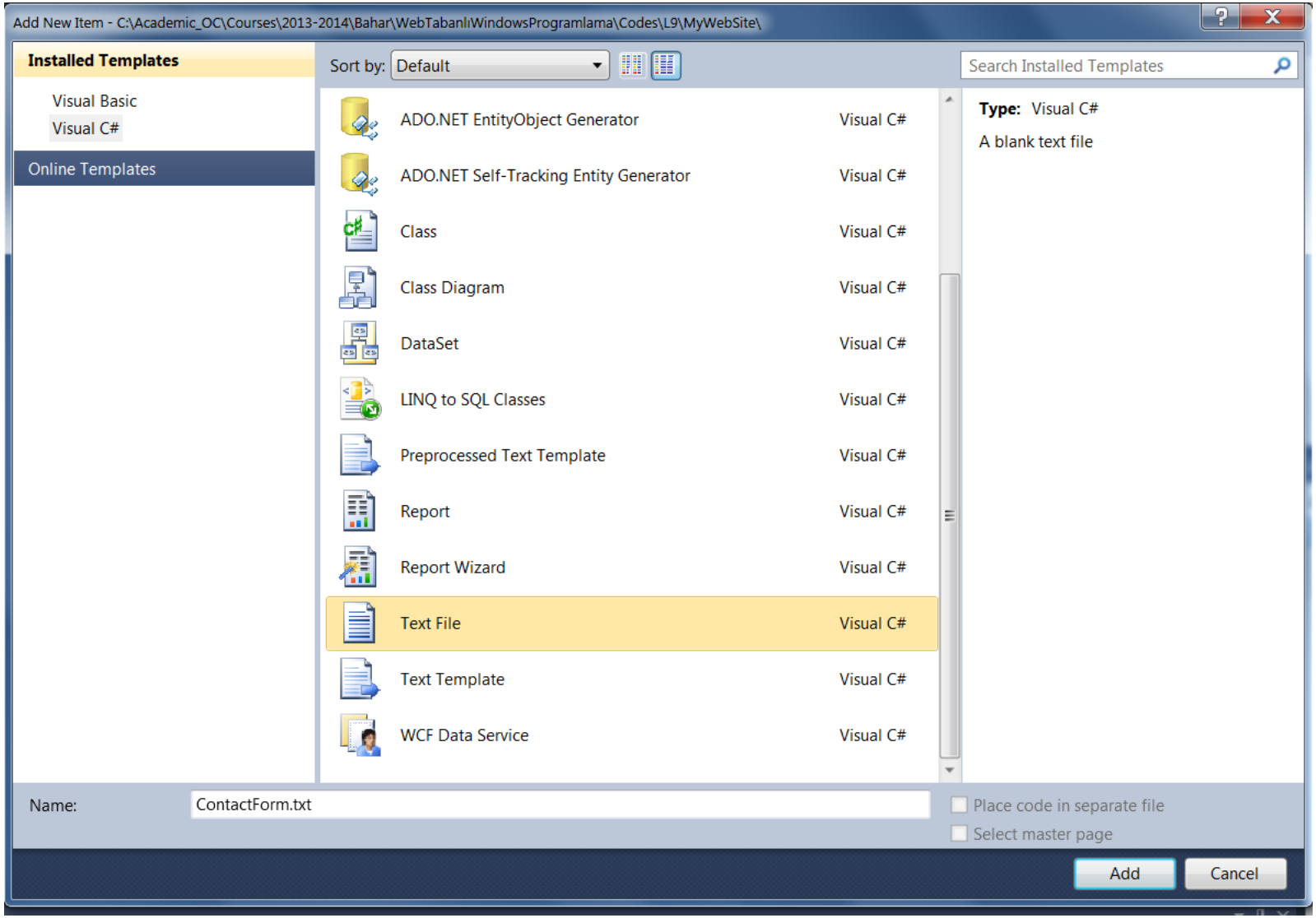


Example
Sending E-Mail from the ContactForm

- Add a new text file to **`App_Data`** folder



Example
Sending E-Mail from the ContactForm



Example
Sending E-Mail from the ContactForm

- Enter the following text:

App_Data/ContactForm.txt*

```
Hi there,

A user has left the following feedback at the site:

Name: ##Name##
E-mail address: ##Email##
Home phone: ##HomePhone##
Business phone: ##BusinessPhone##
Comments: ##Comments##
```

Example
Sending E-Mail from the ContactForm

Open code behind file → **`ContactForm.ascx.cs`**

Add the following namespaces:

```
using System.IO; // Provides access to the File class for reading the file
using System.Net.Mail; // Provides access to the various mail related classes
```

Example
Sending E-Mail from the ContactForm

- In Source View:
  - To hide the entire table programmatically when the form has been submitted:

```
<table class="style1">
```

```
<table class="style1" runat="server" id="FormTable">
```

Example
Sending E-Mail from the ContactForm

- After **`</table>`** tag:

```
</table>

<asp:Label ID="Message" runat="server" Text="Message Sent!" Visible="False"></asp:Label>
```

Example
Sending E-Mail from the ContactForm

- In Design View:
  - **`ValidationSummary`** control's properties:
    - **`ShowSummary = True`**
    - **`ShowMessageBox = False`**

Double-click **Send** button

```
protected void sendButton_Click(object sender, EventArgs e)
{
    if (Page.IsValid)
    {
        string fileName = Server.MapPath("~/App_Data/ContactForm.txt");
        string mailBody = File.ReadAllText(fileName);

        mailBody = mailBody.Replace("##Name##", nameTextBox.Text);
        mailBody = mailBody.Replace("##Email##", emailAddressTextBox.Text);
        mailBody = mailBody.Replace("##HomePhone##", phoneHomeTextBox.Text);
        mailBody = mailBody.Replace("##BusinessPhone##", phoneBusinessTextBox.Text);
        mailBody = mailBody.Replace("##Comments##", commentsTextBox.Text);

        MailMessage myMessage = new MailMessage();
        myMessage.Subject = "Response from web site";
        myMessage.Body = mailBody;
        myMessage.From = new MailAddress("wbwpcourse@gmail.com", "Sender Name");
        myMessage.To.Add(new MailAddress("wbwpcourse@gmail.com", "Receiver Name"));

        SmtpClient mySmtpClient = new SmtpClient();
        mySmtpClient.Host = "smtp.gmail.com";
        mySmtpClient.EnableSsl = true;
        mySmtpClient.Port = 587;
        mySmtpClient.Send(myMessage);

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

Example
Sending E-Mail from the ContactForm

- View `Contact.aspx` in browser

Home > About > Contact

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

Name: Özgü

Email Address: ozgucan@gmail.com

Email Address (Repeat): ozgucan@gmail.com

Home Phone: 3115332

Business Phone: 3115332

Comments: Bla bla

Send

Home > About > Contact

Message Sent!

Read

- `specifiedPickupDirectory`

- Sets the directory where applications save mail messages to be processed by the SMTP server.

http://msdn.microsoft.com/en-us/library/ms164241.aspx