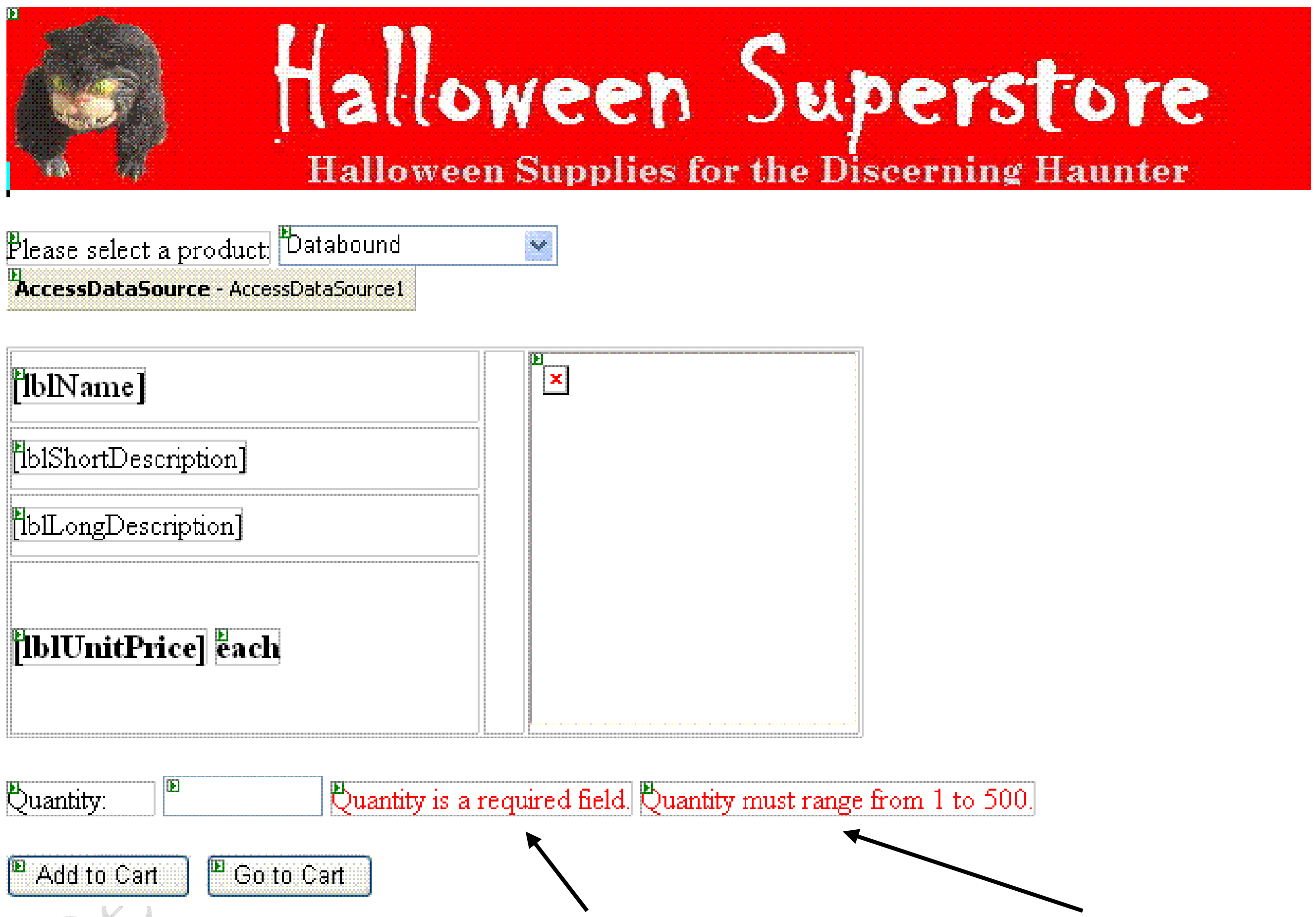
Lecture 6

Validation controls, managing states and navigations

Validation controls

ASP.NET provides six ***validation controls*** that you can use to validate the data on a web form.

The validation controls on the Order form



Quantity is a required field.

Quantity must range from 1 to 500.

Range validator **Required field validator**

Each ***validator*** is associated, a single input control, but you can associate two or more validators with the same input control.

The validation controls provided by ASP.NET

Control	Name	Description
✓	*RequiredFieldValidator*	Checks that an entry has been made
✓	*CampareValidator*	Checks an entry against a constant value or the value of another control. Can also be used to check for a specific data type.
✓	*RangeValidator*	Checks that an entry is written a specified range.
✓	*RegularExpressionValidator*	Checks that an entry matches a pattern that's defined by a regular expression.
✓	*CustomValidator*	Checks an entry using validation code that you write yourself .
▤	*ValidationSummary*	Displays a summary of error messages from the other validation controls

After you add a validator to a web form, you set its properties to determine which input control it validates and how errors are displayed. Below presents the properties you use to do that.

Common validator properties

Property	Description
ControlToValidate	The *ID* of the control to be validated. **That is the most important property!**
Display	Determines how the error message is to be displayed. Specify *Static* to allocate space for the message in the page layout. *Dynamic* to have space allocated only when an error occurs, or *None* to display errors only in a validation summary control. The default is *Static*. If you use two or more validators to validate the same control use *Dynamic*. Then, the validators that pass their validation tests don't take up space on the page.
ErrorMessage	The message that's displayed in the validation control and/or the validation summary control when the validation fails. To display one message in the validation summary and another in the validator, use the Text property for the validator message.
Text	The message that's displayed in the validator when you use the ErrorMessage property to display a message in the validation summary control.
IsValid	Indicates whether the control specified in the *ControlToValidate* property passed the validation.

Enabled	Indicates whether the validation control is enabled.
EnableClientScript	Indicates whether the validation control will be done on the client.
ValidationGroup	Indicates a group of the validation controls.

A validation tests are typically done on the client before the page is posted to the server. In this case a round trip to the server isn't required to display error messages if any invalid data is selected.

In most cases, client-side validation is done when the focus leaves an input control that has validators associated with it. That can happened when the user presses the *Tab* key to move to the next control or click another control to move the focus to that control.

Typical code for processing a page that contains validation controls

```
protected void btnAdd_Click(object sender, EventArgs e)
 {
    if (Page.IsValid)
    {
       CartItem item = new CartItem();

       item.Product = selectedProduct;
       item.Quantity = Convert.ToInt32(txtQuantity.Text);

       this.AddToCart(item);

       Response.Redirect("Cart.aspx");
    }
 }
```

Description

- If a browser supports DHTML (DHTML), the validation controls do their validation on the client using client-script. That way, the validation is performed and error messages are displayed without the page posted to the server.
- Validation is always done on the server too, right after the page is initialized, so the validation is done whether or not the browser supports DHTML. Although most browsers support DHTML, your code should still check that a page is valid in case a browser doesn't support DHTML.
- Validation is performed on the server when you click a button whose *CausesValidation* property is set to *True*. To create a button that doesn't initiate validation, you can set this property to False.
- Validation is performed on the client when the focus leaves the input control. The exception is a required field validator, which performs its validation only when you click a button whose *CausesValidation* property is set to True or when you enter a value into a control and then clear and leave the control.

- If a validation control indicates invalid data, the *IsValid* property of that control is set to *False* and the *IsValid* property of the page is set to *False*. These properties can be tested in your *C#* code.
- If you want to perform validation only on the server, you can set *EnableClientScript* properties of the validation control to *False*. Then, no client-side scripts are generated for validating the data.

How to use the required field validator

This validator checks that the user entered a value into an input control. If the user doesn't enter a value, the *IsValid* property of the validator is set to *False* and its error message is displayed.

Additional property of a required field validator

Property	Description
InitialValue	The initial value of the control that's validates. If the value isn't changed, the validation fails. The default value is an empty string.

Below you can see three examples how to use the required field validator.

A required field validator that checked for a required entry

```
Name:&nbsp;
<asp:TextBox id="txtName" runat="server" Width="213px"></asp:TextBox>&nbsp;
<asp:RequiredFieldValidator id=" RequiredFieldValidator1" runat="server"
        ControlToValidate="txtName" ErrorMessage="You must enter a name.">
</asp:RequiredFieldValidator>
```

In this example, the validator is used to check for a required entry in a text box. If the user doesn't enter anything into the text box, the text in the *ErrorMessage* property is displayed.

The examples below show how you can use the *InitialValue* property of the required field validator to check that the user changed the initial value of a control. By default, this property is set to an empty string, which is what you want if the input control is empty. If you specify an initial value for an input control, however, you'll want to set the *InitialValue* property of the required field validator to the same value.

The example below uses the technique with a text box. Here, the initial value indicates the format for a date entry. If the user doesn't change this value, the validation test will fail.

A required field validator that checks an initial value is changed

```
Birth date:&nbsp;
<asp:TextBox id="txtBirthdate" runat="server" Width="224px"
        Text="mm/dd/yyyy"></asp:TextBox>&nbsp;
<asp:RequiredFieldValidator id="RequiredFieldValidator2" runat="server"
        ControlToValidate="txtBirthDate" InitialValue="mm/dd/yyyy"
        ErrorMessage="You must enter a birth date.">
</asp:RequiredFieldValidator>
```

The third example uses the *InitialValue* property with a list box. Here, the *InitialValue* property is set to *None*, which is the value of the first item in the list. That way, if the user doesn't select another item, the validation test will fail. You can use this technique with a drop-down list or a radio button list.

A required field validator that checks an option is chosen from a list box

```
<asp:ListBox id="lstCardType" runat="server">
        <asp:ListItem Value="None" Selected="True">--Select a credit card--</asp:ListItem>
        <asp:ListItem Value="VISA">Visa</asp:ListItem>
        <asp:ListItem Value="MC">Master Card</asp:ListItem>
        <asp:ListItem Value="AMEX">American Express</asp:ListItem>
</asp:ListBox>
<asp:RequiredFieldValidator id="RequiredFieldValidator1" runat="server"
        ControlToValidate="lstCardType" InitialValue="None"
        ErrorMessage="You must select a credit card.">
</asp:RequiredFieldValidator>
```

How to use the compare validator

This validator lets you compare the value entered into an input control with a constant value or the value of another control. You can also use the compare validator to make sure that the value is a particular data type.

Additional properties of a compare validator

Property	Description
ValueToCompare	The value that the control specified in the ControlToValidate property should be compared to.
Operator	The type of comparison to perform *(Equal, NotEqual, GreaterThan, GreaterThanEquel, LessThan, LessThanEqual, or DataTypeCheck).*
Type	The data type to use the comparison *(String, Integer, Double, Date, or Currency).*
ControlToCompare	The *ID* of the control that the value of the control specified in the *ControlToValidate* property should be compared to.

A compare validator that checks for a value greater than zero

```
Quantity:&nbsp;
<asp:TextBox id=”txtQuantity” runat=”server” Width=”50px”>
</asp:TextBox>&nbsp;
<asp:CompareValidator id=”CompareValidator1” runat=”server”
        ControlToValidate=”txtQuantity” ValueToCompare=”0” Operator=”GreaterThan”
        Type=”Integer”
        ErrorMessage=”Quantity must be greater than zero.”>
</asp:CompareValidator>
```

A compare validator that checks for a numeric entry

```
Quantity:&nbsp;
<asp:TextBox id=”txtQuantity” runat=”server” Width=”50px”>
</asp:TextBox>&nbsp;
<asp:CompareValidator id=”CompareValidator2” runat=”server”
ControlToValidate=”txtQuantity”
          Operator=”DataTypeCheck” Type=”Integer” ErrorMessage=”Quantity must be
numeric.”>
</asp:CompareValidator>
```

A compare validator that compares the values of two controls

```
Start Date:&nbsp;
<asp:TextBox id=”txtStartDate” runat=”server” Width=”64px”></asp:TextBox>
End Date:&nbsp;
<asp:TextBox id=”txtEndDate” runat=”server” Width=”64px”></asp:TextBox>&nbsp;
<asp:CompareValidator id=”CompareValidator3” runat=”server”
          ControlToValidate=”txtEndDate” ControlToCompare=”txtStartDate”
          Operator=”GreaterThan” Type=”Date”
          ErrorMessage=”End Date must be greater than Start Date.”>
</asp:CompareValidator>
```

How to use the range validator

The ***range validator*** validates user input by making sure that it falls within a given range of values.

Additional properties of the range validator

Property	Description
MinimumValue	The minimum value allows for the control
MaximumValue	The maximum value allows for the control
Type	The data type to use for the comparison *(String, Integer, Double, Date, or Currency)*

A range validator that checks for a numeric range

```
Days:&nbsp;
<asp:TextBox id="txtDays" runat="server" Width="64px">
</asp:TextBox>&nbsp;
<asp:RangeValidator id="RangeValidator1" runat="server" ControlToValidate="txtDays"
        Type="Integer" MaximumValue="14" MinimumValue="1"
        ErrorMessage="Days must be between 1 and 14.">
</asp:RangeValidator>
```

How to set a range at runtime

A range validator that checks a date range that's set at runtime

```
Arrival Date:&nbsp;
<asp:TextBox id="txtArrival" runat="server" Width="64px">
</asp:TextBox>&nbsp;
<asp:RangeValidator id="valArrival" runat="server" ControlToValidate="txtArrival"
Type="Date"
        ErrorMessage="You must arrive within 30 days.">
</asp:RangeValidator>
```

Code that sets the minimum and maximum values when the page is loaded

```
protected void Page_Load(object sender, EventArgs e)
{
        If (!IsPostBack)
        {
                valArrival.MinimumValue  = DateTime.Today.ToShortDateString();
                valArrival.MaximumValue
=DateTime.Today.AddDays(30).ToSrortDateString();
        }
}
```

How to use the validation summary controls

The validation summary control lets you summarize all the controls on a page. The summary can be displayed directly on the page, or, if validation is being performed on the client, in a separate message box.

Properties of the validation summary control

Property	Description
DisplayMode	Specifies how the error messages from the validation controls are to be displayed. The available values are *BulletList, List*, or *SingleParagraph*, The default is *BulletList*.

HeaderText	The text that's displayed before the list of error messages.
ShowSummary	A *Boolean* value that determines whether the validation summary should be displayed on the web page. The default is *True*.
ShowMessageBox	A *Boolean* value that determines whether the validation summary should be displayed in a message box (client-side validation only). The default is *False*.

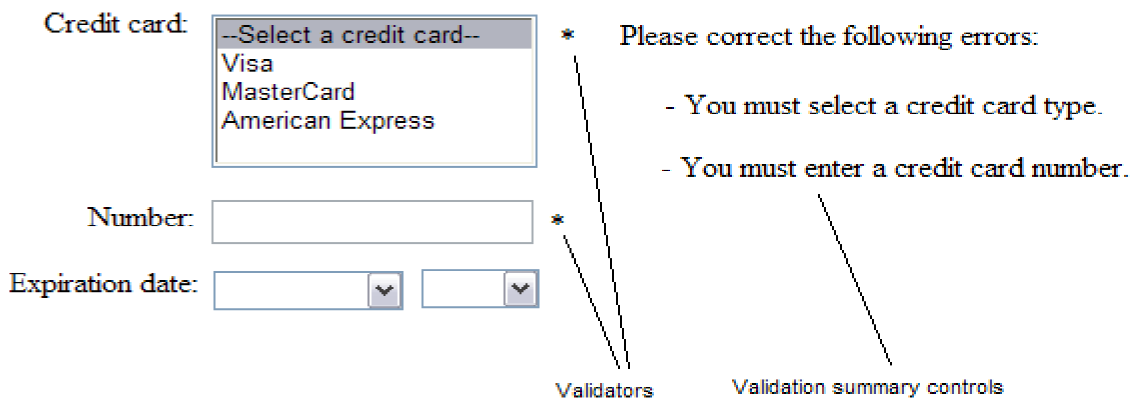
Two validators and a validation summary control that are displayed on the web page

```
<asp:RequiredFieldValidator id="RequiredFieldValidator1" runat="server"
        ErrorMessage="You must select a credit card type."
        Display="Dynamic" ControlToValidate="lstCardType" InitialValue="None">*
</asp:RequiredFieldValidator>

<asp:RequiredFieldValidator id="RequiredFieldValidator2" runat="server"
        ErrorMessage="You must select a credit card number." Display="Dynamic"
        ControlToValidate="txtCardNumber">*
</asp:RequiredFieldValidator>

<asp:ValidationSummary id="ValidationSummary1" runat="server"
        HeaderText="Please correct the following error:">
</asp:ValidationSummary>
```

How the controls appear on the web page



Please correct the following errors:

- You must select a credit card type.
- You must enter a credit card number.

Description

- The ***validation summary control*** displays a summary of error messages that were generated by the page's validator. The summary can be displayed on the web page or in a separate message box.

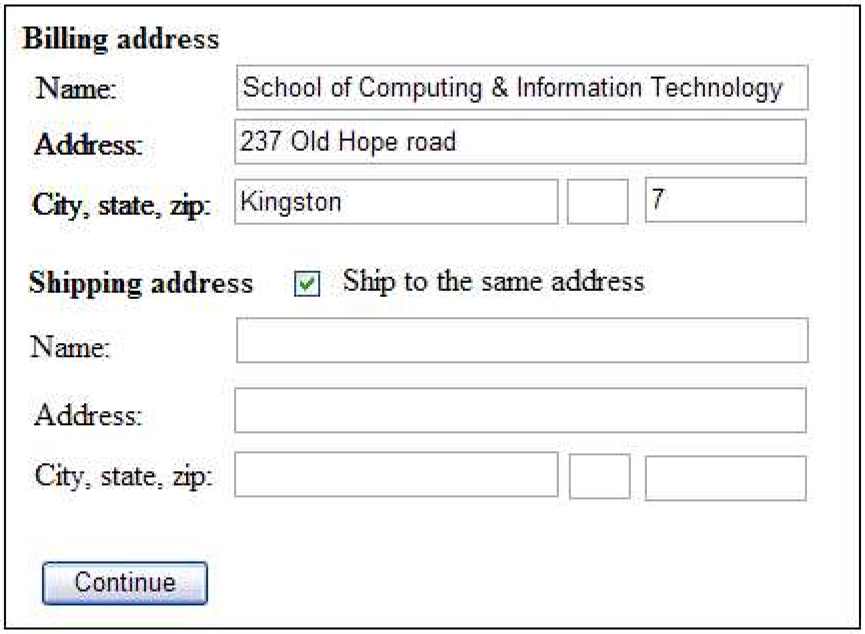
- The error messages displayed in the validation summary controls come from the ErrorMessage property of the page's validators. If you want to display a different message in the validator, set the `Text` property of that control.
- If you don't want to display an error message in the validator, set its Display property to None.

How to initiate server validation manually

Attributes that cause group validation when a button is clicked

Attributes	Description
CausesValidation	Specifies whether validation should be performed when the user clicks the button.
ValidationGroup	Specifies the name of the group to be validated if *CausesValidation* is *True*

A web page that accepts a billing and shipping address



Billing address			
Name:	School of Computing & Information Technology		
Address:	237 Old Hope road		
City, state, zip:	Kingston		7
Shipping address	☑ Ship to the same address		
Name:			
Address:			
City, state, zip:			

A text box with a validator that specifies a validation group

```
<asp:TextBox ID="txtShipToName" runat="server" />
<asp:RequiredFieldValidator ID="RequiredFieldValidator6" runat="server"
        ControlToValidate="txtShipToName" ErrorMessage="You must enter a ship-to name."
        ValidationGroup="ShipTo" >
</asp:RequiredFieldValidator>
```

A button that specified a validation group

```
<asp:Button ID="btnContinue" runat="server" Text="Continue"
        ValidationGroup="BillTo" OnClick="btnContinue_Click" />
```

C# code that conditionally validates the ShipTo group

```
If (!chkShipToSameAsBillTo.Checked)
        Page.Validate("ShipTo");
```

A regular expression validator

The regular expression validator lets you match data to a pattern you specify. A ***regular expression*** is string made up of special pattern-matching symbols.

An additional property of the regular expression validator

Property	Description
ValidationExpression	A string that specifies a regular expression. The regular expression defines a pattern that the input data must match to be valid

A regular expression validator that validates five-digit number

```
<asp:TextBox ID="txtZipCode" runat="server"></asp:TextBox>
<asp:RegularExpressionValidator ID="RegularExpressionValidator1" runat="server"
        ControlToValidate="txtZipCode" ValidationExpression="\d{5}"
        ErrorMessage="Must be a five-digit U.S. zip code.">
<asp:RegularExpressionValidator>
```

A regular expression validator that validates phone numbers

```
<asp:TextBox ID="txtPhone" runat="server"></asp:TextBox>
<asp:RegularExpressionValidator ID="RegularExpressionValidator2" runat="server"
        ControlToValidate="txtPhone" ValidationExpression="((\(\d{3}\) ?)|(\d{3}-))?\d{3}-
        \d{4}"
        ErrorMessage="Must be a valid phone number.">
<asp:RegularExpressionValidator>
```

Common regular expression elements

Element	Description
Ordinary character	Matches any character other than *., $, ^, [, {, (, |,), *, +, ?,* or \
\	Matches any character that follows.
\d	Matches any decimal digit (0-9)
\D	Matches any character other than a decimal digit.
\w	Matches any character any word character (a-z, A-Z, and 0-9)

\W	Matches any character other than a word character
\s	Matches any white space character (space, tab, new line, etc.)
\S	Matches any character other than a whitespace character
[abcd]	Matches any character included between brackets
[^abcd]	Matches any character that is not included between the brackets
[a-z]	Matches any character in the indicated range
{n}	Matches exactly n occurrences of the preceding element or group
{n,}	Matches at least n occurrences of the preceding element or group
{n, m}	Matches at least n but no more than m occurrences of the preceding element or group
*	Matches zero or more occurrences of the preceding element
?	Matches zero or one occurrences of the preceding element
+	Matches one or more occurrences of the preceding element
|	Matches any of the elements separated by the vertical bar
( )	Groups the elements that appear between the parentheses

To start, you need specify any ordinary character, such as a letter or a decimal digit. If a character must be an A, for example, you just include that character in the expression. To include a character other than ordinary character, you must precede it with a backslash. For example, \(specifies that the character must be a left parenthesis, \] specifies that the character must be a right bracket, and \\ specifies that the character must be a backslash. A backslash that's used in this way is called an ***escape character***.

You can also specify a ***character class***, which consist of a set of characters. For example, \d indicates that the character must be a decimal digit, \w indicates that the character must be a ***word character***, and \s indicates that the character must be ***whitespace character***. The upper case version of these elements - \D, \W, and \S – match any character that is not a decimal digit, word character, or whitespace character.

To create a list of possible characters, you enclose them in brackets. For example, *[abcd]* specifies that the character must be the letter *a, b, c,* or *d,* and *[abc]* specifies that the character must be a low case letter. One fairly common construct is *[a-zA-Z],* which specifies that the character must be a lowercase or uppercase letter.

You can also use ***quantifiers*** to indicate how many of the preceding element the data input must contain. To specify an exact number, you just code it in brackets. For example, \d{5} specifies that the input data must be a five-digit number. You can also specify a minimum number and a maximum number of characters. For example, \w{6,20} specifies that the input data must contain from six to twenty word characters. You can also omit the maximum number to require just a minimum number of characters. For example, \w{6} specifies that input data must contain at least 6 word characters. You can also use the *, ?, and + quantifiers to specify zero or more, zero or one, or one or more characters.

If the input data can match one or more patterns, you can use the vertical bar to separate elements. For example, \w+/s{1} means that the input data must contain one or more word characters or a single whitespace characters.

To create groups of elements, you can use parentheses. Then, you can apply quantifier to the entire group or you can separate groups with a vertical bar. For example, (AB)|(SB) specifies that the input characters must be either AB or SB. And (\d{3}-)? specifies that the input characters must contain zero or one occurrence of three-digit number followed by hyphen.

Examples of regular expressions

Expression	Example	Description
\d{3}	289	A three digit number
\w{8,20}	Frankenstein	At least eight but not more than twenty word characters
\d{2}-\d{4}	10-3944	A two digit number followed by hyphen and a four-digit number
\w{1,8}.\w{1,3}	freddy.jpg	Up to eight letters or numbers, followed by a period and up to three letters or numbers
(AB)|(SB)-\d{1,5}	SB-3276	The letters AB or SB, followed by hyphen and a one-to five-digit number
\d{5}(-d\{4})?	93711-2765	A five-digit number, optionally followed by a hyphen and a four-digit number
\w*\d\w*	arm01	A text entry that contains at least one numeral
[xyz]\d{3}	x023	The letter x, y, or z, followed by a three-digit number

Custom validator

You can code your own validation routine that's executed when the page is submitted to the server. This technique is frequently used to validate input data that requires a database lookup.

Properties of the *ServerValidateEventArgs* class

Property	**Description**
Value	The text string to be validated
IsValid	A Boolean property that you set to True if the value passes the validation test or to False

Aspx code for a text box and custom validator

```
<asp:TextBox ID="txtProductCode" runat="server"></asp:TextBox>
<asp:CustomValidator id="valProductCode" runat="server"
        ControlToValidate="txtProductCode"
        ErrorMessage="Product code must be in database"
        OnServerValidate="valProductCode_ServerValidate">
</asp:CustomValidator>
```

C# code for the custom validator

```
protected void valProductCode_ServerValidate(object source, ServerValidateEventArgs args)
{
args.IsValid = HalloveenDB.CheckProductCode(args.Value);
}
```

If the user doesn't enter a value into the associated input control, the custom validator doesn't perform its validation test. As a result, you should also provide a required field validator if a value is required.

A validation routine that validates credit card numbers

C# code for a credit card validator

```
protected void valCreditCardNumber_ServerValidate(object source,
                                                  ServerValidatEventArgs args)
{
        args.IsValid = ValidateCreditCard(args.Value);
}

Private bool ValidateCreditCard(string cardNumber)
{
        int digitSum = 0;
        string digits = "";
        string reverseCardNumber = "";

        // Remove spaces and reverse string
        cardNumber = cardNumber.Replace(" ", null);
        for (int i = cardNumber.Length – 1; i >= 0; i)
                reversedCardNumber += cardNumber[i];
        // Double the digits in evennumbered positions
        for (int i = 0; I < reversedCardNumber.Length; i++)
        {
                if ((i + 1) % 2 == 0)
                        digits += Convert.ToInt32(reversedCardNumber.Subsring(i, 1)) * 2;
                else
                        digits += reversedCardNumber.Subsring(i, 1));
        }

        // Add the digits
        for (int i = 0; i < digits.Length; i++)
                digitSum += Convert.ToInt32(digits.Substring(i, 1));

        // Check that the sum is divisible by 10
        if ((digitSum % 10) == 0)
                return true;
        else
                return false;
}
```

Description of that algorithm

1. Removes any spaces from the number.
2. Reverses the digits in the number.
3. Doubles the digits in even-numbered positions. If the original digit is 5 or greater, this will insert an additional digit into the number.
4. Adds up the individual digits.
5. Divides the result by 10. If the number is 0, the credit card number is valid.