Validations

Today You Will Learn

- Understanding Validation
- The Validation Controls

Understanding Validation

What's particularly daunting is the range of possible mistakes that users can make.

Examples are:

- Users might ignore an important field and leave it blank.
- Users might try to type a short string of nonsense to circumvent a required field.
- Users might make an honest mistake, such as entering a typing error, entering a nonnumeric character in a number field.
- Malicious users might try to exploit a weakness in your code by entering carefully structured wrong values

Understanding Validation

A web application is particularly susceptible to these problems, because it relies on basic HTML input controls that don't have all the features of their Windows counterparts.

Strategy isn't as easy in a server-side web page. To perform validation on the web server, you need to post back the page, and it just isn't practical to post the page back to the server every time the user types a letter.

Understanding Validation

To avoid this sort of problem, you need to perform all your validation at once when a page (which may contain multiple input controls) is submitted. You then need to create the appropriate user interface to report the mistakes. All this involves considerable amount of developers time.

ASP.NET aims to save you this trouble and provide you with a reusable framework of validation controls that manages validation details by checking fields and reporting on errors automatically.

Understanding Validation

Control Class	Description
RequiredFieldValidator	Validation succeeds as long as the input control doesn't contain an empty string.
RangeValidator	Validation succeeds if the input control contains a value within a specific numeric, alphabetic, or date range.
CompareValidator	Validation succeeds if the input control contains a value that matches the value in another input control, or a fixed value that you specify.
RegularExpressionValidator	Validation succeeds if the value in an input control matches a specified regular expression.
CustomValidator	Validation is performed by a user-defined function.

Understanding Validation

- Each validation control can be bound to a single input control. In addition, you can apply more than one validation control to the same input control to provide multiple types of validation.
- If you use the RangeValidator, CompareValidator, or RegularExpressionValidator, validation will automatically succeed if the input control is empty, because there is no value to validate. If this isn't the behavior you want, you should also add a RequiredFieldValidator and link it to the same input control.

Understanding Validation

Server-Side Validation

Every button has a "CausesValidation" property, which can be set to True or False.

- If False, ASP.NET will ignore the validation controls, the page will be posted back, and your event handling code will run normally.
- If True (the default), ASP.NET will automatically validate the page when the user clicks the button. It does this by performing the validation for each control on the page. If any control fails to validate, ASP.NET will return the page with some error information, depending on your settings.

Understanding Validation

Client-Side Validation

Performing Validation at both client(using javascript) and server, ASP.NET makes sure your application can be as responsive as possible while also remaining secure.

The Validation Controls

The validation controls are found in the System.Web.UI.WebControls namespace and inherit from the BaseValidator class.

Table 9-2. Properties of the BaseValidator Class

Property	Description
ControlToValidate	Identifies the control that this validator will check. Each validator can verify the value in one input control. However, it's perfectly reasonable to "stack" validators—in other words, attach several validators to one input control to perform more than one type of error checking.
ErrorMessage and ForeColor	If validation fails, the validator control can display a text message (set by the ErrorMessage property). By changing the ForeColor, you can make this message stand out in angry red lettering. (In previous versions of ASP.NET, the validation controls used red lettering by default. But if you create a new application for ASP.NET 4, your validation messages will use ordinary black text, unless you set the ForeColor property or use CSS styles in your page.)
Display	Allows you to configure whether this error message will be inserted into the page dynamically when it's needed (Dynamic) or whether an appropriate space will be reserved for the message (Static). Dynamic is useful when you're placing several validators next to each other. That way, the space will expand to fit the currently active error indicators, and you won't be left with any unseemly whitespace. Static is useful when the validator is in a table and you don't want the width of the cell to collapse when no message is displayed. Finally, you can also choose None to hide the error message altogether.

The Validation Controls

IsValid	After validation is performed, this returns True or False depending on whether it succeeded or failed. Generally, you'll check the state of the entire page by looking at its IsValid property instead to find out if all the validation controls succeeded.
Enabled	When set to False, automatic validation will not be performed for this control when the page is submitted.
EnableClientScript	If set to True, ASP.NET will add JavaScript and DHTML code to allow client-side validation on browsers that support it.

Table 9-3. *Validator-Specific Properties*

Validator Control	Added Members
RequiredFieldValidator	None required
RangeValidator	MaximumValue, MinimumValue, Type
CompareValidator	ControlToCompare, Operator, Type, ValueToCompare
RegularExpressionValidator	ValidationExpression
CustomValidator	ClientValidationFunction, ValidateEmptyText, ServerValidate event

The Validation Controls

Other Display Options

In some cases, you might have already created a carefully designed form that combines multiple input fields. Perhaps you want to add validation to this page, but you can't reformat the layout to accommodate all the error messages for all the validation controls. In this case, you can save some work by using the ValidationSummary control.

To try this, set the Display property of the RangeValidator control to None. This ensures the error message will never be displayed. However, validation will still be performed and the user will still be prevented from successfully clicking the OK button if some invalid information exists on the page.

Next, add the ValidationSummary in a suitable location (such as the bottom of the page):

```
<asp:ValidationSummary id="Errors" runat="server" />
```

When you run the page, you won't see any dynamic messages as you enter invalid information and tab to a new field. However, when you click the OK button, the ValidationSummary will appear with a list of all error messages.

Ordinarily, Text is left empty, and the validator doesn't show any content in the web page. However, if you set both Text and ErrorMessage, the ErrorMessage value will be used for the summary while the Text value is displayed in the validator.

The Validation Controls

Manual Validation

You can create manual validation in one of three ways:

- Use your own code to verify values. In this case, you won't use any of the ASP.NET validation controls.
- Disable the EnableClientScript property for each validation control. This allows an invalid page to be submitted, after which you can decide what to do with it depending on the problems that may exist.
- Add a button with CausesValidation set to False. When this button is clicked, manually validate the page by calling the Page.Validate() method. Then examine the IsValid property, and decide what to do.

The Validation Controls

Validating with Regular Expressions

Table 9-4. Regular Expression Characters

Character	Description
*	Zero or more occurrences of the previous character or subexpression. For example, 7*8 matches 7778 or just 8.
+	One or more occurrences of the previous character or subexpression. For example, 7+8 matches 7778 but not 8.
( )	Groups a subexpression that will be treated as a single element. For example, (78)+ matches 78 and 787878.
{m,n}	The previous character (or subexpression) can occur from m to n times. For example, A{1,3} matches A, AA, or AAA.
|	Either of two matches. For example, 8|6 matches 8 or 6.
[ ]	Matches one character in a range of valid characters. For example, [A-C] matches A, B, or C.
[^]	Matches a character that isn't in the given range. For example, [^A-B] matches any character except A and B.
.	Any character except newline. For example, .here matches where and there.

The Validation Controls

Validating with Regular Expressions (Cont.)

\s	Any whitespace character (such as a tab or space).
\S	Any nonwhitespace character.
\d	Any digit character.
\D	Any character that isn't a digit.
\w	Any "word" character (letter, number, or underscore).
\W	Any character that isn't a "word" character (letter, number, or underscore).

The Validation Controls

Table 9-5. *Commonly Used Regular Expressions*

Content	Regular Expression	Description
E-mail address*	\S+@\S+\.\S+	Check for an at (@) sign and dot (.) and allow nonwhitespace characters only.
Password	\w+	Any sequence of one or more word characters (letter, space, or underscore).
Specific-length password	\w{4,10}	A password that must be at least four characters long but no longer than ten characters.
Advanced password	[a-zA-Z]\w{3,9}	As with the specific-length password, this regular expression will allow four to ten total characters. The twist is that the first character must fall in the range of a–z or A–Z (that is to say, it must start with a nonaccented ordinary letter).
Another advanced password	[a-zA-Z]\w*\d+\w*	This password starts with a letter character, followed by zero or more word characters, one or more digits, and then zero or more word characters. In short, it forces a password to contain one or more numbers somewhere inside it. You could use a similar pattern to require two numbers or any other special character.
Limited-length field	\S{4,10}	Like the password example, this allows four to ten characters, but it allows special characters (asterisks, ampersands, and so on).
U.S. Social Security number	\d{3}-\d{2}-\d{4}	A sequence of three, two, then four digits, with each group separated by a dash. You could use a similar pattern when requiring a phone number.

The Validation Controls

Custom Validators

The validation takes place in the event handler for the CustomValidator.ServerValidate event. This method receives the value it needs to validate (e.Value) and sets the result of the validation to True or False (e.IsValid).

By default, custom validation isn't performed on empty values. However, you can change this behavior by setting the CustomValidator.ValidateEmptyText property to True

Custom server-side validation isn't performed until the page is posted back. If it troubles you, then you can use the CustomValidator.ClientValidationFunction property. Add a client-side JavaScript function to the .aspx portion of the web page.

```
<asp:CustomValidator id="vldCode" runat="server"
        ErrorMessage="Try a string that starts with 014."
        ControlToValidate="txtCode"
        ClientValidationFunction="MyCustomValidation" />
```

The Validation Controls

```
<script type="text/javascript">
  <!--
    function MyCustomValidation(objSource, objArgs) {
      // Get value.
      var number = objArgs.Value;
      // Check value and return result.
      number = number.substr(0, 3);
      if (number % 7 == 0) {
        objArgs.IsValid = true;
      }
      else {
        objArgs.IsValid = false;
      }
    }
  // -->
  </script>
```

The Validation Controls

Validation Groups

In more complex pages, you might have several distinct groups of controls, possibly in separate panels. In these situations, you may want to perform validation separately.

For example, you might create a form that includes a box with login controls and a box underneath it with the controls for registering a new user. Each box includes its own submit button, and depending on which button is clicked, you want to perform the validation just for that section of the page. This scenario is possible thanks to a feature called *validation groups*. To create a validation group, you need to put the input controls, the validators, and the CausesValidation button controls into the same logical group.