

User Input Validation in ASP.NET

192 out of 256 rated this helpful - [Rate this topic](#)

Anthony Moore
Microsoft Corporation

July 2000
Updated March 2002

Summary: Provides a brief overview of the validation framework in ASP.NET and walks through an example of adding validation to a page. (11 printed pages)

Contents:

[Introduction](#)
[The Problem](#)
[The Objective](#)
[The Solution-Overview](#)
[Client Features](#)
[What Is a Validator?](#)
[Validator Walk-Through](#)
[It's Not Voluntary](#)
[Getting Regular](#)
[Comparing Apples and Apples](#)
[Custom Fit](#)
[The Finale](#)
[Sample Code](#)

Introduction

Validating user input is a common scenario in a Web-based application. For production applications, developers often end up spending a lot more time and code on this task than we would like. In building the ASP.NET page framework, it was important to try and make the task of validating input a lot easier than it has been in the past.

The Problem

In HTML 3.2, validating data is a difficult process. Each time you get a request, you need to write code to check the input and write any errors the user has made back to the page to help the user to correctly fill in the form. This is a taxing process for end users, developers, and servers alike.

DHTML and scripting languages improve things somewhat. It is possible to provide the user with immediate feedback on bad input and to prevent them from posting a page until it has been corrected. However, it can be almost impossible to guarantee that every user of your site has the required scripting environment. This usually means that if you want to use script to improve the interface of your pages, you have to write the same validation logic twice, once on the client, and again on the server, just in case the client script cannot be executed.

The Objective

Our objective with validation is as follows:

- Provide components that can perform 90 percent or more of the typical input validation tasks for data entry pages.

- Have these components perform rich script-based validation on modern browsers that can also effectively fall back to pure HTML 3.2 server-based validation, if required.
- Provide a flexible API so that any validation tasks not covered by the components are easy to complete.

We visited a large number of real pages to determine the sort of scenarios these components needed to be able to handle. We wanted to dramatically reduce the amount of validation code needed for future applications.

The Solution—Overview

The validator controls are the main elements of the solution. A validator is a visual ASP.NET control that checks a specific validity condition of another control. It generally appears to the user as a piece of text that displays or hides depending on whether the control it is checking is in error. It can also be an image, or can even be invisible and still do useful work. There are five types of validator controls that perform different types of checks.

Another element in our solution is the ValidationSummary control. Large data entry pages generally have an area where all errors are listed. The ValidationSummary automatically generates this content by gathering it up from validator controls on the page.

The final element is the Page object itself. It exposes the all-important "IsValid" property, which you check in server code to determine if all of the user input is OK.

Client Features

Most Web sites do all of their validation checks on the server. You need to write the server-based checks anyway for clients without script, so it can be hard to justify writing it all over again for rich clients.

Validation controls change all that, because almost all the duplicated logic is encapsulated in the controls. If a client with Internet Explorer 4.0 or later uses your page, it can do the same input validation that takes place on the server without you having to write any special client script.

The client side validation has a number of features:

- Errors can appear and disappear immediately after the bad input is entered/corrected. This immediate feedback makes it much easier to correct bad input.
- Post-back is prevented if there are errors that are detectable on the client, saving the user time and reducing hits on the server.
- The ValidationSummary updates itself without posting back if it detects errors.
- The ValidationSummary can optionally display a message box to the user if there are errors.
- The client logic is all contained in a JScript library, so no ActiveX components or Java applets are used.
- An object model is exposed on the client to allow enhancement of client-side validation and behavior.

What Is a Validator?

In order to use validators effectively, it helps to have a firm definition of what they are. Let's refine our previous definition a little:

"A validator is a control that checks one input control for a specific type of error condition and displays a description of that problem."

This is an important definition, because it means that you frequently need to use more than one validator control for each input control.

For example, if you want to check whether or not user input in a text box is (a) nonblank, (b) a valid date between a particular range and (c) less than the date in another text input control, you would want to use three validators. While this might seem cumbersome, remember that to be helpful to the user, you would want to have three different text descriptions for all these cases.

The different types of validators are listed as follows:

RequiredFieldValidator	Checks that the user has entered or selected anything.
RegularExpressionValidator	Checks user input against a regular expression. This allows a wide variety of checks to be made and can be used for things like ZIP codes and phone numbers.
CompareValidator	Compares an input control to a fixed value or another input control. It can be used for password verification fields, for example. It is also possible to do typed date and number comparisons.
RangeValidator	Much like CompareValidator, but can check that the input is between two fixed values.
CustomValidator	This allows you to write your own code to take part in the validation framework.

Validator Walk-Through

To demonstrate validation, we will walk through the process of adding validation to an existing page. Here are some example requirements:

Write a web page to collect a new user id and password. The user id must contain 6-10 alpha characters and must not already be in use. The password must contain 4-12 letters and at least one of the characters "@#\$\$%^&*./". The user must re-enter the password to make sure they entered it correctly.

I am going to start with an HTML page that has been minimally converted to work with the ASP+ page framework.

The process of converting the page includes the following steps:

1. Change the extension from ".html" or ".asp" to ".aspx".
2. Change the form and all the input tags to be "runat=server".
3. Use "ID" instead of "name".

Starting code:

```
<html>
<head>
<title>Validation Sample</title>
</head>
<body>

<form runat=server>
<p>Please enter a new User ID and Password:</p>
<table>
  <tr>
    <td>User ID:</td>
    <td><input type=text runat=server id=txtName></td>
  </tr>
  <tr>
    <td>Password:</td>
    <td><input type=password runat=server id=txtPWord></td>
  </tr>
  <tr>
    <td>Re-enter Password:</td>
    <td><input type=password runat=server id=txtRePWord></td>
  </tr>
</table><br>
<input type=submit runat=server id=cmdSubmit value=Submit>
</form>

</body>
</html>
```

[MSDN Library](#)

[.NET Development](#)

[Articles and Overviews](#)

[Web Applications \(ASP.NET\)](#)

[ASP.NET](#)

[User Interface](#)

[ASP.NET Spiced: AJAX](#)

[Creating Dynamic Data Entry User Interfaces](#)

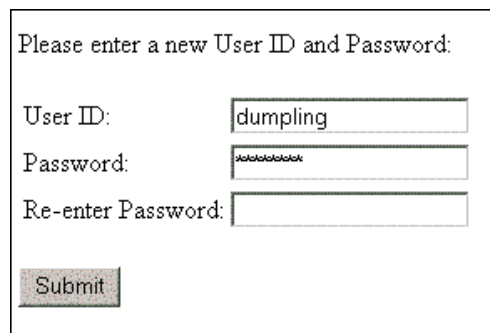
[Creating a Plug-In Framework](#)

[Dynamic Creation of Validation Controls](#)

[Optimizing Color Quantization for ASP.NET Images](#)

User Input Validation in ASP.NET

Starting page:



It's Not Voluntary

The first thing we need to enforce is that the fields get filled in at all.

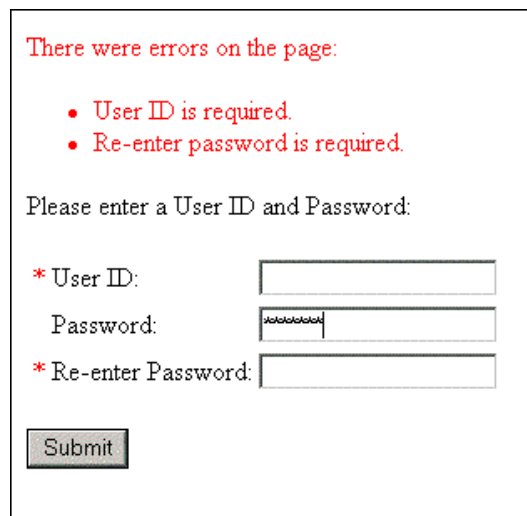
In front of each field, we add a `RequiredFieldValidator`. If the input field is blank, we want to display an asterisk (*) in front of the field and report a text error in a summary area. Here is how we add a `RequiredFieldValidator` to the User ID field:

```
<tr>
  <td>
    <asp:RequiredFieldValidator runat=server
      ControlToValidate=txtName
      ErrorMessage="User ID is required."> *
    </asp:RequiredFieldValidator>
  </td>
  <td>User ID:</td>
  <td><input type=text runat=server id=txtName></td>
</tr>
```

The * is displayed next to the label if the input is blank. The error message is reported in a summary. The "ControlToValidate" property specifies the ID of the control to validate. The final step is to add a validation summary to the top of the page like so:

```
<asp:ValidationSummary runat=server
  HeaderText="There were errors on the page:" />
```

Here is how it looks on the page:



Getting Regular

Next we need to enforce the character requirements for the User ID and Password fields. For these we will use `RegularExpressionValidator` controls. Regular expressions can be very powerful in concisely expressing checks for this sort of information, as well as ZIP codes, phone

numbers, and e-mail addresses.

Here is how we specify the restrictions on User ID:

```
<td>
  <input type=text runat=server id=txtName>
  <asp:RegularExpressionValidator runat=server
    ControlToValidate="txtName"
    ErrorMessage="ID must be 6-10 letters."
    ValidationExpression="[a-zA-Z]{6,10}" />
</td>
```

Note that in this case we did not specify any inner content within the tag. The inner text is equivalent to the Text property of the control. If it is blank, the error message will be displayed where the control is positioned, as well as appearing in the summary.

The password checks can be done with the following two validators. If you use more than one, they must all match before the input is considered valid.

```
<asp:RegularExpressionValidator runat=server display=dynamic
  ControlToValidate="txtPWord"
  ErrorMessage="Password must contain one of @$%&*/."
  ValidationExpression=".*[@$%&*/].*" />
<asp:RegularExpressionValidator runat=server display=dynamic
  ControlToValidate="txtPWord"
  ErrorMessage="Password must be 4-12 nonblank characters."
  ValidationExpression="^[^\s]{4,12}" />
```

Here is the form in action with the expressions:

There were errors on the page:

- ID must be 6-10 letters.
- Password must contain one of @\$%&*/.
- Re-enter password is required.

Please enter a User ID and Password:

User ID: ID must be 6-10 letters.

Password: Password must contain one of @\$%&*/.

* Re-enter Password:

Comparing Apples and Apples

We need to make sure the password re-entry field matches the password. We will use the CompareValidator to do this, since it is capable of working with two input controls at a time:

```
<asp:CompareValidator runat=server
  ControlToValidate=txtRePWord
  ControlToCompare=txtPWord
  ErrorMessage="Passwords do not match." />
```

By default, CompareValidator does a simple string match comparison. If required, it can do more complex comparisons involving dates and numbers.

Custom Fit

The final thing we need to check is that the name is not already taken in our hypothetical site. This is going to require looking at some data on the server. To simulate this, I will create a dummy function in server-side code that checks that the first character is not an "a." The following Visual Basic function is defined on the page:

```

<%@ Page Language="vb" %>
<script runat=server>

public sub CheckID(source as Object, args as ServerValidateEventArgs)
    args.IsValid = args.Value.substring(0, 1).tolower() <> "a"
end sub

</script>

```

To call this function, I will add a CustomValidator, which is designed to call developer code to perform its check. Here is the declaration:

```

<asp:CustomValidator runat=server
    controltovalidate="txtName"
    errormessage="ID is already in use."
    OnServerValidate="CheckID" />

```

On a client with script, all the other validators do their checks on the client first before allowing the submit to take place. If there are errors detected by server-only checks like this one, page will be sent back to the user indicating those errors.

The Finale

Now, the only thing left is to make use of the nicely validated data. All you need to do is check the IsValid property of Page to work out if you can proceed to update your database. Here is how your submit handler might look:

```

public sub OnSubmit(source as Object, e as EventArgs)
    if Page.IsValid then
        ' Now we can perform our transaction.
    end if
end sub

```

As you can see, this handler will allow your data entry pages to consist of code that is specific to your application instead of being full of code to deal with mundane input checking.

Here is some more of the final page in action:

There were errors on the page:

- ID is already in use.
- Passwords do not match.

Please enter a User ID and Password:

User ID:	<input type="text" value="alksjdflkd"/>	ID is already in use.
Password:	<input type="password" value="P@ssw0rd"/>	
Re-enter Password:	<input type="password" value="P@w"/>	Passwords do not match.

Sample Code

```

<%@ Page Language="vb" %>
<script runat=server>

public sub CheckID(source as Object, args as ServerValidateEventArgs)
    args.IsValid = args.Value.substring(0, 1).tolower() <> "a"
end sub

public sub OnSubmit(source as Object, e as EventArgs)
    if Page.IsValid then

```

```

        ' Now we can perform our transaction.
    end if
end sub

</script>
<html>
<head>
<title>Validation Sample</title>
</head>
<body>

<form runat=server>
<asp:ValidationSummary runat=server headertext="There were errors on the page:" />

<p>Please enter a User ID and Password:</p>
<table>
  <tr>
    <td>
      <asp:RequiredFieldValidator runat=server
        controltovalidate=txtName
        errormessage="User ID is required.">*
      </asp:RequiredFieldValidator>
    </td>
    <td>User ID:</td>
    <td>
      <input type=text runat=server id=txtName>
      <asp:RegularExpressionValidator runat=server display=dynamic
        controltovalidate="txtName"
        errormessage="ID must be 6-10 letters."
        validationexpression="[a-zA-Z]{6,10}" />
      <asp:CustomValidator runat=server
        controltovalidate="txtName"
        errormessage="ID is already in use."
        OnServerValidate="CheckID" />
    </td>
  </tr>
  <tr>
    <td>
      <asp:RequiredFieldValidator runat=server
        controltovalidate=txtPWord
        errormessage="Password is required.">*
      </asp:RequiredFieldValidator>
    </td>
    <td>Password:</td>
    <td>
      <input type=password runat=server id=txtPWord>
      <asp:RegularExpressionValidator runat=server display=dynamic
        controltovalidate="txtPWord"
        errormessage="Password must contain one of @#$$%^&*./."
        validationexpression=".*[@#$$%^&*./].*" />
      <asp:RegularExpressionValidator runat=server display=dynamic
        controltovalidate="txtPWord"
        errormessage="Password must be 4-12 nonblank characters."
        validationexpression="^[^s]{4,12}" />
    </td>
  </tr>
  <tr>
    <td>
      <asp:RequiredFieldValidator runat=server
        controltovalidate=txtRePWord
        errormessage="Re-enter password is required.">*
      </asp:RequiredFieldValidator>
    </td>
    <td>Re-enter Password:</td>
    <td>
      <input type=password runat=server id=txtRePWord>
      <asp:CompareValidator runat=server
        controltovalidate=txtRePWord

```

```
controltocompare=txtPWord
errorMessage="Passwords do not match." />
</td>
</tr>
</table><br>
<input type=submit runat=server id=cmdSubmit value=Submit onserverclick=OnSubmit>
</form>
</body>
</html>
```

© Microsoft Corporation. All rights reserved.

Thank you for your feedback

Dev centers

Windows

Windows Phone

Office

Windows Azure

Visual Studio

More...

Learning resources

[Microsoft Virtual Academy](#)

[Channel 9](#)

[Interoperability Bridges](#)

[MSDN Magazine](#)

Programs

[BizSpark \(for startups\)](#)

[DreamSpark](#)

[Faculty Connection](#)

[Microsoft Student](#)

Community

[Forums](#)

[Blogs](#)

[Codeplex](#)

Support

[Self support](#)

[Other support options](#)

Thank you for your feedback

United States (English)

[Newsletter](#)

[Privacy & cookies](#)

[Terms of Use](#)

[Trademarks](#)

© 2014 Microsoft