How To: Use Regular Expressions to Constrain Input in ASP.NET

Retired Content
This content is outdated and is no longer being maintained. It is provided as a courtesy for individuals who are still using these technologies. This page may contain URLs that were valid when originally published, but now link to sites or pages that no longer exist.

patterns & practices Developer Center
J.D. Meier, Alex Mackman, Blaine Wastell, Prashant Bansode, Andy Wigley

Microsoft Corporation

May 2005

Applies To

- ASP.NET version 1.0
- ASP.NET version 1.1
- ASP.NET version 2.0

Summary

This How To shows how you can use regular expressions within ASP.NET applications to constrain untrusted input. Regular expressions are a good way to validate text fields such as names, addresses, phone numbers, and other user information. You can use them to constrain input, apply formatting rules, and check lengths. To validate input captured with server controls, you can use the **RegularExpressionValidator** control. To validate other forms of input, such as query strings, cookies, and HTML control input, you can use the **System.Text.RegularExpressions.Regex** class.

This How To shows how you can use regular expressions within ASP.NET applications to constrain untrusted input.

Contents

Objectives

- Use regular expressions to constrain input, apply format rules, and check lengths.
- Use the ASP.NET **RegularExpressionValidator** control to constrain and validate input.
- Use the **Regex** class to constrain and validate input.
- Learn common regular expressions that can be used to constrain input.

Overview

If you make unfounded assumptions about the type, length, format, or range of input, your application is unlikely to be robust. Input validation can become a security issue if an attacker discovers that you have made unfounded assumptions. The attacker can then supply carefully crafted input that compromises your application by attempting SQL injection, cross-site scripting, and other injection attacks. To avoid such vulnerability, you should validate text fields (such as names, addresses, tax identification numbers, and so on) and use regular expressions to do the following:

- Constrain the acceptable range of input characters.
- Apply formatting rules. For example, pattern-based fields, such as tax identification numbers, ZIP Codes, or postal codes, require specific patterns of input characters.
- Check lengths.

Regular expression support is available to ASP.NET applications through the **RegularExpressionValidator** control and the **Regex** class in the **System.Text.RegularExpressions** namespace.

Using a RegularExpressionValidator Control

If you capture input by using server controls, you can use the **RegularExpressionValidator** control to validate that input. You can use regular expressions to restrict the range of valid characters, to strip unwanted characters, and to perform length and format checks. You can constrain the input format by defining patterns that the input must match.

To validate a server control's input using a RegularExpressionValidator

1. Add a **RegularExpressionValidator** control to your page.
2. Set the **ControlToValidate** property to indicate which control to validate.
3. Set the **ValidationExpression** property to an appropriate regular expression.
4. Set the **ErrorMessage** property to define the message to display if the validation fails.

The following example shows a **RegularExpressionValidator** control used to validate a name field.

```
<%@ language="C#" %>
<form id="form1" runat="server">
    <asp:TextBox ID="txtName" runat="server"/>
    <asp:Button ID="btnSubmit" runat="server" Text="Submit" />
    <asp:RegularExpressionValidator ID="regexpName" runat="server"
                                    ErrorMessage="This expression does not validate."
                                    ControlToValidate="txtName"
                                    ValidationExpression="^[a-zA-Z'.\s]{1,40}$" />
</form>
```

The regular expression used in the preceding code example constrains an input name field to alphabetic characters (lowercase and uppercase), space characters, the single quotation mark (or apostrophe) for names such as O'Dell, and the period or dot character. In addition, the field length is constrained to 40 characters.

Using ^ and $

Enclosing the expression in the caret (**^**) and dollar sign (**$)**markers ensures that the expression consists of the desired content and nothing else. A **^** matches the position at the beginning of the input string and a **$** matches the position at the end of the input string. If you omit these markers, an attacker could affix malicious input to the beginning or end of valid content and bypass your filter.

Using the Regex Class

If you are not using server controls (which means you cannot use the validation controls) or if you need to validate input from sources other than form fields, such as query string parameters or cookies, you can use the **Regex** class within the **System.Text.RegularExpressions** namespace.

To use the Regex class

1. Add a **using** statement to reference the **System.Text.RegularExpressions** namespace.
2. Call the **IsMatch** method of the **Regex** class, as shown in the following example.

```
// Instance method:
Regex reg = new Regex(@"^[a-zA-Z'.]{1,40}$");
Response.Write(reg.IsMatch(txtName.Text));

// Static method:
if (!Regex.IsMatch(txtName.Text,
                   @"^[a-zA-Z'.]{1,40}$"))
{
  // Name does not match schema
}
```

For performance reasons, you should use the static **IsMatch** method where possible to avoid unnecessary object creation.

The following example shows how to use a regular expression to validate a name input through a regular client-side HTML control.

```
<%@ Page Language="C#" %>

<html xmlns="http://www.w3.org/1999/xhtml" >
  <body>
    <form id="form1" method="post" action="HtmlControls.aspx">
        Name:
        <input name="txtName" type="text" />
        <input name="submitBtn" type="Submit" value="Submit"/>
    </form>
  </body>
</html>

<script runat="server">

  void Page_Load(object sender, EventArgs e)
```

```
  {
    if (Request.RequestType == "POST")
    {
      string name = Request.Form["txtName"];
      if (name.Length > 0)
      {
        if (System.Text.RegularExpressions.Regex.IsMatch(name,
                                        "^[a-zA-Z'.]{1,40}$"))
          Response.Write("Valid name");
        else
          Response.Write("Invalid name");
      }
    }
  }

</script>
```

Use Regular Expression Comments

Regular expressions are much easier to understand if you use the following syntax and comment each component of the expression by using a number sign (**#**). To enable comments, you must also specify **RegexOptions.IgnorePatternWhitespace**, which means that non-escaped white space is ignored.

```
Regex regex = new Regex(@"
                        ^           # anchor at the start
                      (?=.*\d)      # must contain at least one numeric character
                      (?=.*[a-z])   # must contain one lowercase character
                      (?=.*[A-Z])   # must contain one uppercase character
                      .{8,10}       # From 8 to 10 characters in length
                      \s            # allows a space
                      $             # anchor at the end",
                      RegexOptions.IgnorePatternWhitespace);
```

Common Regular Expressions

Some common regular expressions are shown in Table 1.

Table 1. Common Regular Expressions

Field	Expression	Format Samples	Description
Name	^[a-zA-Z''-'\s]{1,40}$	John Doe O'Dell	Validates a name. Allows up to 40 uppercase and lowercase characters and a few special characters that are common to some names. You can modify this list.
Social Security Number	^\d{3}-\d{2}-\d{4}$	111-11-1111	Validates the format, type, and length of the supplied input field.

			The input must consist of 3 numeric characters followed by a dash, then 2 numeric characters followed by a dash, and then 4 numeric characters.
Phone Number	^[01]?[- .]?(\([2-9]\d{2}\)|[2-9]\d{2})[- .]?\d{3}[- .]?\d{4}$	(425) 555-0123 425-555-0123 425 555 0123 1-425-555-0123	Validates a U.S. phone number. It must consist of 3 numeric characters, optionally enclosed in parentheses, followed by a set of 3 numeric characters and then a set of 4 numeric characters.
E-mail	^(?("")(""[^""\r\n]+?""@)|(([0-9a-zA-Z]((\.(?!\.))|[-!#\$%&'*\+/=\?\^`\{\}|~\w])*)(?<=[0-9a-zA-Z])@))(?(\[)(\[(\d{1,3}\.){3}\d{1,3}\])|(([0-9a-zA-Z][-\w]*[0-9a-zA-Z]\.)+[a-zA-Z]{2,6}))$	someone@example.com	Validates an e-mail address.
URL	^(ht|f)tp(s?)\:\/\/[0-9a-zA-Z]([-.\w]*[0-9a-zA-Z])*(:(0-9)*)*(\/?)([a-zA-Z0-9\-\.\?\,\'\/\\\+&%\$#_]*)?$	http://www.microsoft.com	Validates a URL
ZIP Code	^(\d{5}-\d{4}|\d{5}|\d{9})$\|^([a-zA-Z]\d[a-zA-Z] \d[a-zA-Z]\d)$	12345	Validates a U.S. ZIP Code. The code must consist of 5 or 9 numeric characters.
Password	(?!^[0-9]*$)(?!^[a-zA-Z]*$)^([a-zA-Z0-9]{8,10})$		Validates a strong password. It must be between 8 and 10 characters, contain at least one digit and one alphabetic character, and must not contain special characters.
Non- negative integer	^\d+$	0 986	Validates that the field contains an integer greater than zero.
Currency (non-negative)	^\d+(\.\d\d)?$	1.00	Validates a positive currency amount. If there is a decimal point, it requires 2 numeric characters after the decimal point. For example, 3.00 is valid but 3.1 is not.
Currency (positive or negative)	^(-)?\d+(\.\d\d)?$	1.20	Validates for a positive or negative currency amount. If there is a decimal point, it requires 2 numeric characters after the decimal point.

Additional Resources

For more information, see the regular expression tutorial at http://www.regular-expressions.info/tutorial.html.

Feedback

Provide feedback by using either a Wiki or e-mail:

- **Wiki**. Use the Security Guidance Feedback Wiki page at http://channel9.msdn.com/wiki/securityguidancefeedback/
- **E-mail**. Send e-mail to secguide@microsoft.com.

We are particularly interested in feedback regarding the following:

- Technical issues specific to our recommendations
- Usefulness and usability issues

Technical Support

Technical support for the Microsoft products and technologies referenced in this guidance is provided by Microsoft Support Services. For support information, please visit the Microsoft Support Web site at http://support.microsoft.com.

Community and Newsgroups

Community support is provided in the forums and newsgroups:

- **MSDN Newsgroups**: http://msdn.microsoft.com/newsgroups/default.asp
- **ASP.NET Forums**: http://forums.asp.net

To get the most benefit, find the newsgroup that corresponds to your technology or problem. For example, if you have a problem with ASP.NET security features, you should use the ASP.NET Security forum.

Contributors and Reviewers

- **External Contributors and Reviewers:** Andy Eunson; Chris Ullman, Content Master; David Raphael, Rudolph Araujo, Foundstone Professional Services; Manoranjan M. Paul
- **Microsoft Consulting Services and PSS Contributors and Reviewers:** Adam Semel, Nobuyuki Akama, Tom Christian, Wade Mascia
- **Microsoft Product Group Contributors and Reviewers:** Stefan Schackow, Vikas Malhotra, Microsoft Corporation
- **MSDN Contributors and Reviewers:** Kent Sharkey, Microsoft Corporation
- **Microsoft IT Contributors and Reviewers:** Eric Rachner, Shawn Veney (ACE Team), Microsoft Corporation
- **Test team:** Larry Brader, Microsoft Corporation; Nadupalli Venkata Surya Sateesh, Sivanthapatham Shanmugasundaram, Sameer Tarey, Infosys Technologies Ltd
- **Edit team**: Nelly Delgado, Microsoft Corporation; Tina Burden McGrayne, Linda Werner & Associates Inc
- **Release Management:** Sanjeev Garg, Microsoft Corporation

Microsoft®
patterns & practices
proven practices for predictable results

Retired Content

This content is outdated and is no longer being maintained. It is provided as a courtesy for individuals who are still using these technologies. This page may contain URLs that were valid when originally published, but now link to sites or pages that no longer exist.