

4 Validation groups

ASP.NET WebForms (.ASPX pages that contain form fields) can contain only a single HTML form. This limitation in the development of HTML form rich pages is by designed and has caused a good deal of grief within the ASP.NET development community. To exacerbate the situation, the ASP.NET built-in validation mechanism provides no means of allowing two or more sub forms (logical forms within the primary outer form) to be validated (versions 1.0 and 1.1 of the framework).

The Mad! Widgets ASP.NET Validation Package overcomes these limitations with the introduction of validation groups and support for multiple forms when using non-server form fields.

Validation groups are sets of logically grouped form fields that can be validated separately from one another on both the client and the server. For example, you might have a page that has a login box on the left, a search form in the header and a registration form within the main content area. When using server form fields, all of these sub forms are physically placed within a large outer HTML form. Each sub form contains a means of submitting the form, e.g. a button or submit image. When the form is submitted, all information contained in each of the form fields is sent back to the server, not just the data from the sub form that was submitted.

4.1 Sub forms and form controls

Most developers place these logical groups of HTML (whether they contain form fields or not) into separate and distinct Web Controls. This is good practise as it keeps your code compartmentalised. Let's assume that the login sub form mentioned above is contained in a Web Control called Login.ascx as per example 4.1.

Example 4.1. The login sub form, Login.ascx.

```
<%@ Control Language="c#" AutoEventWireup="false"
Codebehind="Login.ascx.cs"
Inherits="MadWidgets.Validation.Samples.Login" %>

<div style="padding-left: 5px;">
  Login<br />
  <input type="text" id="Username" size="20" runat="server" />
  <input type="password" id="Password" size="20" runat="server" />
```

```

        <input type="submit" value="Login" runat="server"
            onserverclick="Login_Clicked" />
    </div>

```

Example 4.2. The Code Behind for the login sub form, Login.ascx.cs.

```

using System;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Text.RegularExpressions;
using MadWidgets.Validation;

namespace MadWidgets.Validation.Samples
{
    public abstract class Login : UserControl
    {
        protected HtmlInputText Username;
        protected HtmlInputText Password;

        private void Page_Load(object sender, EventArgs e)
        {
            Validator v = Validator.Current;
            v.Add(new ValidatorEntryRegex(Username, "Please enter a
                valid username", "^[a-z][a-z0-9_]*$",
                RegexOptions.IgnoreCase));

            v.Add(new ValidatorEntryRegex(Password, "The password is
                invalid", "^[a-z][a-z0-9_]*$", RegexOptions.IgnoreCase));
        }

        protected void Login_Clicked(object sender, EventArgs e)
        {
            if (Page.IsValid)
            {
                // Log the user in
            }
        }

        #region Web Form Designer generated code
            // Auto generated code
        #endregion
    }
}

```

The page containing the login form, the search form and the registration form might look like example 4.3.

Example 4.3. The page containing the three sub forms.

```

<%@ Page language="c#" AutoEventWireup="false" %>
<%@ Register TagPrefix="madwidgets" TagName="Search"
    Src="Search.ascx" %>
<%@ Register TagPrefix="madwidgets" TagName="Login"
    Src="Login.ascx" %>
<%@ Register TagPrefix="madwidgets" TagName="Subscribe"
    Src="Subscribe.ascx" %>
<%@ Register TagPrefix="madwidgets" TagName="Validator"
    Src="/Controls/Validation/Validator.ascx" %>

```

```

<html>
  <head>
    <title>Validation Groups</title>
  </head>
  <style type="text/css">
    body, td, input, select {font-family: Verdana,Helvetica,Arial;
      font-size: 80%;}
    .row1 {background-color: #FFA189; padding: 3px;}
    .row2 {background-color: #FFD1C9; padding: 3px;}
    .validatorError {background-color: red; padding: 3px;}
  </style>
  <body style="margin: 0px;">
    <form runat="server">

      <!-- This is the header -->
      <div style="background-color: #ECE8E3; height: 80px;
        padding: 10px;">
        <h1>Validation Groups</h1>
        <madwidgets:Search runat="server" id="SearchForm"
          validationGroup="Search" />
      </div>

      <!-- This is the left column -->
      <div style="width: 130px; height: 400px; float: left;
        background-color: #FCF8F3;">
        <madwidgets:Login runat="server" id="LoginForm"
          validationGroup="Login" />
      </div>

      <!-- This is the content area -->
      <div style="background-color: #DEFEDE; height: 400px;"
        align="center">
        <br />
        <madwidgets:Validator runat="server" id="TheValidator"
          defaulthighlightelement="&lt;tr&gt;"
          initialhighlightclass="" />
        <h3>Please fill in the form below</h3>
        <madwidgets:Subscribe runat="server" id="SubscribeForm"
          validationGroup="Subscribe" />
      </div>
    </form>
  </body>
</html>

```

Example 4.3. shows how simple it is to assign the group to the sub form. Each of the Web Controls in the example (SearchForm, LoginForm and SubscribeForm) contain an attribute *validationGroup*. The presence of this attribute on the control tells the Mad! Widgets Validator control (TheValidator in this example) that all form elements contained within the control as well as all child controls adopt that group.

So, the validation group for the login control is ‘Login,’ the validation group for the search control is ‘Search,’ and the validation group for the registration form is ‘Subscribe.’

Whichever one of these sub forms is submitted is the one that is validated. No other sub form is validated.

4.2 Validation groups on a single control

Some users may have a need to split a form into validation groups while keeping the individual form fields together. This is easily done with the Mad! Widgets ASP.NET Validation Package.

In the previous section, we learned that existence of the *validationGroup* attribute on a control that is hierarchically higher than a form element forces the form element to belong to that group. The *validationGroup* attribute can also, however, be placed on the form element itself. By placing this attribute on the form element directly, you have much greater control over which element belongs to which group. Consequently, it is possible to have more than one validation group contained within a single page or Web Control. Example 4.4 shows how this might be done.

Example 4.4. Multiple validation groups within a single control.

```
<form id="TheForm" method="POST" runat="server">
  <div class="formLabel">Group 1 Field</div>
  <div class="formInput"><input type="text" id="Group1Field1"
    size="35" runat="server" validationGroup="Group1" /></div>

  <div class="formLabel">Group 1 Field</div>
  <div class="formInput"><input type="text" id="Group1Field2"
    size="35" runat="server" validationGroup="Group1" /></div>

  <div class="formLabel">Group 2 Field</div>
  <div class="formInput"><input type="text" id="Group2Field1"
    size="35" runat="server" validationGroup="Group2" /></div>

  <div class="formLabel">Group 2 Field</div>
  <div class="formInput"><input type="text" id="Group2Field2"
    size="35" runat="server" validationGroup="Group2" /></div>

  <div class="formButtonBar">
    <input type="submit" id="Submit" value="Submit Group 1"
      class="formButton" runat="server"
      onserverclick="SubmitGroup1_Clicked"
      validationGroup="Group1" />

    <input type="submit" id="Submit" value="Submit Group 2"
      class="formButton" runat="server"
      onserverclick="SubmitGroup2_Clicked"
      validationGroup="Group2" />
  </div>
</form>
```

Note that the *validationGroup* attribute must be also placed on the submit buttons so that the Validator control knows which group to validate.

4.3 Using validation groups with non-server fields

Adding validation groups to non-server forms is possible, although a fraction more tedious than their server control counterparts. Due to the need for the Validator control on the server to know which field belongs to which group, we must explicitly add each

of the form fields to their associated groups respectively. Let's take the e-Magazine subscription form in example 3.7 with its accompanying Page_Load method from example 3.6. Near the end of the Page_Load method, we might add each of the form fields to the 'Subscribe' group as in example 4.5.

Example 4.5. Adding non-server form fields to the 'Subscribe' validation group.

```
private void Page_Load(object o, EventArgs ea)
{
    // Setup validation
    Validator v = Validator.Current;

    v.Add(new ValidatorEntryRequired("FirstName",
        "You have not supplied your first name.));

    v.Add(new ValidatorEntryRequired("LastName",
        "You have not supplied your last name.));

    v.Add(new ValidatorEntryEmail("Email", "{switch(value, ['' => 'You
        have not supplied', default => concat('', value, ''
        is not')]]} a valid email address.));

    v.Add(new ValidatorEntryDate("DateOfBirth", "{switch(value, ['' =>
        'You have not supplied', default => concat('', value, ''
        is not')]]} a valid date for the date of birth.",
        ValidatorEntryDate.Format.MMDDYYYY));

    v.Add(new ValidatorEntryRequired("EmailType", "You have not selected
        an email format type.));

    v.Add(new ValidatorEntryRequired("WhereHeardOfUs", "Please tell us
        where you heard of e-Magazine.));

    v.Add(new ValidatorEntryOr("Please tell us where you heard of
        e-Magazine by entering something in the 'Other' field.",
        new ValidatorEntryTextComparer("WhereHeardOfUs", "{OTHER}",
            ComparisonMethod.NotEqualTo),
        new ValidatorEntryRequired("WhereHeardOfUsOther"));

    // Register fields into validation group
    v.RegisterGroupForObject("FirstName", "Subscribe");
    v.RegisterGroupForObject("LastName", "Subscribe");
    v.RegisterGroupForObject("Email", "Subscribe");
    v.RegisterGroupForObject("DateOfBirth", "Subscribe");
    v.RegisterGroupForObject("EmailType", "Subscribe");
    v.RegisterGroupForObject("WhereHeardOfUs", "Subscribe");
    v.RegisterGroupForObject("WhereHeardOfUsOther", "Subscribe");

    // Force validation since there are no server forms
    // in the control tree
    if (Request.Form["Submit"] != null)
    {
        Page.Validate();
        Submit_Clicked(o, ea);
    }
}
```



Note: if you have more than one logical form on a page and you are using non-server controls, you must use validation groups, regardless whether you are containing them within a single outer HTML form or in multiple HTML forms.