

7 Creating custom validator summaries

The validation summary that comes shipped with the Mad! Widgets ASP.NET Validation Package is a very simple ASPX file (a basic UserControl) that extends the Mad! Widgets Validator control. This file contains code that iterates over the entire collection of ValidatorEntry objects, checking for validity status and adding the text message of any that are not in a valid state to an HTML list.

It's a very simple piece of code and can consequently be easily duplicated or rewritten to suit any specific formatting or styling requirement.

The validation summary does not need to extend the Validator control contained in the package. It can be a stand alone UserControl that references the Validator control separately. It may be necessary to do this if more than one validation summary is required on a page.

If the validation summary does not extend the Validator control supplied with the package, the Validator control must be placed somewhere in the control hierarchy still – in a location where it is valid to place JavaScript code. Note that when the control writes JavaScript code to the page, it also writes out the `<script>` tags that surround it. So the Validator control can be placed anywhere within the body of the page or within the `<head>` section.

7.1 A basic validation summary

Example 7.1 shows a simple validation summary control that extends the Validator control (MadWidgets.Validation.Validator). The Validator control is an object that manages the ValidatorEntry objects and instructs them to perform validation. By extending the Validator control, the validation summary has access to all public and protected methods available in the control. Consequently, the validation summary does not need to use a reference to the Validator control to access its methods.

The example shown simply iterates over the collection of ValidatorEntry objects held by the Validator control and writes the error message defined respectively to the HTML output. It also provides a 'show me' link where appropriate.

The example uses the `Count` property and the indexer method `this[i]` to perform the iteration. `this[i]` returns the ValidatorEntry object at the index position `i`.

The JavaScript function `showValidationError(id)` sends the cursor to the field that identified by 'id.' This can be either the object id as returned by the `GetObjectId` method or the `ValidatorEntry` object instance id which is written to the client if the object is an instance of `ValidatorEntryClientServer`.

Note that in example 7.1, it is assumed that the CSS styles are defined elsewhere.

Example 7.1. A basic validation summary.

```
<%@ Control Language="c#" AutoEventWireup="false"
    Inherits="MadWidgets.Validation.Validator" %>
<%@ Import namespace="MadWidgets.Validation"%>

<%
    if (!IsValid && IsPostBack)
    {
%>

<table cellpadding="0" cellspacing="0" border="0" class="validatorTable">
    <tr>
        <td class="validatorTitle" colspan="2"><%=SummaryTitle%></td>
    </tr>
<%
    int iError = 0;
    for (int i = 0; i < Count; i++)
    {
        ValidatorEntry ve = this[i];
        if (!ve.IsValid)
        {
            string sErrorMessage = ve.ProcessMessage();
            sErrorMessage = sErrorMessage.Replace("\r\n", "<br /><br />");
            string sStatusMessage =
                Server.HtmlEncode(sErrorMessage.Substring(0,
                    sErrorMessage.Length > 85 ? 85 : sErrorMessage.Length));
            sStatusMessage = sStatusMessage.Replace("\"", "&#" +
                ((int)'\\"').ToString("000") + ";");
            sStatusMessage = sStatusMessage.Replace("'", "&#" +
                ((int)'\'').ToString("000") + ";");
%>
<tr class="validatorRow"<%= (iError % 2) + 1%>">
    <td class="validatorEntry" width="1%"><%= iError + 1%>.&nbsp;</td>
    <td class="validatorEntry" width="99%">
        <%= sErrorMessage%>
<%
        if (ClientSideCodeEnabled && (!(ve is
            ValidatorEntryClientServer) ||
            ((ValidatorEntryClientServer)ve).ClientSideCodeEnabled))
        {
            object oFirst = ve.FirstObject;
            if (oFirst != null)
            {
%>
                [ <a href="javascript:showValidationError('<%= ve is
                    ValidatorEntryClientServer ? ve.Id :
                    Validator.GetObjectId(oFirst)%>');" class="validatorLink"
                    onmouseover="window.status='Validation Failure'; return true;"
                    title="Validation Failure">show me</a> ]
<%
            }
        }
%>
    </td>
</tr>
</table>
```

```

        </tr>
    <%
        iError++;
    }
}
%>
</table>
<br />
<br />
<%
}
%>

```

7.2 Validation summaries for specific validation groups

Example 7.1 is a sample validation summary that will display all error messages regardless of the validation group that they belong to. In some circumstances, you may not want this behaviour. For example, you may have three validation groups on your page – a Login group, a Search group and a Subscribe group – where the associated forms provide fields respectively for logging in, searching the site and subscribing to a service of the site. Rather than display errors from each of these groups in a single validation summary, you can assign a validation summary to each group and place it in a location that suits the group. You may want the error messages associated with the login form to appear directly under the login form and the errors associated with the subscribe form to appear directly above that form. Using the Mad! Widgets ASP.NET Validation Package, this is very easy to achieve.

Example 7.2. A simple validator summary that handles only a specific validation group.

```

<%@ Control Language="c#" AutoEventWireup="false" %>
<%@ Import namespace="MadWidgets.Validation"%>
<%@ Import namespace="System.Collections"%>

<%
    Validator v = Validator.Current;
    if (v.WasGroupPostedForObject(this) && !IsValid && IsPostBack)
    {
%>

<table cellpadding="0" cellspacing="0" border="0" class="validatorTable">
    <tr>
        <td class="validatorTitle" colspan="2"><%=SummaryTitle%></td>
    </tr>
<%
    int iError = 0;
    IList l = v.GetEntriesForPostedGroup();
    for (int i = 0; i < l.Count; i++)
    {
        ValidatorEntry ve = (ValidatorEntry)l[i];
        if (!ve.IsValid)
        {
            string sErrorMessage = ve.ProcessMessage();
            sErrorMessage = sErrorMessage.Replace("\r\n", "<br /><br />");
            string sStatusMessage =
                Server.HtmlEncode(sErrorMessage.Substring(0,
                    sErrorMessage.Length > 85 ? 85 : sErrorMessage.Length));

```

```

        sStatusMessage = sStatusMessage.Replace("\'", "&#" +
            ((int)''').ToString("000") + ";");
        sStatusMessage = sStatusMessage.Replace("'", "&#" +
            ((int)'\').ToString("000") + ";");
    %>
    <tr class="validatorRow"<%= (iError % 2) + 1 %>">
        <td class="validatorEntry" width="1%"><%= iError + 1 %>. &nbsp;</td>
        <td class="validatorEntry" width="99%">
            <%= sErrorMessage %>
    <%
        if (v.ClientSideCodeEnabled && (!(ve is
            ValidatorEntryClientServer) ||
            ((ValidatorEntryClientServer)ve).ClientSideCodeEnabled))
        {
            object oFirst = ve.FirstObject;
            if (oFirst != null)
            {
    %>
                [ <a href="javascript:showValidationError('<%= ve is
                    ValidatorEntryClientServer ? ve.Id :
                    Validator.GetObjectId(oFirst) %>');" class="validatorLink"
                    onmouseover="window.status='Validation Failure'; return true;"
                    title="Validation Failure">show me</a> ]
    <%
            }
        }
    %>
    </td>
</tr>
<%
    iError++;
}
}
%>
</table>
<br />
<br />
<%
}
%>

```

Example 7.2 shows the same validation summary as example 7.1 except that it has been restricted to a specific validation group and it no longer extends the Validator control. The primary difference in the two examples is the `WasGroupPostedForObject(this)` call and the `GetEntriesForPostedGroup()` call, which returns true only if the form was posted back for the group associated with the validation summary control. Example 7.3 shows how the validation summary might be included in the page and associated with the group “Login”.

Example 7.3. Assigning a validation group to a validation summary.

```

<mycompany:MyValidationSummary runat="server" id="valsum"
    validationGroup="Login" />

```