

## 3 Getting started

To begin integrating the Mad! Widgets ASP.NET Validators into your pages, you will need a page with a form in it (a WebForm). If you have existing .NET Framework validators in your WebForm, remove all of them so that you are left with a bare page or control with nothing more than form controls in it.

### 3.1 Creating the WebForm

Throughout this section we will be developing a simple WebForm containing a few form fields that are used for an electronic magazine subscription.

We begin with the bare WebForm. Example 3.1 shows the WebForm used for the electronic magazine subscription. Note that example 3.1 is a C# example. A VB.NET equivalent of this example is shipped with the package under the directory */samples*.

*Example 3.1. e-Magazine subscription.*

```
<%@ Page Language="C#" %>
<%@ Import namespace="MadWidgets.Validation.Samples" %>
<%@ Import namespace="System.Globalization" %>

<script runat="server">

void Submit_Clicked(object o, EventArgs ea)
{
    if (Page.IsValid)
    {
        try
        {
            // Subscribe the user to the magazine
            Subscription s = new Subscription();
            s.FirstName = FirstName.Text;
            s.LastName = LastName.Text;
            s.Email = Email.Text;
            s.DateOfBirth = DateTime.Parse(DateOfBirth.Text,
                CultureInfo.CurrentCulture);
            s.Type = EmailType.Items[EmailType.SelectedIndex].Value;
            s.WhereHeardOfUs = WhereHeardOfUs.Items[
                WhereHeardOfUs.SelectedIndex].Value;
            s.WhereHeardOfUsOther = WhereHeardOfUsOther.Text;
            s.Save();
        }
    }
}
```



```

        <asp:ListItem value="TEXT">Textual
            Format</asp:ListItem>
        <asp:ListItem value="HTML">HTML
            Format</asp:ListItem>
        <asp:ListItem value="RTF">Rich Text
            Format</asp:ListItem>
    </asp:DropDownList>
</td>
</tr>
<tr class="row2">
    <td class="label" valign="baseline">Where did you hear
        about e-Magazine?</td>
    <td class="field">
        <asp:DropDownList id="WhereHeardOfUs" runat="server">
            <asp:ListItem value="">-- Please select where you
                heard of e-Magazine --</asp:ListItem>
            <asp:ListItem value="FRIEND">Through a
                friend</asp:ListItem>
            <asp:ListItem value="ENGINE">Through a search
                engine</asp:ListItem>
            <asp:ListItem value="LINK">By clicking a
                hyperlink</asp:ListItem>
            <asp:ListItem value="PAPER">In a paper
                magazine</asp:ListItem>
            <asp:ListItem value="OTHER">Other (see
                below)</asp:ListItem>
        </asp:DropDownList><br />
        <asp:TextBox id="WhereHeardOfUsOther"
            width="180" runat="server" />
    </td>
</tr>
<tr>
    <td colspan="2" align="right"><br /><asp:Button
        id="Submit" runat="server"
        onclick="Submit_Clicked"
        text=" Submit " /></td>
</tr>
</table>
</form>
</asp:Panel>

<asp:Panel id="Page2" runat="server" visible="false">
    <p>Thank you for subscribing to e-Magazine!</p>
</asp:Panel>
</body>
</html>

```

This WebForm contains just 7 fields:

- First name.
- Last name.
- Email address.
- Date of birth.
- The type of email format desired.
- A survey question about where the user heard of e-Magazine.

It also contains a Submit handler called Submit\_Clicked which creates the subscription and stores it. Once stored successfully a second panel is made visible and the first form hidden. This second panel merely states “Thank you for subscribing to e-Magazine!”

Note that for the sake of this demonstration it is not necessary to discuss or understand the Subscription class used in the Submit handler.

## 3.2 Adding Mad! Widgets validation

The WebForm in example 3.1 contains no validation except for the try/catch block in the Submit handler. It's now time to add Mad! Widgets validation to the page.

Add the following two lines to the top of the page:

```
<%@ Register TagPrefix="madwidgets" TagName="Validator"
    Src="/Controls/Validation/Validator.ascx" %>
```

```
<%@ Import Namespace="MadWidgets.Validation" %>
```

The Register tag tells the CLR where to find Mad! Widgets Validation Summary control while the Import tag tells the CLR where to find the validator classes that come with the package. Note that the path to the Validation Summary control can be any path of your choosing. You will have decided where to place this control (Validator.ascx) when you installed the package on your web server (see [Setting up the web.config file](#)). You may have also created a custom Validation Summary control (see *Creating a custom Validation Summary*).

Add the following line in the location that you would like the Validation Summary to appear.

```
<madwidgets:Validator runat="server" id="TheValidator" />
```

This is the Validation Summary control. The Validation Summary control that comes with the package doubles as the Validator control because it hierarchically extends the Validator control class. In some circumstances you may wish to split these. More information is provided later in this document about doing this.

Note that the Validation Summary is a control that displays error message in relation to the form fields that failed validation. It serves no other purpose. On the other hand, the Mad! Widgets Validator control is the control that actually performs the validation in the background. The Validator control is located in the package MadWidgets.Validation.

With the Validator control now present on the WebForm, you can start adding validation to individual fields.

Add a Page\_Load method to the script section (the `<script runat="server">` section at the top of the page) as in example 3.2.

Example 3.2. The `Page_Load` method of the ASPX page.

```
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 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)));
}
```

If you do not have `AutoEventWireup="true"` in your `@Page` tag, you will need to add it as follows:

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

This tells the CLR to call the local script's `Page_Load` method upon load.



**Note:** if you are placing the validation code in your Code Behind class, *you should not* add this attribute to the tag unless you have a `Page_Load` method in the page script. Similarly, if `AutoEventWireup` is true by default on your site (as determined by your `web.config` file), you will need to explicitly set it to false in the `@Page` tag if you do not have a `Page_Load` method on the page script. This prevents the Code Behind's `Page_Load` from getting called twice.

Take another look at the `Page_Load` that we've just added to the script section of the page. You will see seven validators. The last validator, `ValidatorEntryOr`, is a more complex one and we'll discuss that shortly. Firstly, let's go through each validator to find out what it's doing.

1. The first validator is a `ValidatorEntryRequired` validator.

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

This will produce the message “You have not supplied your first name” if nothing is entered in the `FirstName` field.

2. The second validator is also a `ValidatorEntryRequired` validator.

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

This produces the message “You have not supplied your last name” if nothing is entered in the `LastName` field.

3. The third validator is a `ValidatorEntryEmail` validator.

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

This produces one of two messages. If the user has not entered anything in the `Email` field, the message will be “You have not supplied a valid email address.” If the user has entered something in the field but it is not a valid email address (e.g. `support@domain`), the message will be “‘support@domain’ is not a valid email address.”

As you can see, the message contains the inline message function *switch*. You will learn more about inline message functions later in this document (see [Inline message blocks](#)).

4. The fourth validator is a `ValidatorEntryDate` validator.

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

Like the third validator, this validator produces one of two messages. If the user has not entered anything in the `DateOfBirth` field, the message will be “You have not supplied a valid date of birth.” If the user has entered something in the field but it is not a valid date (e.g. `13/13/1968`), the message will be “‘13/13/1968’ is not a valid date of birth.”

5. The fifth validator is another a `ValidatorEntryRequired` validator.

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

This produces the message “You have not selected an email format” if nothing is selected from the EmailType drop-down list.

6. The sixth validator is yet another a ValidatorEntryRequired validator.

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

This produces the message “Please tell us where you heard of e-Magazine” if nothing is selected from the WhereHeardOfUs drop-down list.

7. The final validator is a ValidatorEntryOr validator.

```
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)));
```

This more complex validator is used in this instance to ensure that if the user has selected ‘Other’ from the drop-down list, that they have entered something in the ‘Other’ text field as well. If the user does not select ‘Other’ from the drop-down list, then this validator will not produce an error message, regardless whether something is entered in the ‘Other’ text field or not. If the user selected ‘Other’ from the drop-down list and they did not enter something in the ‘Other’ text field, then this validator would produce the message “Please tell us where you heard of e-Magazine by entering something in the 'Other' field.”

The logic of this validator is equivalent to the C# statement:

```
WhereHeardOfUs.SelectedValue != "OTHER" ||  
WhereHeardOfUsOther.Text != ""
```

Or in VB.NET:

```
WhereHeardOfUs.SelectedValue <> "OTHER" Or  
WhereHeardOfUsOther.Text <> ""
```

As you can imagine the ValidatorEntryOr and the ValidatorEntryAnd validators are very powerful. They can be used to create, if desired, quite complex hierarchical validation logic. They can be nested and be used with any other type of validator class.

### **3.3 Validation styles and highlight elements**

If you have turned on ticks in the Mad! Widgets Validation Package on your website, you will see crosses and ticks appear in your form fields as they progress from an invalid to a valid state. This works by manipulating the CSS styles on the form fields directly behind the scenes as the user types or clicks in the form and changes the validation state of the form accordingly.

Ticks and crosses are not the only type of visual indication that a form field has passed or not passed validation. In addition to ticks and crosses, you can manipulate the style of the surrounding HTML elements in the form. Every form field can have a nominated 'highlight element.' A highlight element is an HTML element that the form field is associated with and that can be used to present a visual indication of the validation state of the field. Highlight elements are assigned to the form field when the page first loads and cannot be reassigned to the form field thereafter.

You can assign the highlight element to the form field in a number of ways. The most likely way to assign it is through the web.config file. Most sites on the internet use a common look and feel throughout the site, including all the forms on the site. Invariably the HTML structure of the each of the forms on the site is identical throughout. You can take advantage of this by adding an entry in the web.config that is used to locate the highlight element for every form field on your site without having to add information on a page level. The entry in the web.config file is 'validatorDefaultHighlightElement.'

In [Setting up the web.config file](#), you were shown how to set up this entry as well as all other entries associated with the Mad! Widgets Validation Package. This entry takes a 'highlight element signature.' The highlight element signature is a conditional statement that the client-side code uses to locate a form field's highlight element. The signature comes in the form:

```
<tagName[property=value]>
```

This condition states:

```
Starting from the form field element, navigate up the HTML
hierarchy until an element with the tag name tagName is found
that possesses a property called property whose value is equal
to value.
```

When the client-side code finds the element that matches the condition, it is registered as the 'highlight element' for the form field.

The highlight element signature comes in a simpler form as follows:

```
<tagName>
```

In this case the condition is:

```
Starting from the form field element, navigate up the HTML
hierarchy until an element with the tag name tagName is found.
```

The simpler form makes no check on the properties of the element that is found.



**Note:** more than one form field can have the same highlight element. For example, if the highlight element on our example form was <tr>, then the form fields WhereHeardOfUs and WhereHeardOfUsOther would share the same highlight element.



**Note:** the property noted in the highlight element signature refers to a property on the HTML DOM element – it is not an HTML attribute. For example, if you wanted to match a row with the CSS class equal to ‘row2,’ then property would need to be ‘className’ and not ‘class’ because ‘className’ is the DOM property for the HTML attribute ‘class.’ In other words the highlight element signature would look like this:

```
<tr[className=row2]>
```

So, what does the client-side code actually do with the highlight element?

When the page containing the form first loads on the browser and all highlight elements are found and recorded, the client-side code also remembers the CSS class names associated with the elements. If, upon page load, any of the validators are in an invalid state on the browser, the client-side code will swap the CSS class on the associated highlight element with CSS class registered with the Validator control as the ‘InitialHighlightClass’ or if this was not set or is null, with the CSS class registered with the Validator control as the HighlightClass. If any form field is not in an invalid state (i.e. it is valid), then the original CSS class on the highlight element remains untouched.

Once the page has loaded, the InitialHighlightClass is never used again. Thereafter, if a form field is invalid, the highlight element will be assigned the HighlightClass.

In both cases, once the validation passes for a given field or set of fields, the CSS class originally found on the highlight element is reassigned to the element.

It’s now time to add a highlight element signature to our form and to prepare the CSS classes.

Assuming you have set up the web.config file as per the examples provided in the section [Setting up the web.config file](#), add the class validatorError to the CSS style block on the page as follows:

```
<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>
```

Next modify the control tag as follows:

```
<madwidgets:Validator runat="server" id="TheValidator"
  defaulthighlightelement="&lt;tr&gt;" initialhighlightclass="" />
```

In this example, you will notice two changes, the addition of the attribute 'defaulthighlightelement' and the addition of the attribute 'initialhighlightclass.' The former of these is the highlight element signature and it states merely '<tr>.' Note that it is escaped. You should escape this value in order to maintain well-formed HTML. The second attribute, initialhighlightclass, blanks out the validatorInitialHighlightClass property set up in the web.config file, i.e.:

```
<add key="validatorInitialHighlightClass" value="validatorInitial" />
```

In our example, we do not want to alter the style on the highlight elements when the page loads, only when the page is being edited. Blanking out the Validator control property 'InitialHighlightClass' at the page level prevents the client-side code from swapping the styles on the highlight elements during page load.

Example 3.3 now shows the final WebForm complete with all validation logic and style information.

*Example 3.3. e-Magazine subscription complete with validation logic.*

```
<%@ Page Language="C#" AutoEventWireup="true" %>
<%@ Register TagPrefix="madwidgets" TagName="Validator"
    Src="/Controls/Validation/Validator.ascx" %>
<%@ Import Namespace="MadWidgets.Validation" %>
<%@ Import namespace="MadWidgets.Validation.Samples" %>
<%@ Import namespace="System.Globalization" %>

<script runat="server">

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));
```

```

}

void Submit_Clicked(object o, EventArgs ea)
{
    if (Page.IsValid)
    {
        try
        {
            // Subscribe the user to the magazine
            Subscription s = new Subscription();
            s.FirstName = FirstName.Text;
            s.LastName = LastName.Text;
            s.Email = Email.Text;
            s.DateOfBirth = DateTime.Parse(DateOfBirth.Text,
                CultureInfo.CurrentCulture);
            s.Type = EmailType.Items[EmailType.SelectedIndex].Value;
            s.WhereHeardOfUs = WhereHeardOfUs.Items[
                WhereHeardOfUs.SelectedIndex].Value;
            s.WhereHeardOfUsOther = WhereHeardOfUsOther.Text;
            s.Save();

            // Swap panels upon success
            Page1.Visible = false;
            Page2.Visible = true;
        }
        catch (Exception e)
        {
            Validator.Current.AddError("An error occurred in the form");
        }
    }
}

</script>

<html>
<head>
<title>Subscribe to e-Magazine</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>
<h1>Subscribe to e-Magazine</h1>

<asp:Panel id="Page1" runat="server" visible="true">
<p>To receive the monthly e-Magazine FREE,
please fill in the form below.</p>

<p>Note that all fields are <b>required</b>.

<madwidgets:Validator runat="server" id="TheValidator"
    defaulthighlightelement="&lt;tr&gt;" initialhighlightclass="" />

<form name="TheForm" id="TheForm" method="POST" runat="server">
<table cellpadding="0" cellspacing="1" border="0">
<tr class="row1">
<td class="label" valign="baseline">First name</td>
<td class="field"><asp:TextBox id="FirstName"
    width="180" runat="server" /></td>
</tr>

```

```

<tr class="row2">
  <td class="label" valign="baseline">Last name</td>
  <td class="field"><asp:TextBox id="LastName"
    width="180" runat="server" /></td>
</tr>
<tr class="row1">
  <td class="label" valign="baseline">Email address</td>
  <td class="field"><asp:TextBox id="Email"
    width="180" runat="server" /></td>
</tr>
<tr class="row2">
  <td class="label" valign="baseline">Date of birth</td>
  <td class="field"><asp:TextBox id="DateOfBirth"
    width="180" runat="server" /></td>
</tr>
<tr class="row1">
  <td class="label" valign="baseline">Type of email</td>
  <td class="field">
    <asp:DropDownList id="EmailType" runat="server">
      <asp:ListItem value="">-- Please select a
        type --</asp:ListItem>
      <asp:ListItem value="TEXT">Textual
        Format</asp:ListItem>
      <asp:ListItem value="HTML">HTML
        Format</asp:ListItem>
      <asp:ListItem value="RTF">Rich Text
        Format</asp:ListItem>
    </asp:DropDownList>
  </td>
</tr>
<tr class="row2">
  <td class="label" valign="baseline">Where did you hear
    about e-Magazine?</td>
  <td class="field">
    <asp:DropDownList id="WhereHeardOfUs" runat="server">
      <asp:ListItem value="">-- Please select where you
        heard of e-Magazine --</asp:ListItem>
      <asp:ListItem value="FRIEND">Through a
        friend</asp:ListItem>
      <asp:ListItem value="ENGINE">Through a search
        engine</asp:ListItem>
      <asp:ListItem value="LINK">By clicking a
        hyperlink</asp:ListItem>
      <asp:ListItem value="PAPER">In a paper
        magazine</asp:ListItem>
      <asp:ListItem value="OTHER">Other (see
        below)</asp:ListItem>
    </asp:DropDownList><br />
    <asp:TextBox id="WhereHeardOfUsOther"
      width="180" runat="server" />
  </td>
</tr>
<tr>
  <td colspan="2" align="right"><br /><asp:Button
    id="Submit" runat="server"
    onclick="Submit_Clicked"
    text=" Submit " /></td>
</tr>
</table>
</form>
</asp:Panel>

<asp:Panel id="Page2" runat="server" visible="false">
  <p>Thank you for subscribing to e-Magazine!</p>

```

```

        </asp:Panel>
    </body>
</html>

```

### 3.4 Placing validation in your Code Behind class

In ASP.NET you are not restricted to referencing your form fields in the ASPX page itself, you can also reference the form field controls through a code behind class. Some people prefer to add validation code in the code behind class because it keeps the backend code (C#, VB.NET, etc.) separate from the display code (XML, HTML, etc.).

With the Mad! Widgets ASP.NET Validation Package, this is very easy to do and is little different from adding validation in the ASPX page itself. In the following example, we've created a copy of the e-Magazine subscription form, placing the Page\_Load and Submit\_Clicked methods in the code behind.

*Example 3.4. The code behind class GettingStarted\_CB. This can be found in the file GettingStarted\_CB.aspx.cs.*

```

using System;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Globalization;
using MadWidgets.Validation;

namespace MadWidgets.Validation.Samples
{
    public class GettingStarted_CB : Page
    {
        protected TextBox    FirstName;
        protected TextBox    LastName;
        protected TextBox    Email;
        protected TextBox    DateOfBirth;
        protected DropDownList EmailType;
        protected DropDownList WhereHeardOfUs;
        protected TextBox    WhereHeardOfUsOther;
        protected Panel      Page1;
        protected Panel      Page2;

        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)));
}

protected void Submit_Clicked(object o, EventArgs ea)
{
    if (Page.IsValid)
    {
        try
        {
            // Subscribe the user to the magazine
            Subscription s = new Subscription();
            s.FirstName = FirstName.Text;
            s.LastName = LastName.Text;
            s.Email = Email.Text;
            s.DateOfBirth = DateTime.Parse(DateOfBirth.Text,
                CultureInfo.CurrentCulture);
            s.Type = EmailType.Items[EmailType.SelectedIndex].Value;
            s.WhereHeardOfUs = WhereHeardOfUs.Items[
                WhereHeardOfUs.SelectedIndex].Value;
            s.WhereHeardOfUsOther = WhereHeardOfUsOther.Text;
            s.Save();

            // Swap panels upon success
            Page1.Visible = false;
            Page2.Visible = true;
        }
        catch (FormatException e)
        {
            Response.Write("The date of birth is in an invalid format");
        }
        catch (Exception e)
        {
            Response.Write("An error occurred in the form");
        }
    }
}

#region Web Form Designer generated code
override protected void OnInit(EventArgs e)
{
    //
    // CODEGEN: This call is required by the ASP.NET Web
    // Form Designer.
    //
    InitializeComponent();
    base.OnInit(e);
}

/// <summary>
/// Required method for Designer support - do not modify

```

```

    /// the contents of this method with the code editor.
    /// </summary>
    private void InitializeComponent()
    {
        this.Load += new System.EventHandler(this.Page_Load);
    }
}
#endregion
}
}

```

*Example 3.5. The ASPX page. This can be found in the file GettingStarted\_CB.aspx.*

```

<%@ Page Language="c#" AutoEventWireup="false"
CodeBehind="GettingStarted_CB.aspx.cs"
Inherits="MadWidgets.Validation.Samples.GettingStarted_CB" %>
<%@ Register TagPrefix="madwidgets" TagName="Validator"
Src="/Controls/Validation/Validator.ascx" %>

<html>
  <head>
    <title>Subscribe to e-Magazine</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>
    <h1>Subscribe to e-Magazine</h1>

    <asp:Panel id="Page1" runat="server" visible="true">
      <p>To receive the monthly e-Magazine FREE,
      please fill in the form below.</p>

      <p>Note that all fields are <b>required</b>.</p>

      <madwidgets:Validator runat="server" id="TheValidator"
        defaulthighlightelement="&lt;tr&gt;" initialhighlightclass="" />

      <form name="TheForm" id="TheForm" method="POST" runat="server">
        <table cellpadding="0" cellspacing="1" border="0">
          <tr class="row1">
            <td class="label" valign="baseline">First name</td>
            <td class="field"><asp:TextBox id="FirstName"
              width="180" runat="server" /></td>
          </tr>
          <tr class="row2">
            <td class="label" valign="baseline">Last name</td>
            <td class="field"><asp:TextBox id="LastName"
              width="180" runat="server" /></td>
          </tr>
          <tr class="row1">
            <td class="label" valign="baseline">Email address</td>
            <td class="field"><asp:TextBox id="Email"
              width="180" runat="server" /></td>
          </tr>
          <tr class="row2">
            <td class="label" valign="baseline">Date of birth</td>
            <td class="field"><asp:TextBox id="DateOfBirth"
              width="180" runat="server" /></td>
          </tr>
        </table>
      </form>
    </asp:Panel>
  </body>
</html>

```

```

</tr>
<tr class="row1">
  <td class="label" valign="baseline">Type of email</td>
  <td class="field">
    <asp:DropDownList id="EmailType" runat="server">
      <asp:ListItem value="">-- Please select a
        type --</asp:ListItem>
      <asp:ListItem value="TEXT">Textual
        Format</asp:ListItem>
      <asp:ListItem value="HTML">HTML
        Format</asp:ListItem>
      <asp:ListItem value="RTF">Rich Text
        Format</asp:ListItem>
    </asp:DropDownList>
  </td>
</tr>
<tr class="row2">
  <td class="label" valign="baseline">Where did you hear
    about e-Magazine?</td>
  <td class="field">
    <asp:DropDownList id="WhereHeardOfUs" runat="server">
      <asp:ListItem value="">-- Please select where you
        heard of e-Magazine --</asp:ListItem>
      <asp:ListItem value="FRIEND">Through a
        friend</asp:ListItem>
      <asp:ListItem value="ENGINE">Through a search
        engine</asp:ListItem>
      <asp:ListItem value="LINK">By clicking a
        hyperlink</asp:ListItem>
      <asp:ListItem value="PAPER">In a paper
        magazine</asp:ListItem>
      <asp:ListItem value="OTHER">Other (see
        below)</asp:ListItem>
    </asp:DropDownList><br />
    <asp:TextBox id="WhereHeardOfUsOther"
      width="180" runat="server" />
  </td>
</tr>
<tr>
  <td colspan="2" align="right"><br /><asp:Button
    id="Submit" runat="server"
    onclick="Submit_Clicked"
    text=" Submit " /></td>
</tr>
</table>
</form>
</asp:Panel>

<asp:Panel id="Page2" runat="server" visible="false">
  <p>Thank you for subscribing to e-Magazine!</p>
</asp:Panel>
</body>
</html>

```

As you can see, all that we've done here is removed the `<script runat="server">` from the ASPX page and placed it in the code behind class. We've also removed the Import statements and placed them in the code behind file.

This example demonstrates how easy it is to support modern development patterns such as Model-View-Controller with the validation package.

### 3.5 Using validation on non-server form fields

The Mad! Widgets ASP.NET Validation Package supports both server forms and non-server forms. Server forms are forms containing fields that are instantiated as controls on the server. Non-server forms are the traditional kind of HTML forms that, as far as the server is concerned, is simply raw HTML that gets sent to the browser.

The Mad! Widgets ASP.NET Validation Package can support both types of forms because, behind the scenes, the two are represented in very much the same manner. There are a few differences, but these mainly centre around inline message blocks. And even then the differences are subtle.

Setting up validation on non-server form fields is almost identical to setting up validation on server form field controls. The primary difference is that instead of passing Control objects to the validation constructors, you instead pass the names of the form fields to the constructors.

In example 3.6 we've taken the Page\_Load method from the previous examples in this chapter and converted it to use non-server form field names instead.

*Example 3.6. Setting up validation for non-server form fields.*

```
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")));

    // Force validation since there are no server forms
    // in the control tree
    if (Request.Form["Submit"] != null)
    {
```

```

    Page.Validate();
    Submit_Clicked(o, ea);
}
}
}

```

*Example 3.7 The HTML form associated with example 3.6.*

```

<form name="TheForm" id="TheForm" method="POST">
  <table cellpadding="0" cellspacing="1" border="0">
    <tr class="row1">
      <td class="label" valign="baseline">First name</td>
      <td class="field"><input type="text" name="FirstName"
        width="180" /></td>
    </tr>
    <tr class="row2">
      <td class="label" valign="baseline">Last name</td>
      <td class="field"><input type="text" name="LastName"
        width="180" /></td>
    </tr>
    <tr class="row1">
      <td class="label" valign="baseline">Email address</td>
      <td class="field"><input type="text" name="Email"
        width="180" /></td>
    </tr>
    <tr class="row2">
      <td class="label" valign="baseline">Date of birth</td>
      <td class="field"><input type="text" name="DateOfBirth"
        width="180" /></td>
    </tr>
    <tr class="row1">
      <td class="label" valign="baseline">Type of email</td>
      <td class="field">
        <select name="EmailType">
          <option value="">-- Please select a
            type --</option>
          <option value="TEXT">Textual
            Format</option>
          <option value="HTML">HTML
            Format</option>
          <option value="RTF">Rich Text
            Format</option>
        </select>
      </td>
    </tr>
    <tr class="row2">
      <td class="label" valign="baseline">Where did you hear
        about e-Magazine?</td>
      <td class="field">
        <select name="WhereHeardOfUs">
          <option value="">-- Please select where you
            heard of e-Magazine --</option>
          <option value="FRIEND">Through a
            friend</option>
          <option value="ENGINE">Through a search
            engine</option>
          <option value="LINK">By clicking a
            hyperlink</option>
          <option value="PAPER">In a paper
            magazine</option>
          <option value="OTHER">Other (see
            below)</option>
        </select><br />
        <input type="text" name="WhereHeardOfUsOther"

```

```

        width="180" />
    </td>
</tr>
<tr>
    <td colspan="2" align="right"><br /><input
        name="Submit" type="submit"
        text=" Submit " /></td>
</tr>
</table>
</form>

```

There are a few things that you may have noticed between example 3.6 and 3.7. Firstly, the Control objects in the constructors of the ValidatorEntry classes in example 3.6 have been replaced with each of the names from the respective form field from example 3.7. Secondly, due to the lack of any server form control on the page, the CLR will no longer try to validate the page on the server, so we have to do this explicitly by calling *Page.Validate()*. Finally, you will notice that the server controls embedded in the HTML in the previous examples have been replaced with their traditional HTML counterparts, specifically, INPUT and SELECT tags. Note that the name takes over where the id was used on the server control.