

5 Inline message blocks

In [Getting Started](#), you learned how to add Mad! Widgets validation to your WebForms. Some of the example validators introduced in that chapter used messages that contained embedded blocks of code. These blocks of code are *inline message blocks*. They are used to incorporate dynamic content into your error messages, allowing them to become more descriptive and helpful to the end user than their static counterparts.

Inline message blocks allow you to be more creative with your error messages. You can use the value entered in a text field, the value or set of values selected from a drop-down list, a radio button value, checkbox values and more as a source for the dynamic content that you use within the messages.

Say, for example, you have a date field on which there is an age restriction of 18 years old. If the date entered indicates that the user is 16 years old, you can produce a message saying, “The birth date you entered is 5 March 93, you are too young to register here.” Example 3 demonstrates how this can be done using the `ValidatorEntryDateRange` validator.

Example 5.1. Inline message block using `ValidatorEntryDateRange`.

```
v.Add(new ValidatorEntryDateRange(DateOfBirth, "The birth  
date you entered is {value}, you are too young to register  
here.", ValidatorEntryDate.Format.MMDDYYYY, DateTime.MinValue,  
DateTime.Now.AddYears(-18)));
```

This validator is tied to a form control called `DateOfBirth`. It states that if the date entered is less than today’s date minus 18 years, then all is OK. However, if the date entered is greater than today’s date minus 18 years, then produce the error message.

In our example, we have assumed that the user literally entered ‘5 May 93,’ but the user might just as well have entered 3/5/93. The problem with this type of date format is that it is ambiguous. Did the user mean 3 May 93 or 5 March 93? This is where the enumeration `ValidatorEntryDate.Format` comes in handy. In our example, we are using the enumeration constant `ValidatorEntryDate.Format.MMDDYYYY`. This constant tells the validator that if a date is entered in a format that might be considered ambiguous, that the first set of digits should be considered the month value, the second the day of month value and the third the year value. So in our example, the value 3/5/93 would be interpreted as 5 March 1993.

The next thing we might want to do to our dynamic value is ensure, regardless of the format that the date is entered in, that it is displayed in the message in a fixed format. Modify the message as per example 5.2:

Example 5.2. Inline message block using ValidatorEntryDateRange and the functions cdate and formatDate.

```
v.Add(new ValidatorEntryDateRange(DateOfBirth, "The birth
    date you entered is {formatDate(cdate(value, format),
    'd MMMM yyyy')}, you are too young to register here.",
    ValidatorEntryDate.Format.MMDDYYYY, DateTime.MinValue,
    DateTime.Now.AddYears(-18)));
```

The inline message block now performs the following actions on the value. First, using the function, *cdate*, it converts the string value entered to a date type (*Date* on the browser, *DateTime* on the server). Next it passes the date value into *formatDate* which formats the value into the format ‘d MMM yyyy,’ which in our example would look like ‘3 Mar 1993’ regardless whether the user entered the value as 5 March 93 or 3/5/93. Note that when the conversion of the value entered is performed using the function *cdate*, it takes as its second parameter *format*. This property is used on the validator classes *ValidatorEntryDate*, *ValidatorEntryDateRange* and *ValidatorEntryDateComparer* and refers to the expected input format determined by the *ValidatorEntryDate.Format* constant. If this parameter is not supplied, *cdate* will make its best guess of the input value.

The question springs to mind, what happens if the user doesn’t enter anything or if the value entered is not a date? In this example, *cdate* will pass through the invalid value to *formatDate* and *formatDate* will pass it through to the message. So whether the value is blank or an invalid date, then it will still be embedded in the message. You can prevent this behaviour with the *switch* function.

Once again modify the message as per example 5.3.

Example 5.3. Getting fancy with the switch function.

```
v.Add(new ValidatorEntryDateRange(DateOfBirth, "{switch(value,
    ['' => 'Please enter your birth date', default =>
    switch(isdate(value, format), [true => concat('The birth date
    you entered is ', formatDate(cdate(value, format),
    'd MMMM yyyy'), ', you are too young to register here'),
    default => 'The date of birth entered is invalid'])]}",
    ValidatorEntryDate.Format.MMDDYYYY,
    DateTime.MinValue, DateTime.Now.AddYears(-18)));
```

Let’s break this elaborate concoction down:

```
switch(value,
[
    '' => 'Please enter your birth date',
    default => switch(isdate(value, 'MMDDYYYY'),
    [
        true => concat('The birth date you entered is ',
            formatDate(cdate(value, 'MMDDYYYY'), 'd MMMM yyyy'),
            ', you are too young to register here'),
        default => 'The date of birth entered is invalid'
```

```
    1)
1)
```

The inline message function, *switch*, is similar to a C# *switch* or a VB.NET *Select Case*. It takes a value and tries to match it against a key within an array. If it can match the value with a key, it takes the value of the key (the right side) and returns it. If it can't find a match, it looks for the keyword *default*, if it exists, and returns whatever the value of *default* is.

In example 5, we have two switches, one nested inside the other. The first switch checks the value entered, if it is blank (''), it returns the string, 'Please enter your birth date.' In every other case, i.e. if the user entered something, the second switch is called. This switch first calls the function, *isdate*, which checks whether the value entered is a valid date. If it is, it returns the concatenation of three strings, 'The birth date you entered is' plus the result of *formatDate* plus ', you are too young to register here.' If the value entered is not a valid date, it returns the default value, which is a string stating, 'The date of birth entered is invalid.'

In example 5.3, we demonstrated the use of an *associative array* within the inline message block. This is an array that contains a set of elements that are associated by *key* and *value*. Within the inline message block you can create associative arrays using the following format:

```
[key1 => value1, key2 => value2, ... keyn => valuen]
```

Some HTML form elements also form associative arrays. These are: radio button lists, select lists and checkbox lists. A dropdown list (*select list* in HTML speak) for example might look like the following .NET control:

```
<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>
```

The associations in this example are:

```
" " => -- Please select where you heard of e-Magazine --
"FRIEND" => Through a friend
"ENGINE" => Through a search engine
"LINK" => By clicking a hyperlink
"PAPER" => In a paper magazine
"OTHER" => Other (see below)
```

So this dropdown list is equivalent to the inline message block associative array:

```
[
  '' => '-- Please select where you heard of e-Magazine --',
  'FRIEND' => 'Through a friend',
  'ENGINE' => 'Through a search engine',
  'LINK' => 'By clicking a hyperlink',
```

```

    'PAPER' => 'In a paper magazine',
    'OTHER' => 'Other (see below)'
]

```

Fortunately you don't have to reproduce the dropdown list array for use in the inline message blocks explicitly. You can access the dropdown list array through the property *allvalues*.

Example 5.4. Using allvalues.

```
switch(value, allvalues)
```

Example 5.4 shows how you might extract the text (display) value from a dropdown list. For example, if the item 'By clicking a hyperlink' were selected from the example above, the inline message property, *value*, would be equal to 'LINK.' The switch function in example 5.4 would use this value as a key to search through the associative array, *allvalues*, and return the text value selected.

There's a simpler way to obtain the text value selected from a dropdown list though, and that is by using the property *textvalue*. So example 5.4 is equivalent to the property *textvalue*.

Table 5.1 describes each of the global properties available in inline message blocks. These can be used with all validators. In addition to these, each validator typically has a small set of properties that are pertinent to its purpose.

Table 5.1. Inline message properties.

Property	Type	Form Field Types and Values
value	string	All field types except multiple select lists (i.e. DropDownList set to multiple) and checkbox list controls (CheckBoxList).
value	associative array	Multiple select lists and CheckBoxLists. This contains the elements that are currently selected in the list extracted from the <i>allvalues</i> array.
allvalues	associative array	For list-type fields, the array holds all of the items in the list in their key => value pairs. For non list-type fields, this property is always null.
selectedvalues	associative array	For list-type fields, the array holds the items that are selected in the list in their key => value pairs. These items are extracted from the <i>allvalues</i> array. For non list-type fields, this property is always null.

Property	Type	Form Field Types and Values
textvalue	string	<p data-bbox="767 237 1334 342">Multiple select lists and CheckBoxLists – a concatenation of all keys in the equivalent associative array delimited by commas.</p> <p data-bbox="767 383 1283 454">All other list types, the text value of the currently selected item.</p> <p data-bbox="767 495 1353 524">All other types, the value entered in the field.</p>



Note: allvalues and selectedvalues cannot be used with non-server controls. Instead, when using non-server controls, you will need to place explicit associative arrays in your inline message blocks if you require this functionality.