

Custom Validators & Formatters

Shlomy Gantz
President, BlueBrick Inc.

Agenda

- Creating a Custom Formatter
- Creating a Custom Validator
- Creating Custom Tween effect

About me

```
<mx:Title text="President, BlueBrick Inc.">
```

```
<mx:Experience software="16">
```

```
<mx:Experience flex = "4">
```

```
<mx:Model id="Titles">
```

```
<root>
```

```
  <titles>
```

```
    <teaching> Adobe Certified Instructor </teaching>
```

```
    <adobe> Adobe Community Expert </adobe>
```

```
    <community> Manager, NYFLEX user group </community>
```

```
    <ego> Speaker, Author. </ego>
```

```
  </titles>
```

```
</root>
```

```
</mx:Model>
```

```
<mx:Mom status="Very Proud">
```

About Formatters

- Formatters are used to format data into strings.
- Typically used in text fields/labels
- Formatters perform a one-way conversion of raw data to a formatted string.
- Flex includes standard formatters:
 - Currency
 - Dates
 - Numbers
 - Phone numbers
 - ZIP codes.

Standard Formatters

- To use a standard formatter
 - Declare a formatter in your MXML code
 - Call the formatter's `format()` method within the curly braces (`{ }`) syntax for binding data, and specify the value to be formatted as a parameter to the `format()` method.

Standard Formatters

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <!-- Declare a PhoneFormatter-->
  <mx:PhoneFormatter
    id="usPhone"
    formatString="(###) ###-####" />

  <!-- Declare the input control.-->
  <mx:Label text="Enter 10 digit phone number (#####):" />
  <mx:TextInput id="myPhone" />

  <!-- display the formatted data.-->
  <mx:TextArea text="{usPhone.format(myPhone.text)}" />

</mx:Application>
```

Formatter Class

- All Flex formatters are subclasses of the [mx.formatters.Formatter](#) class.
- The custom Formatter class declares a [format\(\)](#) method

Creating a custom formatter

- Create a class that extends
 - [mx.formatters.Formatter](#)
 - Any of the standard formatters
 - Must contain a `format()` method

Creating a custom Formatter

ExampleFormatter.as

```
package customFormatter
{
import mx.formatters.Formatter;
  public class ExampleFormatter extends Formatter
  {

    // Constructor
    public function ExampleFormatter() {
      // Call base class constructor.
      super();
    }

    ...

  }
}
```

Creating a custom Formatter

ExampleFormatter.as

```
...  
// Override format().  
override public function format(valueIn:Object):String {  
  
    var newValue:String = valueIn.toString();  
  
    // 1. If the value is long, format the string.  
    if(newValue.length > 20) {  
        return newValue.substr(0,20)+'...';  
    }  
  
    // 2. If the value is short, just return the string.  
    return newValue;  
    }  
}
```

Calling a Custom Formatter

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
xmlns:custFmt="customFormatter.*">

<!-- Declare the formatter -->
<custFmt:ExampleFormatter id="shortFormat" />

<mx:Panel title="{shortFormat.format(TextIn.text)}">
  <!-- Text input for string to be formatted. -->
  <mx:TextArea id="TextIn" />
</mx:Panel>

</mx:Application>
```

Walkthrough 1

- Creating a custom formatter

Handling Errors in Formatter

- Usually, when an error occurs, an empty string is returned
- Description of the error should be written to the formatter's error property.
- The error property is inherited from the [Formatter](#) superclass.

Handling Errors in Formatter

```
...  
override public function format(valueIn:Object):String {  
  
    if( valueIn.length == 0)  
        { error=" Invalid value";  
          return ""  
        }  
  
...  
}
```

Handling Formatter Errors

- **Test for an empty string in the result returned by the formatter.**
- **Check the error property to find the cause.**

Handling Formatter Errors

```
....  
private function formatWithError(value:Object):String {  
    var formatted:String = myFormatter.format(value);  
    if (formatted == "") {  
        if (myFormatter.error != null ) {  
            if (myFormatter.error == "Invalid value")  
                {formatted = ": The value is not valid.";}  
            else  
                {formatted = ": The formatString is not valid."; }  
        }  
    }  
    return formatted;  
}  
....
```

Walkthrough 2

- Handling error in Formatter

Using SwitchSymbolFormatter

- Parses the format string and replaces each placeholder character with a number from the input string (numeric)
- The default placeholder character is the number sign (#).
- `SwitchSymbolFormatter.formatValue()`

Walkthrough 3

- Using SwitchSymbolFormatter

SwitchSymbolFormatter

- Using a different placeholder

```
var myFormatter = new SwitchSymbolFormatter("$");
```

Validator

INNOVATIVE SOLUTIONS, PROVEN RESULTS

About Validators

- *validator* is used to ensure the values in the fields of an object meet certain criteria
- Typically used in text input controls

Standard Validators

- Flex includes standard validators:
 - Credit Cards
 - Phone numbers
 - ZIP codes.
 - Currency
 - Dates
 - Email Addresses
 - Numbers
 - RegEx
 - SSN
 - Strings

Using a Standard Validator

- Use the source and property properties

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">  
  
  <mx:Form id="contactForm">  
    <mx:FormItem id="email" label="Email">  
      <mx:TextInput id="txtEmail"/>  
    </mx:FormItem>  
    <mx:Button label="Retrieve Password" />  
  </mx:Form>  
  
  <mx:EmailValidator id="emailV" source="{txtEmail}" property="text"/>  
  
</mx:Application>
```

Using a Standard Validator

- To use a standard validator
 1. Declare a validator in your MXML code
 2. Call the validator's validate().

`validate(value:Object = null, supressEvents:Boolean = false):ValidationResultEvent`

- `value`
 - Null use the source and property
 - Not null - validate object
- `supressEvents` -
 - false, dispatch the valid/invalid event
 - true, do not dispatch events.

Using a Standard Validator

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Script>
    <![CDATA[ ... ]]>
  </mx:Script>

  <mx:StringValidator id="firstNameV" required="true"/>
  <mx:Form>
    <mx:FormItem label="First name">
      <mx:TextInput id="txtFirst"/>
    </mx:FormItem>
    <mx:Button label="Submit" click="validateForm();"/>
  </mx:Form>
</mx:Application>
```

Using a Standard Validator

```
<mx:Script>
  <![CDATA[
    import mx.events.ValidationResultEvent;
    import mx.controls.Alert;

    private var vResult:ValidationResultEvent;

    private function validateForm():void      {
      vResult = firstNameV.validate(txtFirst.text);
      if (vResult.type==ValidationResultEvent.INVALID){
        Alert.show("Problem with First Name Field");
      }
      else{Alert.show("Validation OK");      }
    }
  ]>
</mx:Script>
```

Validator Class

- All Flex validators are subclasses of the [mx.validators.Validator](#) class.
- Use the `Validator.validate()` method to invoke a validator

Creating a Custom Validator

- **override of the protected `Validator.doValidation()` method**
 - Method takes a single argument, `value`, of type `Object`,
 - Returns an `Array of ValidationResult` objects for each field the validator fails on.
 - For fields that pass the validation, you omit the `ValidationResult` object.

Creating a Custom Validator

- **Flex creates ValidationResult objects for fields that validate successfully.**
- **The base Validator class implements the logic to handle required fields by using the required property.**
- **In the doValidation() method of your validator class, you typically call the base class's doValidation() method to perform the verification for a required field.**
- **The remainder of the doValidation() method contains your custom validation logic.**

ValidationResult

Properties

isError: Boolean

- Indicates whether or not the result is an error. Set this property to true.

subField: String

- Name of the subfield associated with the ValidationResult object.

errorCode: String

- Contains an error code. (Example: “ERR-01”)

errorMessage: String

- Contains the error message. (Example: “Cannot contain numbers”)

Walkthrough 4

- Customizing a simple validator

Custom Validator

- You can validate an Object that may contain several items all at once.

Walkthrough 5

- Creating Custom Validator for Multiple Fields

Thank you

Shlomy Gantz

shlomy@bluebrick.com

<http://www.shlomygantz.com>