Creating Client And Server-Side Form Validation Using The Validator Toolkit For ASP.NET MVC

Jürgen Bäurle

February 2008

Parago Media GmbH & Co. KG

Introduction

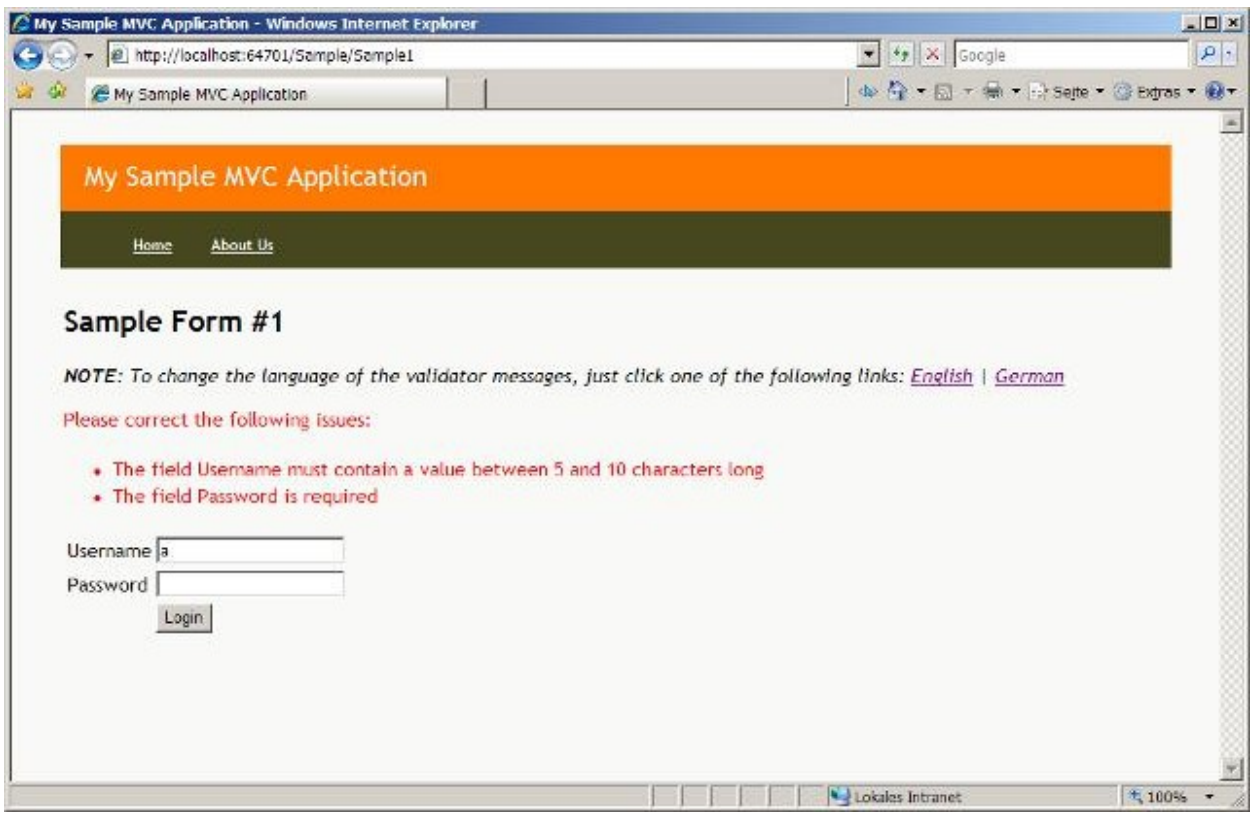
Microsoft just released the first preview of ASP.NET MVC framework and it's Microsoft's way to develop web applications based on the model-view-controller architecture. It will not replace the classic ASP.NET WebForms model. For more information of the new framework, please see the Scott Guthrie's blog message:

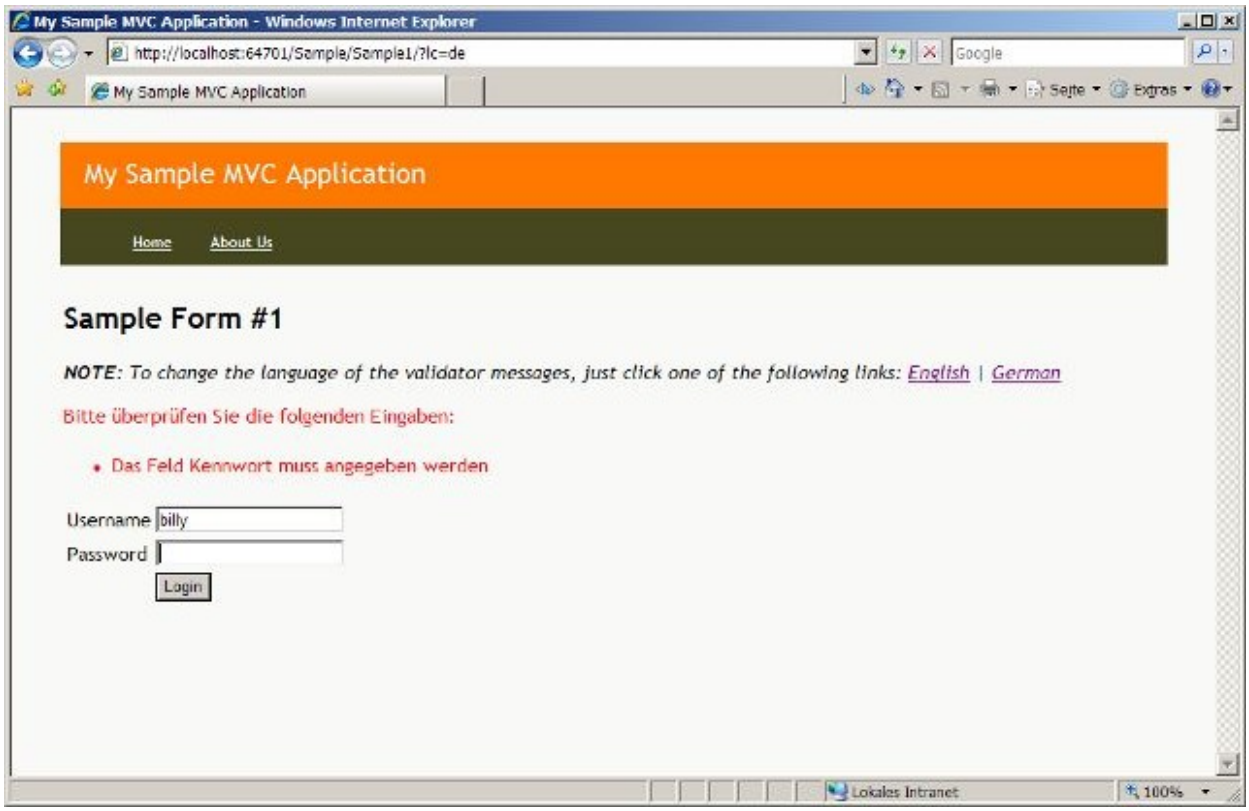
http://weblogs.asp.net/scottgu/archive/2007/10/14/asp-net-mvc-framework.aspx

Since there is just a first preview version available, the final feature list is not complete. At this time of point there is no built-in solution to validate a HTML form on the client and server-side using a standardized system. The *Validator Toolkit for ASP.NET MVC* offers a way to do form validation on the client and server-side using *validation sets*. The toolkit is a new project on Microsoft's Open Source Community site CodePlex.com:

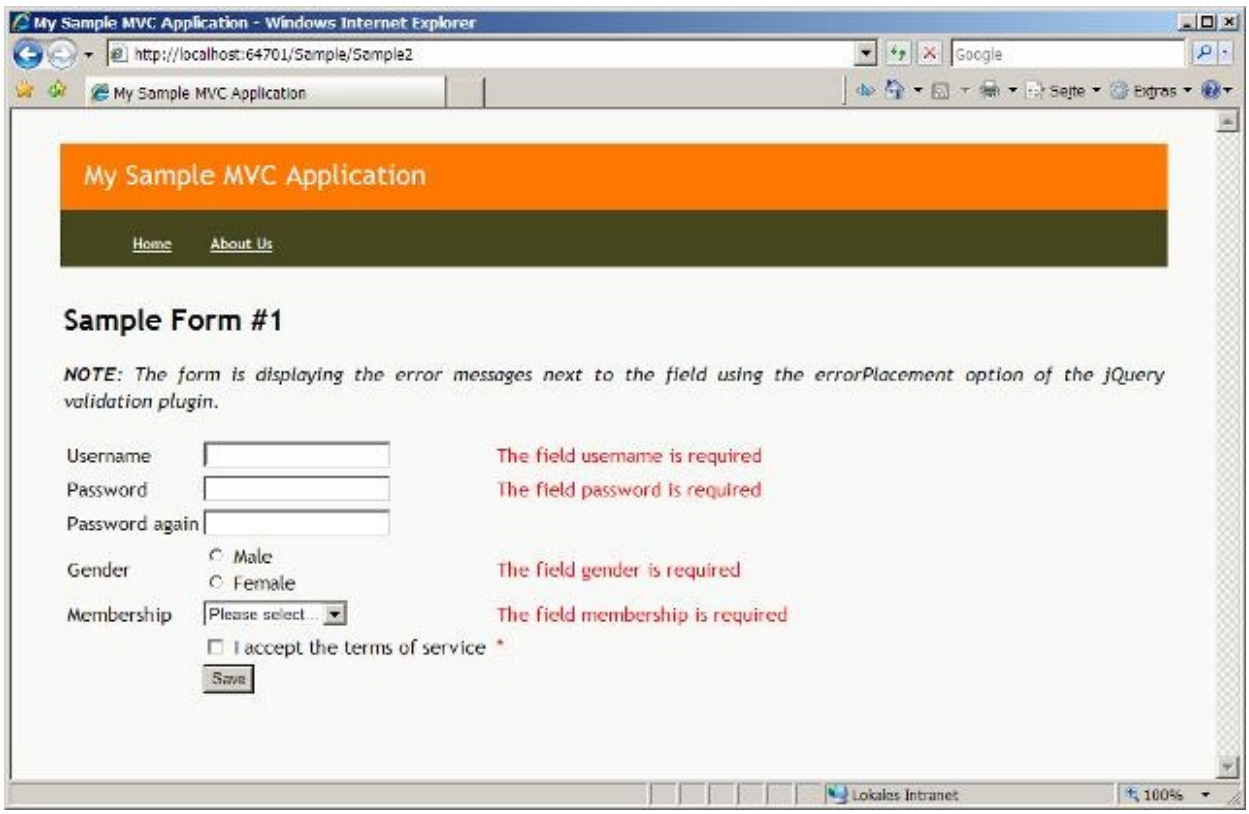
http://www.codeplex.com/MvcValidatorToolkit

The screenshots below gives you an idea how the toolkit will work:





Here another screenshot how the error messages can be displayed:



Now, let's start!

Solution

As mentioned above, the toolkit is using validation sets as a crucial element. Validation sets are special classes that derive from the ValidationSet base class and define all validation rules (validators) for a HTML form. The toolkit is generating client-side JavaScript code based on the defined validators during the view rendering.

The client-side uses the very powerful jQuery JavaScript library in conjunction with the jQuery validation plug-in to fulfil the final task of client validation. For more information regarding jQuery and the plug-in, please take a look at the following links:

http://www.jquery.com
http://plugins.jquery.com/project/validate

The validation plug-in used by the toolkit is slightly customized to support all the needed behaviours. Beside using the jQuery library for client validation, you can use it for a lot of other stuff, like animations or DOM manipulation. There are also a lot of plug-ins developed that are extending the core library. Validation sets are also used to validate forms on the server (based on the Request.Form collection).

Before we go ahead let's have a look at a sample validation set:

```
public class LoginValidationSet : ValidationSet {

   protected override ValidatorCollection GetValidators() {

      return new ValidatorCollection
      (
         new ValidateElement("username") { Required = true, MinLength = 5, MaxLength = 30 },
         new ValidateElement("password") { Required = true, MinLength = 3, MaxLength = 50 }
      );

   }
}
```

This LoginValidationSet class defines the rules for validating a simple login form by overriding the GetValidators method of the base class. The method must return a ValidatorCollection instance with all validators used to validate the HTML form later on. In this case the username field is required and the input for it must contain at least 5 characters and a maximum of 30 characters. The password field is required, too, but within the boundaries of 3 and 50 characters.

The order of the defined validators also defines the execution order of validation process. If the toolkit would use custom attributes to set validation rules instead of the method GetValidators, there is no guarantee for the validation process to validate in the same order the attributes are defined, since the Type.GetCustomAttributes method returns the attribute list in alphabetical order.

Of course, you can also write your own custom validators or you may use the ValidateScriptMethod validator which allows you to call a specific JavaScript function on the client-side and a method within the validation set class for the server-side. More on that later.

Once a validation set class is defined you attach it to the view and the HTML form processing controller action using the ValidationSet attribute, like this:

```
//
// File: LoginController.cs
//
public class LoginController : Controller
{
   [ControllerAction]
   public void Login()
   {
      RenderView("Login");
   }

   [ControllerAction]
   [ValidationSet(typeof(LoginValidationSet))]
   public void Authenticate()
   {
      if(this.ValidateForm())
         RenderView("Overview");
      else
```

```
            RenderView("Login");
      }
}

…

//
// File: Login.aspx.cs (CodeBehind)
//
[ValidationSet(typeof(LoginValidationSet))]
public partial class Login : ViewPage
{
      …
}
```

The controller action Authenticate is then calling the ValidateForm method, which uses the ValidationSet attribute to do the server-side form validation, based on the NameValueCollection Request:Form.

Within the HTML page you need to initialize script validation for the login form (loginForm) as follows:

```
<script type="text/javascript">
   $(function(){
      updateSettingsForSample1ValidationSet($('#loginForm').validate({rules:{}}));
   });
</script>
```

In the next step you define the HTML form as usual:

```
<form id="loginForm" action="/Login/Authenticate" method="post">
   Username: <input type="text" id="username" name="username" /><br />
   Password: <input type="text" id="password" name="password" /><br />
   <input type="submit" value="Login" />
</form>
```

Finally, the script that define the validation rules must be defined:

```
<% this.RenderValidationSetScripts(); %>
```

You also need to include the jQuery JavaScript library into the form or master page. See the Validator Toolkit sample site for more information. Sample site of the toolkit includes the scripts in the master page:

```
<script type="text/javascript" src="../../Content/jDate.js"></script>
<script type="text/javascript" src="../../Content/jQuery.Core.js"></script>
<script type="text/javascript" src="../../Content/jQuery.Delegate.js"></script>
<script type="text/javascript" src="../../Content/jQuery.Validation.js"></script>
```

That's basically all you need to do include form validation on the client and server-side. The next section gives you an overview of the standard validators and the usage of them.

Standard Validators

The toolkit offers a handful of standard validators out of the box. The following table gives you an overview of the provided validators:

ValidatePresence	Validates the presence of input value. Sample: `new ValidatePresence("username")`

ValidateDate	Validates input value against the given date formats. Setting the DateFormats property to a comma-separated list of date formats, allows you to check if the value is formatted in one of the specified formats. You may check for date formats like 'MM/ DD/ YYYY'. If DateFormats is not defined than the validator will use the culture information of the current thread.
	
	Sample:
	`new ValidateDate("startDate", "yyyy-mm-dd,yyyymmdd")`
	`new ValidateDate("startDate") { DateFormats = "yyyy-mm-dd,yyyymmdd" }`

ValidateMin
ValidateMax
VAlidateRange
	Validates input value either against a minimum, maximum or integer range.
	Sample:
	`new ValidateMin("step", 5)`
	`new ValidateRange("step") { Min = 1, Max = 99 }`

ValidateMinLength
ValidateMaxLength
VAlidateRangeLength
	Validates input value either against a minimum, maximum or integer range.
	Sample:
	`new ValidateMaxLength("password ", 30)`
	`new ValidateRangeLength("password") { MinLength = 5, MaxLength = 30 }`

ValidateElement	Joins multiple validators into one validator to keep things simple and allows the following to validate: Required, Min, Max, MinLength, MaxLength Sample: `new ValidateElement("username") { Required = true, MinLength = 5 }`
ValidateEqualTo	Validates input value either against another input value. This validator is useful to compare password inputs. Sample: `new ValidateEqualTo("password", "passwordAgain")`

ValidateScriptMethod	Validates input value using a custom JavaScript function and validation set class method.
	
	Sample:
	`new ValidateScriptMethod("username") { MethodName = "valUsername" }`
	`new ValidateScriptMethod("username") {`
	`MethodName = "valiUsername", Parameters = "{Opt1:2, Opt2: "AB"}" }`

There are still some validators missing, e.g. a general regular expression validator or a specific email validator.

Validation Sets

Each validation set definition class derives from the ValidationSet base class. This base class contains and offers common functionality for the validation process. Let's take a look on the sample below to explain some possibilities the validation set offers to validate complex forms:

```
public class LoginValidationSet : ValidationSet {

   string Username = "";
   string Password = "";

   protected override ValidatorCollection GetValidators() {

      return new ValidatorCollection
      (
         new ValidateElement("username") { Required = true, MinLength = 5, MaxLength = 30 },
         new ValidateElement("password") { Required = true, MinLength = 3, MaxLength = 50 },
         new ValidateScriptMethod("username", "validateUsername")
      );

   }

   protected bool ValidateUsername() {
      // DO HERE SOME VALIDATION AND RETURN RESULT AS BOOLEAN VALUE
      return true;
   }

   protected override void OnValidate()
   {
      if(Username.StartsWith("billy") && Password.StartsWith("gat"))
         throw new ValidatorException("username", "The username/password combination …");
   }
}
```

Creating non-public instance member fields of type String like the Username or Password fields, allows the base class to populate those fields with the accompanied input field values. This is a simple way to access input values without checking the underlying values collection (e.g. Request.Form).

The ValidateScriptMethod validator defines a JavaScript function (validateUsername) to call during the client-side validation. This function must be defined or included in the HTML page. The ValidationSet base class is checking during the server-side validation if the current validation set class is containing a case-insensitive method named validateUsername.

Once all validators defined with the GetValidators method are called during the validation process, the base class will call an overall method called OnValidate. By overriding this method you can do some final validation. If you want to throw an exception you need to throw the ValidatorException with the field name and message as parameters.

Using the described techniques, the possibilities of the jQuery library and the custom validators (see below) you can validate most complex forms quite efficient. The next section will describe ways to localize the messages.

Localization

It's easy to localize error messages of the toolkit using the standard folder. If the default settings are not changed, the default error message for each validator is stored in the ValidationSet.resx file (in folder the App_GlobalResources). The naming convention for the resource key is as follows: `<VALIDATORNAME>_DefaultErrorMessage`.

To change the default name of the ValidationSet.resx resource file, a derived validation set class can set the name by using the static field DefaultMessageResourceName or by adding the MessageResourceName attribute to the class. It is also possible to combine the techniques and also use more than one resource file.

The sample site within the Validator Toolkit contains usage examples of localized error messages and field names. The localization is simple and straightforward.

Custom Validators

Creating custom validators is quite simple, but requires some basic knowledge of the jQuery JavaScript library and the validation plug-in if the validator wants to support client validation. The sample site includes a custom validator called ValidateBuga, which checks the input value against the constant string "buga". Each validator derive from the Validator class, which provides a couple of virtual methods a custom validator must override:

GetClientMethodData	This method defines by returning an instance of the ValidatorMethodData class the name of the custom validator (for the jQuery validation plug-in), the JavaScript function code and the default error message.
GetClientRule	This method returns the plug-in client rule to use once the validator is defined within a validation set class.
GetClientMessage	This method returns the localized error message for a given element.
GetDefaultErrorMessageFormat	This method returns the default error message of the custom validator.
Validate	This method validates the input value for the given element. If the input is not valid the validator must call the InsertError method of the Validator base class to signal an error.

Here the source code of the ValidateBuga sample validator:

```
public class ValidateBuga : Validator {

   public ValidateBuga(string elementsToValidate)
      : base(elementsToValidate) {
   }

   public override ValidatorMethodData GetClientMethodData() {
      return new ValidatorMethodData(
         "buga",
         "function(value,element,parameters){return value=='buga';}",
         "$.format('" + ErrorMessageFormat + "')"
      );
   }

   public override string GetClientRule(string element) {
      return "buga:true";
   }
```

```
   public override string GetClientMessage(string element) {
      return string.Format("buga:'{0}'", GetLocalizedErrorMessage(element))
         .Replace("'", "\'");
   }

   protected override void Validate(string element) {
      if(Values.ContainsKey(element) == false || (Values[element] ??
         string.Empty).Trim() != "buga")
         InsertError(element);
   }

   protected override string GetDefaultErrorMessageFormat() {
      return "The field {0} must contain the value \"buga\"";
   }
}
```

Another way to use custom validation is by using the validator ValidateScriptMethod. It allows to call a JavaScript function on the client-side and a specific method of the validation set class on the server-side. The custom validation is part of the validation set class and is not usable in multiple validation sets.

Summary

The Validator Toolkit is a simple and easy way to add client-side and server-side HTML form validation to the new ASP.NET MVC framework. It is easy to extend the toolkit by adding custom validators. You may use the toolkit until Microsoft provides a solution for HTML form validation.

Contact Information

If you have any feedback or suggestions, please feel free to contact me:

Jürgen Bäurle
jbaurle@parago.de
http://www.parago.de/jbaurle

Parago Media GmbH & Co. KG
Im Wengert 3 | 71336 Waiblingen, Germany | Phone +49.7146.861803 | Internet http://www.parago.de