* 1. 

Hands-On Lab

ASP.NET MVC Custom Validation

* 1. Lab version: 1.1.0
  2. Last updated: 1/11/2011

Contents

[Overview 3](#_Toc282145876)

[Exercise 1: Adding Custom Range Validation Using Validation Attributes 4](#_Toc282145877)

[Task 1 – Adding a validation attribute class 5](#_Toc282145878)

[Task 2 – Creating the Custom Data Annotation 7](#_Toc282145879)

[Task 3 – Running the Application 8](#_Toc282145880)

[Exercise 2: Adding Validation at Client Side 9](#_Toc282145881)

[Task 1 – Creating a ModelClientValidationRule for price ranges 9](#_Toc282145882)

[Task 2 – Adding attribute validation at client side 11](#_Toc282145883)

[Task 3 – Adding a JavaScript Function to Perform Client-Side Validation 12](#_Toc282145884)

[Task 4 – Modifying the Create View to Execute Client-Side Validation 12](#_Toc282145885)

[Task 5 – Running the Application 13](#_Toc282145886)

[Exercise 3: Using IValidatableObject Custom Validation 14](#_Toc282145887)

[Task 1 – Implementing IValidatableObject Validation Method 14](#_Toc282145888)

[Task 2 – Running the Application 15](#_Toc282145889)

[Exercise 4: Using Unobtrusive jQuery at Client Side 16](#_Toc282145890)

[Task 1 – Running the Application before Enabling Unobtrusive jQuery 16](#_Toc282145891)

[Task 2 – Enabling Client Validation from Web.config 18](#_Toc282145892)

[Task 3 – Adding Unobtrusive jQuery to Master Page 18](#_Toc282145893)

[Task 4 – Running the Application Using Unobtrusive jQuery Validation 19](#_Toc282145894)

[Summary 21](#_Toc282145895)

Overview

* 1. **Note:** This Hands-on Lab assumes you have basic knowledge of **ASP.NET MVC and ASP.NET MVC validation**. If you have not used **ASP.NET MVC validation** before, we recommend you to go over **ASP.NET MVC Helpers, Forms and Validation** Hand-on Lab.

In **ASP.NET MVC Helpers, Forms and Validation** Hands-on Lab, you have been validating data from the create and edit album forms. In this Hands-on Lab, you will use custom validation logic to control price range in the server side as well as in the client.

* 1. In order to achieve that, in the first exercise you will create a custom validation property attribute and then add it to the Album model. The existing create and edit views in the StoreManager controller will process the new custom validation from the server side.
  2. In the second exercise, you will add JavaScript code and associate it to the previous custom validation, making the same views run that validation code at client side.

Then, you will implement a custom server-side validation using ASP.NET4 IValidatableObject Interface, performing an extra validation at server side that will run after attribute validation is performed.

Lastly, you will use ASP.NET MVC 3 Unobtrusive jQuery integration, executing the validation defined in the first exercise at client side with jQuery libraries.

# Objectives

* 1. In this Hands-On Lab, you will learn how to:
  + Create a custom validation attribute to extend the validation capabilities of the solution
  + Run the validation logic at client side to improve user experience
  + Implement ASP.NET IValidatableObject to extend validation scope
  + Integrate Unobtrusive jQuery library to enhance client side validation

# System Requirements

* 1. You must have the following items to complete this lab:
  + ASP.NET and ASP.NET MVC 3
  + Visual Studio 2010 Express
  + SQL Server Database (Express edition or above)
    1. **Note:** You can install the previous system requirements by using the Web Platform Installer 3.0: <http://go.microsoft.com/fwlink/?LinkID=194638>.

# Setup

#### Installing Code Snippets

* 1. For convenience, much of the code you will be managing along this lab is available as Visual Studio code snippets. To install the code snippets run **.\Source\Assets\CodeSnippets.vsi** file.

# Exercises

* 1. This Hands-On Lab is comprised by the following exercises:
  2. Exercise 1: Adding custom range validation
  3. Exercise 2: Adding validation at client side
  4. Exercise 3: Using IValidatableObject custom validation
  5. Exercise 4: Using Unobtrusive jQuery at client side
  6. Estimated time to complete this lab: **45 minutes**.
  7. **Note:** Each exercise is accompanied by an **End** folder containing the resulting solution you should obtain after completing the exercises. You can use this solution as a guide if you need additional help working through the exercises.

# Next Step

Exercise 1: Adding Custom Range Validation Using Validation Attributes

Exercise 1: Adding Custom Range Validation Using Validation Attributes

* 1. In this exercise, you will learn how to create a custom validation by using MVC3 ValidationAttribute class at server side. For that purpose, you will include a range validation in the Store Manager example to validate that the price of an album belongs to a certain range.

Task 1 – Adding a validation attribute class

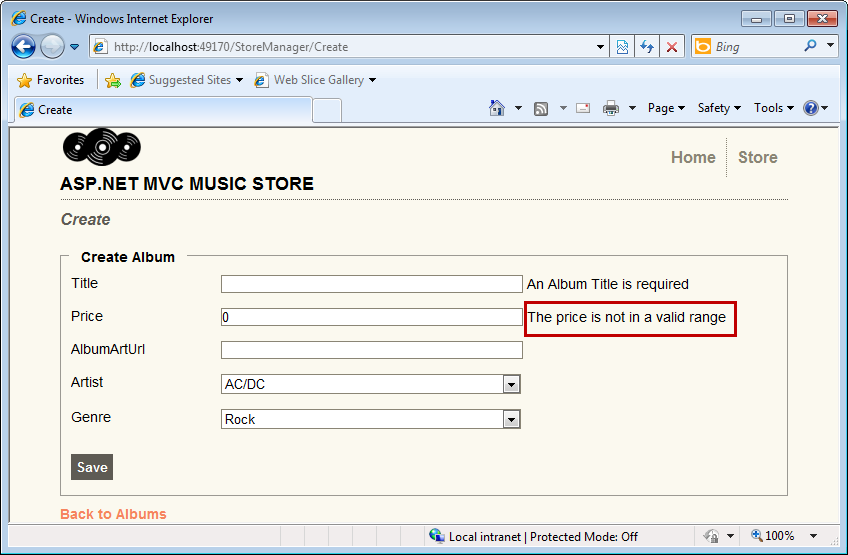
* 1. In this task, you will add a validation attribute class at server side. Each attribute to be validated should have a new class inherited from **ValidationAttribute**.
  2. Open the begin solution **MvcMusicStore.sln** at Source\Ex01-Server Side Custom Validation\Begin
  3. Create a new folder **Validations** at project root, which will include all the custom rules.
  4. Add a new C# class into the **Validations** folder and rename it to **PriceValidationAttribute.cs**
  5. Open **PriceValidationAttribute.cs** and add a reference to the **System.ComponentModel.DataAnnotations** namespace:
     1. C#
     2. using System;
     3. using System.Collections.Generic;
     4. using System.Web;
     5. **using System.ComponentModel.DataAnnotations;**
  6. Inherit the **PriceValidationAttribute** class from **ValidationAttribute**. Additionally, you have to add the **AttributeUsage** attribute specification, which in this case can be set to fields and properties:
     1. C#
     2. using System;
     3. using System.Collections.Generic;
     4. using System.Web;
     5. using System.ComponentModel.DataAnnotations;
     6. **[AttributeUsage(AttributeTargets.Field | AttributeTargets.Property, AllowMultiple = false, Inherited = true)]**
     7. public class PriceValidationAttribute : **ValidationAttribute**
     8. {
     9. }
     10. **Note:**AttributeUsage determines how custom attribute class’s attributes can be applied.
     11. **[System.AttributeUsage(System.AttributeTargets.All,**
     12. **AllowMultiple = false, Inherited = true)]**
     13. **AttributeTargets:** This parameter specifies all the application elements on which it is **valid to apply an attribute**. It is possible to use **System.AttributeTargets.All**, or to specify each member (for example, Class, Property, Method or Field).
     14. **AllowMultiple:** If it is set to true, then the attribute can be applied more than once in the target
     15. **Inherit:** If it is set to false, then the attribute is not inherited by derived attribute classes.
     16. You could read more about **AttributeUsage** in [msdn](http://msdn.microsoft.com/en-us/library/tw5zxet9%28v=VS.100%29.aspx).
  7. Define the thresholds for the album price range. To do this, add the following code to the generated class:
     1. (Code Snippet – ASP.NET MVC 3 Custom Validation – Ex1 Custom Range Validation Thresholds - CSharp)
     2. C#
     3. using System;
     4. using System.Collections.Generic;
     5. using System.Linq;
     6. using System.Web;
     7. using System.ComponentModel.DataAnnotations;
     8. namespace MvcMusicStore.Validations
     9. {
     10. [AttributeUsage(AttributeTargets.Field | AttributeTargets.Property, AllowMultiple = false, Inherited = true)]
     11. public class PriceValidationAttribute : ValidationAttribute
     12. {
     13. **private decimal minPrice = 0.01M;**
     14. **private decimal maxPrice = 100.00M;**
     15. }
     16. }
     17. **Note:** The values of **minPrice** and **maxPrice** are initialized here, but they could be retrieved from a database query or a Web Service call, making the application more maintainable.
  8. Implement a class initializer and the method **IsValid** from the base class. This step is required for custom validation at server side:
     1. (Code Snippet – ASP.NET MVC 3 Custom Validation – Ex1 Custom Range Validation IsValid- CSharp)
     2. C#
     3. using System;
     4. using System.Collections.Generic;
     5. using System.Linq;
     6. using System.Web;
     7. using System.ComponentModel.DataAnnotations;
     8. [AttributeUsage(AttributeTargets.Field | AttributeTargets.Property, AllowMultiple = false, Inherited = true)]
     9. public class PriceValidationAttribute : ValidationAttribute
     10. {
     11. private decimal minPrice = 0.01M;
     12. private decimal maxPrice = 100.00M;
     13. **public PriceValidationAttribute(): base("The price is not in a valid range")**
     14. **{**
     15. **}**
     16. **public override bool IsValid(object value)**
     17. **{**
     18. **decimal price = (decimal)value;**
     19. **if (price < this.minPrice || price > this.maxPrice)**
     20. **return false;**
     21. **return true;**
     22. **}**
     23. }

Task 2 – Creating the Custom Data Annotation

* 1. In this task, you will add a new data annotation tag to the **Album** class thatwill be automatically linked to the validation method just created:
  2. Open the **Album.cs** file at **MvcMusicStore\Models**.
  3. Add **Validations** namespace into the **Album** class
     1. C#
     2. using MvcMusicStore.Validations;
  4. Replace current data annotation with the **PriceValidation** attribute:
     1. C#
     2. ~~[Range(0.01, 100.00, ErrorMessage = "Price must be between 0.01 and 100.00")]~~
     3. **[PriceValidation]**
     4. After the replacement, the **Album.cs** file should look like the following:
     5. C#
     6. namespace MvcMusicStore.Models
     7. {
     8. [MetadataType(typeof(AlbumMetaData))]
     9. public partial class Album
     10. {
     11. // Validation rules for the Album class
     12. ...
     13. [Required(ErrorMessage = "Price is required")]
     14. [PriceValidation]
     15. public object Price { get; set; }
     16. }
     17. }

Task 3 – Running the Application

In this task, you will test that the **StoreManager** Create View template validates the price using custom validation.

* 1. Press **F5** to run the Application.
  2. The project starts in the Home page. Browse **/StoreManager/Create** and click **Save** without filling the form to verify that you get the price range validation message:
     1. 
     2. Figure 1
     3. Validating an album price

# Next Step

* 1. Exercise 2: Adding validation at client side

Exercise 2: Adding Validation at Client Side

1. In the previous exercise you added a custom range validation method for the album price in the server side.
2. In this exercise, you will learn how to add validation at client side. For that purpose you will implement MVC3 **IClientValidatable** Interface at client side. This will improve the user experience, as the error message will appear before saving the album.

Task 1 – Creating a ModelClientValidationRule for price ranges

* 1. **Note:** **ModelClientValidationRule** is an ASP.NET MVC class that provides a base class container for client validation rules sent to the browser. In order to achieve that, any generic rule should inherit from this class.
  2. In this task, you will create a new ModelClientValidationRule called **ModelClientPriceRangeValidationRule**. This derived class will provide a custom validation rule.
  3. Open the begin solution **MvcMusicStore.sln** at Source\Ex02- Client Side Custom Validation\Begin.
  4. Create a new C# class file into **Validations** folder, and rename it **ModelClientPriceRangeValidationRule.cs**.
  5. Open **ModelClientPriceRangeValidationRule.cs**. and add a reference to the **System.Web.Mvc**  namespace:
     1. C#
     2. using System;
     3. using System.Collections.Generic;
     4. using System.Linq;
     5. using System.Web;
     6. **using System.Web.Mvc;**
  6. Inherit the **ModelClientPriceRangeValidationRule** classfrom **ModelClientValidationRule**:
     1. C#
     2. using System;
     3. using System.Collections.Generic;
     4. using System.Linq;
     5. using System.Web;
     6. using System.Web.Mvc;
     7. namespace MvcMusicStore.Validations
     8. {
     9. public class ModelClientPriceRangeValidationRule**: ModelClientValidationRule**
     10. {
     11. }
     12. }
  7. Create a new class constructor for **ModelClientPriceRangeValidationRule** to set the base class properties with your custom parameters:
     1. (Code Snippet – ASP.NET MVC 3 Adding validation at client side – Ex2 Constructor - CSharp)
     2. C#
     3. using System.Web.Mvc;
     4. namespace MvcMusicStore.Validations
     5. {
     6. public class ModelClientPriceRangeValidationRule : ModelClientValidationRule
     7. {
     8. **public ModelClientPriceRangeValidationRule(string errorMessage, decimal minPrice, decimal maxPrice)**
     9. **{**
     10. **ErrorMessage = errorMessage;**
     11. **ValidationType = "priceOnRange";**
     12. **ValidationParameters.Add("minPrice", minPrice);**
     13. **ValidationParameters.Add("maxPrice", maxPrice);**
     14. **}**
     15. }
     16. }
     17. **Note:** In the following steps you will see how this piece is connected with the general solution.

Task 2 – Adding attribute validation at client side

* 1. In this task, you will add a method to **PriceValidationAttribute** class to perform validation at client-side.
  2. Open **PriceValidationAttribute.cs** from **\MvcMusicStore\Validations**.

Make the **PriceValidationAttribute** class implement the **IClientValidatable** interface

* + 1. C#
    2. public class PriceValidationAttribute : ValidationAttribute, **IClientValidatable**
  1. Make the **PriceValidationAttribute** override the method **GetClientValidationRules**
     1. (Code Snippet – ASP.NET MVC 3 Adding validation at client side – Ex2 GetClientValidationRules override - CSharp)
     2. C#
     3. ...
     4. public class PriceValidationAttribute : ValidationAttribute, IClientValidatable
     5. {
     6. ...
     7. public override bool IsValid(object value)
     8. {
     9. ...
     10. }
     11. **// Client-Side validation**
     12. **public IEnumerable<ModelClientValidationRule> GetClientValidationRules(ModelMetadata metadata, ControllerContext context)**
     13. **{**
     14. **var rule = new ModelClientPriceRangeValidationRule("The price is not in a valid range.", this.minPrice, this.maxPrice);**
     15. **yield return rule;**
     16. **}**
     17. ...

Task 3 – Adding a JavaScript Function to Perform Client-Side Validation

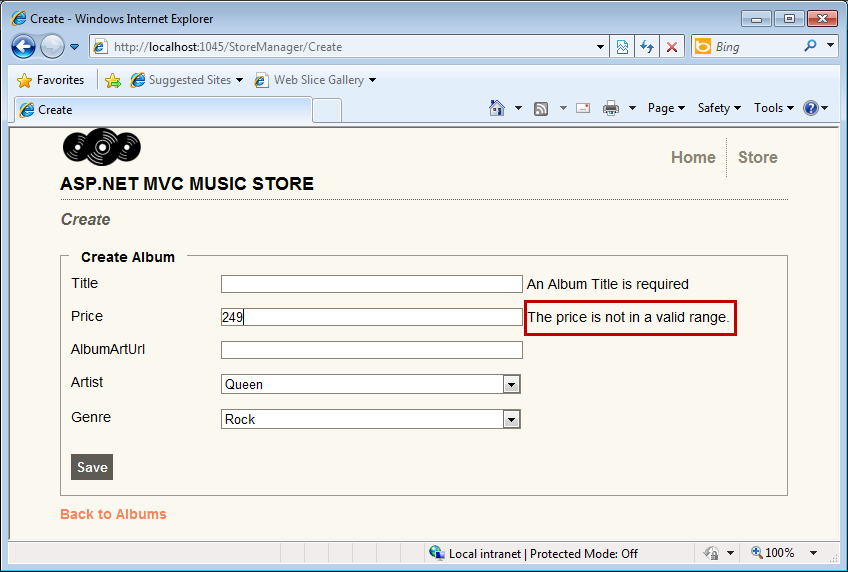
* 1. Create a new JavaScript file at **\MvcMusicStore\Scripts** and rename it to **PriceValidation.js**.
  2. Open **PriceValidation.js** and insert the following client side code that will validate the price range. You will note that it has the same logic as the **IsValid** method of **PriceValidationAttribute** class created in the previous exercise:
     1. JavaScript
     2. Sys.Mvc.ValidatorRegistry.validators["priceOnRange"] = function(rule) {
     3. var minPrice = rule.ValidationParameters.minPrice;
     4. var maxPrice = rule.ValidationParameters.maxPrice;
     5. var message = rule.ErrorMessage;
     6. return function (value, context) {
     7. if (value > maxPrice || value < minPrice) {
     8. return false;
     9. }
     10. return true;
     11. };
     12. };

Task 4 – Modifying the Create View to Execute Client-Side Validation

* 1. Open **Create.aspx** view from **\Views\StoreManager**.
  2. Add references to the following three JavaScript files that will take care of Ajax features, MVC validation and the custom price validation. Next, enable HTML client validation:
     1. HTML
     2. …
     3. <h2>Create</h2>
     4. **<script src="/Scripts/MicrosoftAjax.js" type="text/javascript"></script>**
     5. **<script src="/Scripts/MicrosoftMvcValidation.js" type="text/javascript"></script>**
     6. **<script src="/Scripts/PriceValidation.js" type="text/javascript"></script>**
     7. **<% Html.EnableClientValidation(); %>**
     8. <% using (Html.BeginForm()) {%>
     9. …

Task 5 – Running the Application

In this task, you will test that the **StoreManager** create view template performs a range validation at client side when the user enters an album price.

1. Press **F5** to run the Application.
2. The project starts in the Home page. Browse **/StoreManager/Create** and in the **Price** field enter a value outside of the validation range [0.01, 100]. You will see the following error message:
   * 1. 
     2. Figure
     3. Validating an album price at client side

# Next Step

Exercise 3: Using IValidatableObject custom validation

Exercise 3: Using IValidatableObject Custom Validation

In this exercise, you will learn how to implement a custom server-side validation using ASP.NET4 **IValidatableObject** Interface. In the previous exercises you worked with validation attributes to perform custom validation of several attributes. Now you will use **IValidatableObject** to prevent the insertion of repeated albums, performing an extra validation at server side that will run after attribute validation is performed.

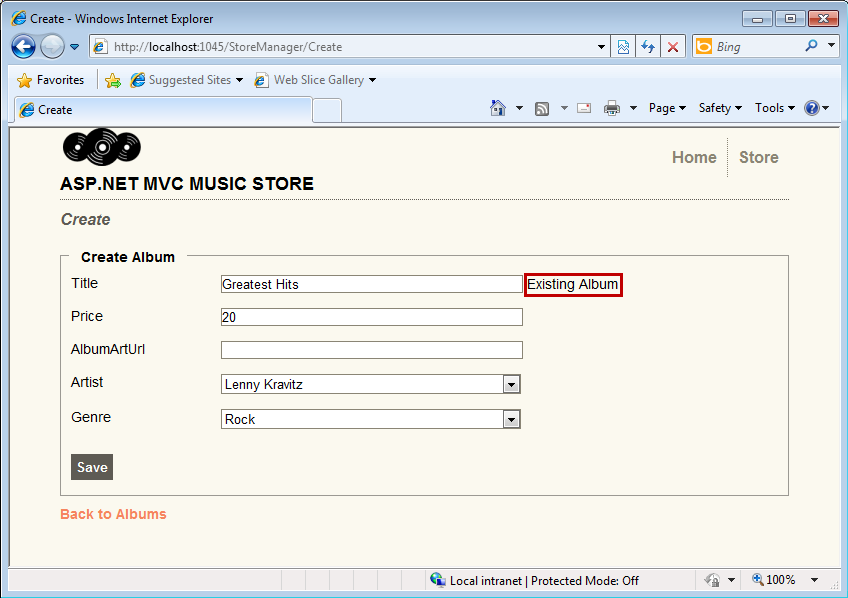
Task 1 – Implementing IValidatableObject Validation Method

* 1. Open the begin solution MvcMusicStore.sln at **Source\Ex03-Custom Validation IValidatableObject\Begin**.
  2. Open the **Album.cs** file at **\Models** and implement **IValidatableObject** interface:
     1. C#
     2. …
     3. namespace MvcMusicStore.Models
     4. {
     5. [MetadataType(typeof(AlbumMetaData))]
     6. public partial class Album **: IValidatableObject**
     7. {
     8. …
     9. **Note: IValidatableObject** interface (from System.ComponentModel.DataAnnotation) provides a way to invalidate an object. It has one member, **Validate** Method, which determines if the object that implements the interface is valid:
     10. **IEnumerable<ValidationResult> Validate(ValidationContext validationContext)**
     11. You could read more about **ValidationContext** in [msdn](http://msdn.microsoft.com/en-us/library/microsoft.visualstudio.modeling.validation.validationcontext.aspx).
  3. Implement **IValidatableObject Validate** method on **Album** class. The method will check if there is any album with the same title and artist in the database and returns a ValidationResult object.
     1. (Code Snippet – ASP.NET MVC 3 Adding IValidatableObject validation – Ex3 Validate method - CSharp)
     2. C#
     3. **public IEnumerable<ValidationResult> Validate(ValidationContext validationContext)**
     4. **{**
     5. **MusicStoreEntities storeDB = new MusicStoreEntities();**
     6. **if (storeDB.Albums.Any(a => a.Title.Trim().ToUpper() == this.Title.Trim().ToUpper() && a.ArtistId == (int)this.ArtistId))**
     7. **yield return new ValidationResult("Existing Album", new string[] { "Title" });**

1. **}**
   * 1. **Note: ValidationResult** has a message (“Existing album”) and a target field in which the error is associated (“Title”). You will later note that the error message is displayed next to Title field.
     2. As **Validate** method is implemented into Model classes, you might not feel comfortable with the idea of having logic inside the model. For that reason, the use of **partial classes** to implement **Validate** method is recommended.

Task 2 – Running the Application

In this task, you will try to create a repeated album to test the validation.

1. Press **F5** to run the Application.
2. The project starts in the Home page. Browse **/StoreManager/Create**.
3. Fill the form with the following data: “**Greatest Hits**” in the **Title** field and “**Lenny Kravitz”** for **Artist.** Then, write any valid data in the other fields.
4. Press **Save** to submit the form. You will get the error message **Existing Album** at the right of the title:
   * 1. 
     2. Figure
     3. Validating the insertion of an existing album

# Next Step

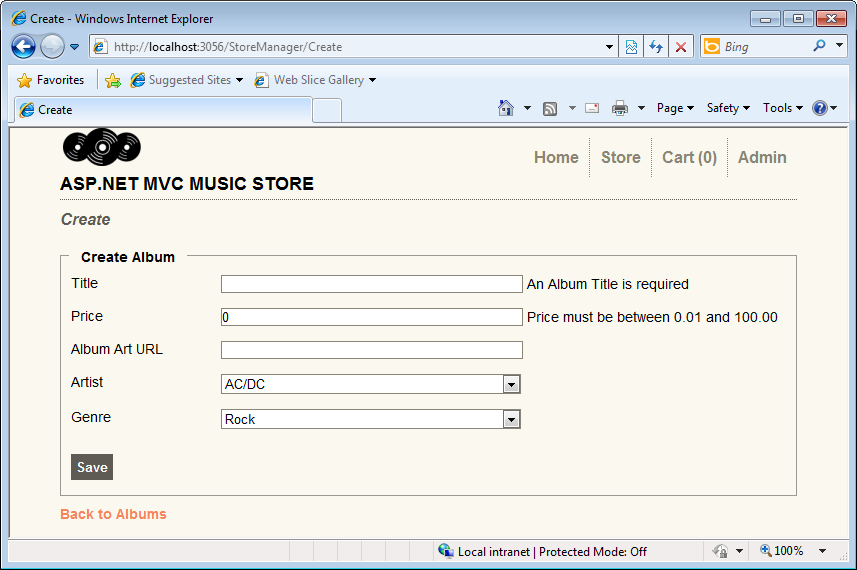
Exercise 4: Using Unobtrusive jQuery at client side

Exercise 4: Using Unobtrusive jQuery at Client Side

* 1. In this exercise, you will learn how to enable MVC 3 Unobtrusive jQuery validation at client side. For that reason you will include jQuery libraries into the Master Page and enable unobtrusive jQuery globally.
  2. **Note:** The Unobtrusive jQuery uses data-ajax prefix JavaScript to invoke action methods on the server rather than intrusively emitting inline client scripts.

Task 1 – Running the Application before Enabling Unobtrusive jQuery

In this task, you will run the application before including jQuery in order to compare both validation models.

* 1. Open the begin solution **MvcMusicStore.sln** at Source\Ex04-Unobtrusive JavaScript Validation\Begin and press **F5** to run the application.
  2. The project starts in the Home page. Browse **/StoreManager/Create** and click **Save** without filling the form to verify that you get validation messages:
     1. 
     2. Figure
     3. Client validation disabled
  3. In the browser, open the **Create** view source code:
     1. HTML – Create.html
     2. …
     3. <form action="/StoreManager/Create" method="post">
     4. **<div class="validation-summary-errors"><ul><li style="display:none"></li>**
     5. </ul></div>
     6. …
     7. <p>
     8. <label for="Album\_Title">Title</label>
     9. <input class="input-validation-error" id="Album\_Title" name="Album.Title" type="text" value="" />
     10. **<span class="field-validation-error">An Album Title is required</span>**
     11. </p>
     12. <p>
     13. <label for="Album\_Price">Price</label>
     14. <input class="input-validation-error" id="Album\_Price" name="Album.Price" type="text" value="0" />
     15. <span class="field-validation-error">Price must be between 0.01 and 100.00</span>
     16. …

Task 2 – Enabling Client Validation from Web.config

* 1. In this task, you will enable unobtrusive jQuery **client validation** from **Web.config** file, which is by default set to false in all new ASP.NET MVC 3 projects.
  2. Open **Web.Config** file at project root, and make sure that the **ClientValidationEnabled** key value is set to **true**.
     1. XML - Web.Config
     2. …
     3. <configuration>
     4. <appSettings>
     5. <add key="ClientValidationEnabled" value="**true**"/>
     6. <add key="UnobtrusiveJavaScriptEnabled" value="true"/>
     7. </appSettings>
     8. …
     9. **Note:** You can also enable client validation by code at Global.asax.cs to get the same results:

**HtmlHelper.ClientValidationEnabled = true;**

Additionally, you can assign ClientValidationEnabled attribute into any controller to have a custom behavior.

Task 3 – Adding Unobtrusive jQuery to Master Page

* 1. In this task, you will add the unobtrusive jQuery references into the Master Page.
  2. Open **Site.Master** at **Views\Shared**.
  3. Add MVC 3 script references to **jQuery**:
     1. XML

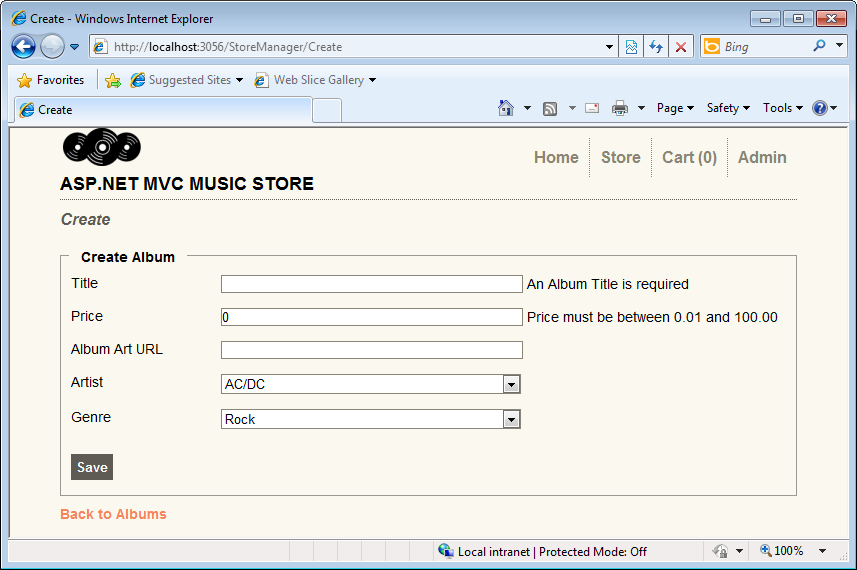
…

* + 1. <head runat="server">
    2. <link href="/Content/Site.css" rel="Stylesheet" type="text/css" />
    3. <title><asp:ContentPlaceHolder ID="TitleContent" runat="server" /></title>
    4. **<script src="../../Scripts/jquery-1.4.1.js" type="text/javascript"></script>**
    5. **<script src="../../Scripts/jquery.validate.js" type="text/javascript"></script>**
    6. **<script src="../../Scripts/jquery.validate.unobtrusive.js" type="text/javascript"></script>**

</head>

* + 1. **Note:** All these jQuery libraries are included in all MVC 3 new projects. You can find more libraries in the project’s folder **/Scripts**.

Task 4 – Running the Application Using Unobtrusive jQuery Validation

* 1. In this task, you will test that the **StoreManager** create view template performs client side validation using jQuery libraries when the user creates a new album.
  2. Press **F5** to run the application.
  3. The project starts in the Home page. Browse **/StoreManager/Create** and click **Save** without filling the form to verify that you get validation messages:
     1. 
     2. Figure
     3. Client validation with jQuery enabled
  4. In the browser, open the source code for Create view:
     1. HTML
     2. …
     3. </title>
     4. **<script src="../../Scripts/jquery-1.4.1.js" type="text/javascript"></script>**
     5. **<script src="../../Scripts/jquery.validate.js" type="text/javascript"></script>**
     6. **<script src="../../Scripts/jquery.validate.unobtrusive.js" type="text/javascript"></script>**
     7. </head>
     8. …
     9. <p>
     10. <label for="Album\_Title">Title</label>
     11. **<input data-val="true" data-val-length="The field Title must be a string with a maximum length of 160." data-val-length-max="160" data-val-required="An Album Title is required" id="Album\_Title" name="Album.Title" type="text" value="" />**
     12. </p>
     13. <p>
     14. <label for="Album\_Price">Price</label>
     15. **<input data-val="true" data-val-number="The field Price must be a number." data-val-range="Price must be between 0.01 and 100.00" data-val-range-max="100" data-val-range-min="0.01" data-val-required="Price is required" id="Album\_Price" name="Album.Price" type="text" value="0" />**
     16. <span class="field-validation-valid" data-valmsg-for="Album.Price" data-valmsg-replace="true"></span>
     17. </p>
     18. <span class="field-validation-valid" data-valmsg-for="Album.Title" data-valmsg-replace="true"></span>
     19. </p>
     20. …

**Note:** For each client validation rule, Unobtrusive jQuery adds an attribute with data-val-rulename="message". Below is a list of tags that Unobtrusive jQuery inserts into the html input field to perform client validation:

* **Data-val**
* **Data-val-number**
* **Data-val-range**
* **Data-val-range-min / Data-val-range-max**
* **Data-val-required**
* **Data-val-length**
* **Data-val-length-max / Data-val-length-min**

All the data values are filled with model **Data Annotation**. Then, all the logic that works at server side can be run at client side.

For example, Price attribute has the following data annotation in the model:

* + 1. **[Required(ErrorMessage = "Price is required")]**
    2. **[Range(0.01, 100.00, ErrorMessage = "Price must be between 0.01 and 100.00")]**

**public object Price { get; set; }**

After using Unobtrusive jQuery, the generated code is:

<input data-val="true"

**data-val-number="The field Price must be a number."**

**data-val-range="Price must be between 0.01 and 100.00"**

**data-val-range-max="100"**

**data-val-range-min="0.01"**

**data-val-required="Price is required"**

id="Album\_Price" name="Album.Price" type="text" value="0" />

# Next Step

Summary

Summary

* 1. By completing this Hands-On Lab you have learned how to use custom validations in your page with the user of the following concepts:
  + Data annotations for model validation
  + ASP.NET MVC 3 IClientValidatable interface
  + A Custom MVC ModelClientValidationRule
  + MVC ValidationAttribute inheritance
  + A Custom server validation using IValidatableObject interface
  + ASP.NET MVC 3 Unobtrusive jQuery validation