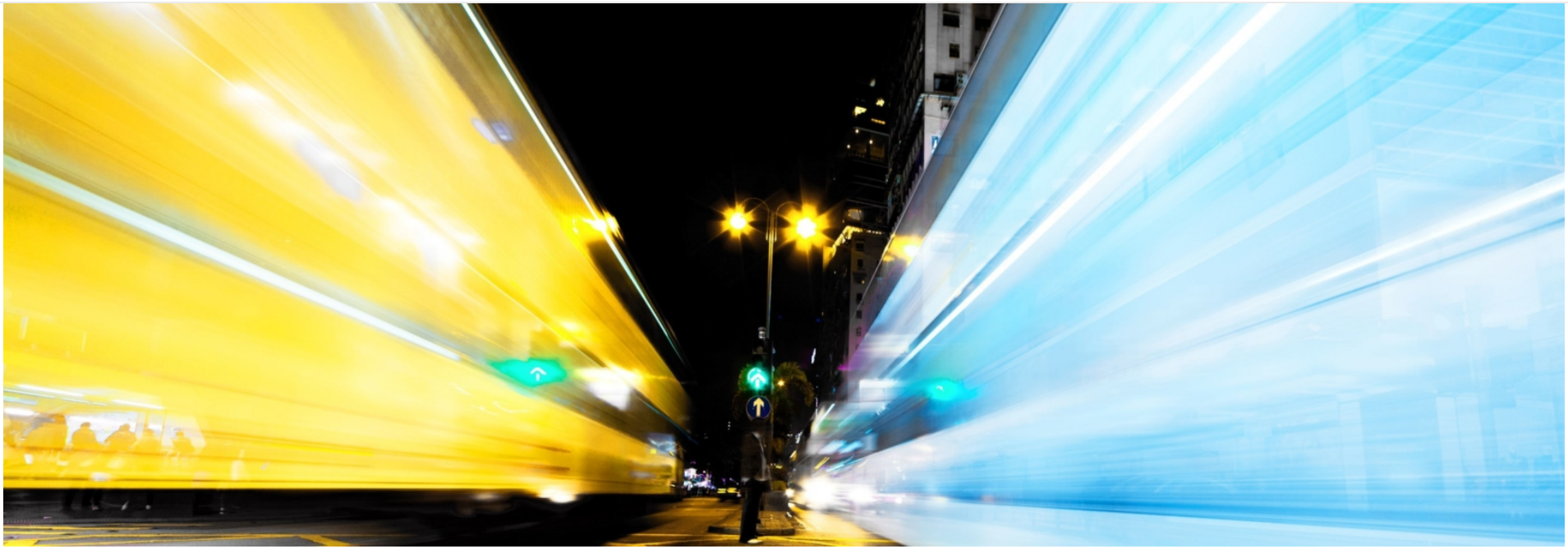
CapTech

MENU

BLOG | SEPTEMBER 11, 2013

Handling Multiple Validation Rules in ASP.NET MVC Using DataAnnotations and Unobtrusive jQuery Validator - Part 1

AUTHOR
Ammar Ahmed



We often come across the requirement to validate the same field for different conditions. Imagine, for example, you have a name field that must satisfy three different conditions:

- it must be an individual person (e.g., can't have 'and', '&' or '/')
- it cannot contain a variation of'Not Applicable'
- it cannot contain numbers

Given these three rules, how can we present a unified validation experience for the user? Using .NET DataAnnotations, validation is easily achieved using attribute decorators on our model:

```
public class RegisterModel
 {
 [Required]
```

```
 [Required]
 [OnlyOne]
 [CannotBeNotApplicable]
 [CannotContainNumbers]
 public string LastName { get; set; }

 [Email]
 public string Email { get; set;}
}
```

Here is an example for creating custom validation in ASP.NET MVC3. However, when you apply more than one validation attribute to a field, only one error message is displayed at a time, and further validation is short-circuited. Until you correct the error shown in the error message, it does not check the other validations. Moreover, the order in which the validation is fired is not known. This may result in a bad user experience.

In this post we will combine these independent validation attributes in a single group. By logically grouping individual validation rules into one validation rule, we can minimize the clutter of DataAnnotations on our model and also check more than one validation rule at one time while customizing our error message accordingly. In Part-2 we'll show you how to integrate jQuery's client-side validators to achieve a unified user experience.

Given the requirements above, if the user enters 'Mr & Mrs 1234', then we should display something like: "This field must be an individual and cannot contain numbers."

SERVER SIDE VALIDATION

To achieve this result, we create a custom validator we'll call ValidNameAttribute to encapsulate a subset of specific validators that will perform each of the validation rules listed above. The IsValid method simply validates against each member rule and appends the validation message to an error message string.

```
public class ValidNameAttribute: ValidationAttribute
{
```

```
 public ValidNameAttribute()
 {
 _onlyOneEntityValidator = new OnlyOneAttribute
 {ErrorMessage = "cannot contain more than one {0}"}
 _cannotBeNotApplicable = new CannotBeNotApplicableAttribute
 {ErrorMessage = "cannot be Not Applicable"};
 _cannotContainNumbers = new CannotContainNumbersAttribute
 {ErrorMessage = "cannot contain numbers"};
 }

protected override ValidationResult IsValid(object value, ValidationContext validationContext)
 {
 var errormessage = new StringBuilder("This field ");
 if (value != null)
 {
 bool isValid = true;
 if(!_onlyOneEntityValidator.IsValid(value))
 {
 isValid = false;
 errormessage.Append(_onlyOneEntityValidator
 .FormatErrorMessage(validationContext.DisplayName)+", ");
 }
 if(!_cannotBeNotApplicable.IsValid(value))
 {
 isValid = false;
 errormessage.Append(_cannotBeNotApplicable
 .FormatErrorMessage(validationContext.DisplayName) + ", ");
 }
 if(!_cannotContainNumbers.IsValid(value))
 {
 isValid = false;
 errormessage.Append(_cannotContainNumbers
 .FormatErrorMessage(validationContext.DisplayName) + " ");
 }

 return isValid ? null
 : new ValidationResult(errormessage.ToString(), new[] { validationContext.MemberName });
 }
```

Decorating one property in our model with the new ValidName attribute shows how we now see validation messaging for all invalid inputs at once, as compared to the original where validation occurs sequentially.

```
public class RegisterModel
 {
 [Required]
 [ValidName]
 public string FirstName { get; set; }

 [Required]
 [OnlyOne]
 [CannotBeNotApplicable]
 [CannotContainNumbers]
 public string LastName { get; set; }

 [Email]
 public string Email { get; set;}

 }
```

The error message is generated at run time, includes only the messages of the failed rules from the subset of validators.

In Part-2 we will integrate it with unobtrusive jQuery valdiation to acheive same result on the client-side.

RELATED CONTENT

BLOG

Forging a Path to Business Agility: The Foundations

CapTech

MENU

BLOG

Living the BA SME Life: An Interview with Philly's Dee Gavlick

UP NEXT

Meet the ASQ →

How can we help you?

GET IN TOUCH