ASP.NET MVC custom multiple fields validation

Asked 3 years ago Active 3 years ago Viewed 4k times

4

1

I'm developing an ASP.NET MVC 5.2.3 custom data annotation for validation in Visual Studio 2015. It needs to take any number of fields and ensure that if one has a value, they all must have a value; if they're all null/blank, it should be okay.

A few examples have helped:

- ASP.NET MVC implement custom validator use IClientValidatable
- MVC Form Validation on Multiple Fields
- http://www.macaalay.com/2014/02/24/unobtrusive-client-and-server-side-age-validation-in-mvc-using-custom-data-annotations/

However, I'm not sure how to do the client-side validation where you have an unknown number of fields being validated.

How do you pass that to the client using the implementation of the `GetClientValidationRules()` method of the `IClientValidatable` interface?

Also, how do I apply this new data annotation to the properties on my view model? Would it look like this?

```
[MultipleRequired("AppNumber", "UserId", /* more fields */), ErrorMessage =
"Something..."]
[DisplayName("App #")]
public int AppNumber { get; set; }

[DisplayName("User ID")]
public int UserId { get; set; }
```

Here's as far as I could get with the `MultipleRequiredAttribute` custom data annotation class:

```
public class MultipleRequiredAttribute : ValidationAttribute, IClientValidatable
{
    private readonly string[] _fields;
    }

    protected override ValidationResult IsValid(object value, ValidationContext
validationContext)
    {
        // If any field has value, then all must have value
        var anyHasValue = _fields.Any(f => !string.IsNullOrEmpty(f));

        if (!anyHasValue) return null;

        foreach (var field in _fields)
        {
            var property = validationContext.ObjectType.GetProperty(field);
            if (property == null)
                return new ValidationResult($"Property '{field}' is undefined.");

            var fieldValue = property.GetValue(validationContext.ObjectInstance, null);

            if (string.IsNullOrEmpty(fieldValue?.ToString()))
                return new
ValidationResult(FormatErrorMessage(validationContext.DisplayName));
        }

        return null;
    }

    public IEnumerable<ModelClientValidationRule> GetClientValidationRules(ModelMetadata
metadata, ControllerContext context)
    {
        yield return new ModelClientValidationRule
        {
            ErrorMessage = ErrorMessage,
            ValidationType = "multiplerequired"
        };
    }
}
```

Thank you.

c# asp.net asp.net-mvc validation data-annotations

edited May 23 '17 at 12:18 asked Feb 3 '17 at 20:45

Community Alex

you build a custom function for jquery Validate js plugin on client side – Steve Feb 3 '17 at 20:55

2 Start by reading The Complete Guide To Validation In ASP.NET MVC 3 - Part 2. In your `GetClientValidationRules()` method, you add a `ModelClientValidationRule` where you can pass a (say) comma separated list of the property names - i.e. your `fields` values - which can be parsed and used in the client side scripts (if your having issues, let me know and I'll add an answer but wont get a chance for a few hours) – user3559349 Feb 3 '17 at 21:03

Thanks, @StephenMuecke! One of my issues was how to pass the values to the client. – Alex Feb 3 '17 at 21:17

1 You question states *if one has a value, they all must have a value* but your code is not validating that (and you would also need to apply the attribute to all properties if that is the case) – user3559349 Feb 4 '17 at 0:32

1 Also your `return new ValidationResult($"Property '{field}' is undefined.");` does not really makes sense (displaying that message in the view would be meaningless and confusing to the user) - either ignore it, or check in in the constructor and throw an exception – user3559349 Feb 4 '17 at 0:35

1 Answer

2

In order to get client side validation, you need to pass the values of the 'other properties' in the `ModelClientValidationRule` by using the `.Add()` method of the rules `ValidationParameters` property, and then write the client side scripts to add the rules to the `$.validator`.

But first there are a few other issues to address with your attribute. First you should execute your `foreach` loop only if the value of the property you applied the attribute is `null`. Second, returning a `ValidationResult` if one of the 'other properties' does not exist is confusing and meaningless to a user and you should just ignore it.

The attribute code should be (note I changed the name of the attribute)

```
public class RequiredIfAnyAttribute : ValidationAttribute, IClientValidatable
{
    private readonly string[] _otherProperties;
    private const string _DefaultErrorMessage = "The {0} field is required";

    public RequiredIfAnyAttribute(params string[] otherProperties)
    {
        if (otherProperties.Length == 0) // would not make sense
        {
            throw new ArgumentException("At least one other property name must be
provided");
        }
    protected override ValidationResult IsValid(object value, ValidationContext
validationContext)
    {
        if (value == null) // no point checking if it has a value
        {
            foreach (string property in _otherProperties)
            {
                var propertyName = validationContext.ObjectType.GetProperty(property);
                if (propertyName == null)
                {
                    continue;
                }
                var propertyValue =
propertyName.GetValue(validationContext.ObjectInstance, null);
                if (propertyValue != null)
                {
                    return new
ValidationResult(FormatErrorMessage(validationContext.DisplayName));
                }
            }
        }
        return ValidationResult.Success;
    }
    public IEnumerable<ModelClientValidationRule> GetClientValidationRules(ModelMetadata
metadata, ControllerContext context)
    {
        var rule = new ModelClientValidationRule
        {
            ValidationType = "requiredifany",
            ErrorMessage = FormatErrorMessage(metadata.GetDisplayName()),
        };
        / pass a comma separated list of the other propeties
        rule.ValidationParameters.Add("otherproperties", string.Join(",",
_otherProperties));
        yield return rule;
    }
}
```

The scripts will then be

```
sandtrapValidation = {
    getDependentElement: function (validationElement, dependentProperty) {
        var dependentElement = $('#' + dependentProperty);
```

```
            var name = validationElement.name;
            var index = name.lastIndexOf(".") + 1;
            var id = (name.substr(0, index) + dependentProperty).replace(/[\.\[\]]/g, "_");
            dependentElement = $('#' + id);
            if (dependentElement.length === 1) {
                return dependentElement;
            }
            // Try using the name attribute
            name = (name.substr(0, index) + dependentProperty);
            dependentElement = $('[name="' + name + '"]');
            if (dependentElement.length > 0) {
                return dependentElement.first();
            }
            return null;
        }
    }

    $.validator.unobtrusive.adapters.add("requiredifany", ["otherproperties"], function
    (options) {
        var element = options.element;
        var otherNames = options.params.otherproperties.split(',');
        var otherProperties = [];
        $.each(otherNames, function (index, item) {
            otherProperties.push(sandtrapValidation.getDependentElement(element, item))
        });
        options.rules['requiredifany'] = {
            otherproperties: otherProperties
        };
        options.messages['requiredifany'] = options.message;
    });

    $.validator.addMethod("requiredifany", function (value, element, params) {
        if ($(element).val() != '') {
            // The element has a value so its OK
            return true;
        }
        var isValid = true;
        $.each(params.otherproperties, function (index, item) {
            if ($(this).val() != '') {
                isValid = false;
            }
        });
        return isValid;
    });
```

user3559349

1 Its a general purpose function for finding an associated element in the DOM. You may have a model containing properties which are complex objects or collections, so it might be rendering as inputs with (say) `name="Employees[0].FirstName" id="Employees_0__FirstName"` and `name="Employees[0].LastName" id="Employees_0__LastName"` . So assume you want to validate the `LastName` is required if `FirstName` is provided. All you can pass in the `GetClientValidationRules()` method is the name of the other property i.e. `LastName` – user3559349 Feb 6 '17 at 21:29

1 The method first checks if the DOM includes an element with `id="LastName"` . For a simple object, that will return an element. But in this case it wont, so the next part of the function gets the name of the current element (which is `name="Employees[0].FirstName"`) and gets the part to the left of the last dot (`Employees[0]`) and appends to other property to generate `Employees[0].LastName` . Because searching by `id` is faster than by `name` attribute, the `.replace()` generates `Employees_0__LastName` and a search for that element is performed. – user3559349 Feb 6 '17 at 21:33

1 In most cases that will find what you want, but some users override the `id` attribute so nothing would be found, and the final check if based on finding the element based on the `name` attribute. – user3559349 Feb 6 '17 at 21:36

1 As a side note, I simplified that method a bit and omitted some code related to elements that are checkboxes and radio buttons which need to be handled a bit differently, but I don't think that would be applicable in your case – user3559349 Feb 6 '17 at 21:42

1 The 'alias' is just namespacing the function (its part of a my jquery plugin that includes numerous validation functions that are not handles by the built in validation attributes) and just prevents any possible (albeit unlikely) conflicts with plugins – user3559349 Feb 8 '17 at 0:55