



C# Corner Annual Conference 2020 Tickets on Sale Now

X



C# Corner

Top 10 Cloud Service Providers

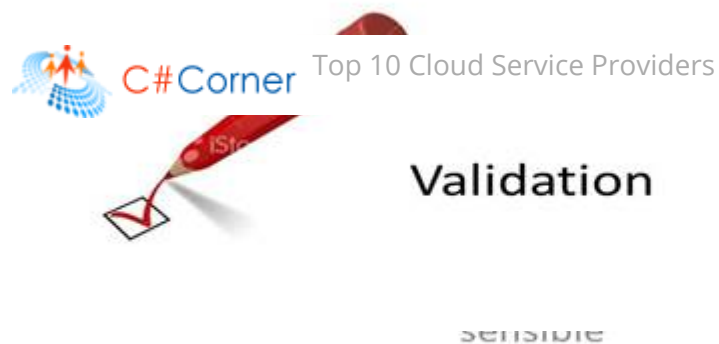


Custom Data Annotation Validation I

[Become a member](#)[Login](#)[Post](#)[Ask Question](#)[Download Free .NET & JAVA Files API](#)[Try Free File Format APIs for Word/Excel/PDF](#)

Introduction

Validating user input has always been a challenging task for web developers. Not only do we want validation logic executing in the browser, but also we must validate the logic running on the server. The client side logic gives users instant feedback on the information they entered into a web page and is an expected feature in today's applications. Meanwhile the server validation logic is in place because you should never trust information arriving from the network.



Join a member Login
Post Ask Question

In this article, we will see how custom validation logic works in data annotations in MVC framework. We will see how we can implement our custom logic in data annotation attributes. If you are new in data annotation validation please read my first article for built in data annotation validation.

- [Data Annotation And Validation in MVC](#)

In this article we will try to implement our custom validation logic for address field in which address cannot accept more than 50 words.

Overview

For this article we created a application ASP.NET MVC application name *DataAnnotationsValidations* (you can download the source code for better understanding) and we are using Student Model Class that contains student relation information in which we are going to validate using Data Annotation.

```
01. public class StudentModel
02. {
03.     public Guid StudentId { get; set; }
04.     public string FirstName { get; set; }
05.     public string LastName { get; set; }
06.     public DateTime DateOfBirth { get; set; }
```

```

07.     public string Address { get; set; }
08.     public string ContactNo { get; set; }
09.     public string EmailId { get; set; }

```



C# Corner

Top 10 Cloud Service Providers

```

12.     public string Password { get; set; }
13. }

```

[Come a member](#)
[Login](#)
[Post](#)
[Ask Question](#)

We have added a Student Controller and added Post action method to add new student. In this post action method we will

Student Controller

```

01. using System.Web.Mvc;
02. using DataAnnotationsValidations.Models;
03.
04. namespace DataAnnotationsValidations.Controllers
05. {
06.     public class StudentController : Controller
07.     {
08.         // GET: Student
09.         public ActionResult Index()
10.         {
11.             return View();
12.         }
13.
14.         // GET: Student/Create
15.         public ActionResult Create()
16.         {
17.             return View();
18.         }
19.
20.         // POST: Student/Create
21.         [HttpPost]
22.         public ActionResult Create(StudentModel student)
23.         {
24.             try
25.             {
26.                 if (ModelState.IsValid)
27.                 {

```

```
28. |
29. |         return RedirectToAction("Index");
30. |     }
```



C# Corner

Top 10 Cloud Service Providers

```
33. |         catch
34. |         {
35. |             return View();
36. |         }
```

```
39. |     }
40. | }
```

[Come a member](#)[Login](#)[Post](#)[Ask Question](#)

We have added a student view for Create action method when we run that view it will look like this.

Add New Student

**C# Corner**

Top 10 Cloud Service Providers

First Name

Last Name

DateOfBirth

Address

ContactNo

EmailId

ConfirmEmail

UserName

Password

Income

Create[Come a member](#)[Login](#)[Post](#)[Ask Question](#)

Create Student View

```
01. @model DataAnnotationsValidations.Models.StudentModel
02.
03. @{
04.     ViewBag.Title = "Add Student";
05. }
```

```
06.
07. <h3>Add New Student</h3>
08.
```



C# Corner Top 10 Cloud Service Providers

```
11. @Html.AntiForgeryToken()
12.
13. <div class="form-horizontal">
14.     <hr />
```

[Come a member](#)

[Login](#)

[Post](#)

[Ask Question](#)

```
17.         @Html.LabelFor(model => model.StudentId, htmlAttributes: new { @class = "control-
18. label col-md-2" })
19.         <div class="col-md-10">
20.             @Html.EditorFor(model => model.StudentId, new { htmlAttributes = new { @class = "form-
21. control" } })
22.             @Html.ValidationMessageFor(model => model.StudentId, "", new { @class = "text-
23. danger" })
24.         </div>
25.     </div>
26.     <div class="form-group">
27.         @Html.LabelFor(model => model.FirstName, htmlAttributes: new { @class = "control-
28. label col-md-2" })
29.         <div class="col-md-10">
30.             @Html.EditorFor(model => model.FirstName, new { htmlAttributes = new { @class = "form-
31. control" } })
32.             @Html.ValidationMessageFor(model => model.FirstName, "", new { @class = "text-
33. danger" })
34.         </div>
35.     </div>
36.     <div class="form-group">
37.         @Html.LabelFor(model => model.LastName, htmlAttributes: new { @class = "control-
38. label col-md-2" })
39.         <div class="col-md-10">
40.             @Html.EditorFor(model => model.LastName, new { htmlAttributes = new { @class = "form-
41. control" } })
42.             @Html.ValidationMessageFor(model => model.LastName, "", new { @class = "text-
43. danger" })
44.         </div>
```



Top 10 Cloud Service Providers

```

38.         </div>
39.
40.         <div class="form-group">
41.             <div class="col-md-10">
42.                 @Html.EditorFor(model => model.DateOfBirth, new { htmlAttributes = new { @class = "form-control" } })
43.                 @Html.ValidationMessageFor(model => model.DateOfBirth, "", new { @class = "text-danger" })
44.             </div>
45.         </div>
46.
47.         <div class="form-group">
48.             @Html.LabelFor(model => model.Address, htmlAttributes: new { @class = "control-label col-md-2" })
49.             <div class="col-md-10">
50.                 @Html.EditorFor(model => model.Address, new { htmlAttributes = new { @class = "form-control" } })
51.                 @Html.ValidationMessageFor(model => model.Address, "", new { @class = "text-danger" })
52.             </div>
53.         </div>
54.
55.         <div class="form-group">
56.             @Html.LabelFor(model => model.ContactNo, htmlAttributes: new { @class = "control-label col-md-2" })
57.             <div class="col-md-10">
58.                 @Html.EditorFor(model => model.ContactNo, new { htmlAttributes = new { @class = "form-control" } })
59.                 @Html.ValidationMessageFor(model => model.ContactNo, "", new { @class = "text-danger" })
60.             </div>
61.         </div>
62.
63.         <div class="form-group">
64.             @Html.LabelFor(model => model.EmailId, htmlAttributes: new { @class = "control-label col-md-2" })
65.             <div class="col-md-10">
66.                 @Html.EditorFor(model => model.EmailId, new { htmlAttributes = new { @class = "form-control" } })
67.

```

[Come a member](#) [Login](#)
[Post](#) [Ask Question](#)

```

68.         @Html.ValidationMessageFor(model => model.EmailId, "", new { @class = "text-
danger" })
69.     </div>

```



C# Corner

Top 10 Cloud Service Providers

```

72.     <div class="form-group">
73.         @Html.LabelFor(model => model.ConfirmEmail, htmlAttributes: new { @class = "control-
label col-md-2" })
74.     </div class="col-md-10">

```

Join a member

Login

Post

Ask Question

```

76.         @Html.ValidationMessageFor(model => model.ConfirmEmail, "", new { @class = "text-
danger" })
77.     </div>
78. </div>
79.
80.     <div class="form-group">
81.         @Html.LabelFor(model => model.UserName, htmlAttributes: new { @class = "control-
label col-md-2" })
82.         <div class="col-md-10">
83.             @Html.EditorFor(model => model.UserName, new { htmlAttributes = new { @class = "form-
control" } })
84.             @Html.ValidationMessageFor(model => model.UserName, "", new { @class = "text-
danger" })
85.         </div>
86.     </div>
87.
88.     <div class="form-group">
89.         @Html.LabelFor(model => model.Password, htmlAttributes: new { @class = "control-
label col-md-2" })
90.         <div class="col-md-10">
91.             @Html.EditorFor(model => model.Password, new { htmlAttributes = new { @class = "form-
control" } })
92.             @Html.ValidationMessageFor(model => model.Password, "", new { @class = "text-
danger" })
93.         </div>
94.     </div>
95.
96.     <div class="form-group">
97.         <div class="col-md-offset-2 col-md-10">
98.             <input type="submit" value="Create" class="btn btn-danger" />

```


[Come a member](#) [Login](#)

This is the initial set up we need to run this data annotation validation project. Now we are going to add data annotations one by one.

[Post](#) [is](#) [Ask Question](#) [tion](#)

Data annotations are attributes we can find in the `System.ComponentModel.DataAnnotations` namespace. These attributes provide server side validation and framework also supports client side validation.

Custom Annotations

Imagine we want to restrict the address field value of a student to limited number of words. For example we might say 50 words is more than enough for an address field. You might also think that this type of validation (limiting a string to a maximum number of words) is something we can reuse with other modules in our application. If so, the validation logic is a candidate for packaging into reusable attributes.

So making the attributes reusable we are going to place all the custom annotation logic in one common place/folder, say `Attributes`, as we know all the validation annotations (such as `Range`, `StringFormat`) ultimately derive from the `ValidationAttribute` base class. This base class is abstract and resides under `System.ComponentModel.DataAnnotations` namespace.

To keep validation logic in one place I have added a `Attributes` folder name in application solution and added a class `MaxWordAttributes` class in it like this:

```
01. using System.ComponentModel.DataAnnotations;
02.
03. namespace DataAnnotationsValidations.Attributes
04. {
05.     public class MaxWordAttributes : ValidationAttribute
06.     {
07.     }
```

08. | }

To implement the validation logic, we need to override one of the `IsValid` methods provided by the base class. Overriding the `IsValid` method, we can pass a `Model` object instance, and friendly display name of the property we are validating, among other things. [Come a member](#) [Login](#)

```
01. public class MaxWordAttributes : ValidationAttribute
02. {
03.
04.
05.     return ValidationResult.Success;
06. }
07. }
```

[Post](#)[Ask Question](#)

The first parameter to the `IsValid` method is the value to validate. If the value is valid you can return successful validation results, but before we can determine whether the value is valid we need to know the count of the words in that field. So for that a programmer should pass how many values is too many for that property, to do that we need to create a constructor to accept a number of words.

```
01. public class MaxWordAttributes : ValidationAttribute
02. {
03.     private readonly int _maxWords;
04.     public MaxWordAttributes(int maxWords)
05.     {
06.         _maxWords = maxWords;
07.     }
08.     protected override ValidationResult IsValid(object value, ValidationContext validationContext)
09.     {
10.         return ValidationResult.Success;
11.     }
12. }
```

Now we have parameterized the maximum word count so we can implement our validation logic and catch the validation error.

```
01. public class MaxWordAttributes : ValidationAttribute
02. {
03.     private readonly int _maxWords;
```

```

04.     public MaxWordAttributes(int maxWords)
05.     {
06.         maxWords = maxWords;
07.
08.         // ...
09.     }
10.     if (value == null) return ValidationResult.Success;
11.     var textValue = value.ToString();
12.     return textValue.Split(' ').Length > maxWords
13.         ? new ValidationResult("Too long Address!")
14.         : ValidationResult.Success;
15. }
16. }

```



Top 10 Cloud Service Providers

Join a member

Login

Post

Ask Question

Here we just split the value and compare the count with maxWords. If it is greater than that it should return validation error message otherwise a successful result.

Here the problem with the last line `new ValidationResult("Too long Address!")` is we can see this is just a hardcoded error message. To write a good quality code we never want hardcoded error message. We always want to pass the error message and that error message should display instead of this hardcoded one.

To do that we just need to change our code a bit.

```

01. public class MaxWordAttributes : ValidationAttribute
02. {
03.     private readonly int _maxWords;
04.     public MaxWordAttributes(int maxWords)
05.         : base("{0} has to many words.")
06.     {
07.         _maxWords = maxWords;
08.     }
09.     protected override ValidationResult IsValid(object value, ValidationContext validationContext)
10.     {
11.         if (value == null) return ValidationResult.Success;
12.         var textValue = value.ToString();
13.         if (textValue.Split(' ').Length <= _maxWords) return ValidationResult.Success;
14.         var errorMessage = FormatErrorMessage(validationContext.DisplayName);
15.         return new ValidationResult(errorMessage);

```

```
16.     }
17. }
```



Top 10 Cloud Service Providers

[Come a member](#)
[Login](#)

- First we place the default error message to the base constructor. We should pull default file if we build an international stander application.

[Post](#) [Ask Question](#) [Source](#)

second change, the call inherited `FormatErrorMessage` method, will automatically format the string using the display name property.

`FormatErrorMessage` ensures that we use a correct error message string even if the string is localized into a resource file. We can use our custom annotation attributes in the following ways.

```
01. [DataType(DataType.MultilineText)]
02. [MaxWordAttributes(50)]
03. public string Address { get; set; }
```

In first use we will get default error message if validation failed and in second use we will get the message that we set in our model.

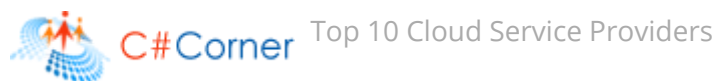
```
01. [DataType(DataType.MultilineText)]
02. [MaxWordAttributes(50, ErrorMessage="There are too many words in {0}.")]
03. public string Address { get; set; }
```

Address

Imagine living on Mars or a moon of Jupiter. The conversation around moving out into the solar system has always been a lively one. Would you go? By choice? By necessity? We're quickly making our own planet inhospitable to human life. Do you suppose we can pollute and corrupt

There are too many words in Address.

Remote



The Remote attributes enable us to perform client side validation with server callback. Take for example, if the UserName property is of a StudentModel, We are not going to allow the user name that already exists in the database. To say for every student there must be a unique UserName. To achieve this we need to use Remote attributes.

```
02. | public string UserName { get; set; }
```

Inside the attributes we can set the name of the action and the name of the controller the client call should call. The client code will send the value the user enters for the UserName property automatically and an overload of the attributes constructor allow us to specify additional fields to send to the server.

```
01. | public JsonResult CheckUserName(string userName)
02. |     {
03. |         var studentUserList = new List<string> { "Manish", "Saurabh", "Akansha", "Ekta", "Rakesh" };
04. |
05. |         var result = studentUserList.Any(userName.Contains);
06. |         return Json(result, JsonRequestBehavior.AllowGet);
07. |     }
```

For example if we have a user hardcoded string list we can easily replace that string list with the value we get from database. When we run the application and try to enter the value that is inside the string list it will show validation failed error message.

Conclusion

In this article, we learned about the custom data annotation validation in ASP.Net MVC framework. If you have any question or comments regarding this article, please post it in the comment section of this article. I hope you like it.


[Join a member](#)
[Login](#)

ASP.NET

Data Annotations

MVC

Validation

Post

Ask Question



Manish Kumar Choudhary is a passionate and pragmatic software engineer specializing in web application development with ASP.NET MVC, Web API, Entity Framework, NHibernate, Angular, Backbone, HTML5, and CSS. He started pro... [Read more](#)

<https://www.c-sharpcorner.com/authors/ee01e6/manish-kumar-choudhary.aspx>

188

3.3m

2

5

6



Type your comment here and press Enter Key (Minimum 10 characters)



Could you upload the code for this article, as you did for your first article? Thanks in advance.

[Jroughgarden](#)

1876 13 0

Oct 25, 2018

0 0 Reply



Yep, this is useful, thanks!

[Joe Gurria Celimendiz](#)

1883 6 0

Sep 28, 2018

0 0 Reply



This client side validation is kind of like ajax and is way more easy and simple to use than implementing IClientValidatable, thank for sharing!


[Chia-Chi Yen](#)

1886 3 0

May 07, 2018


0 0 Reply

Thanks for sharing.




Ravi Kandel

819 1.6k 326.9k




Top 10 Cloud Service Providers



kalu singh rao

305 6.3k 82k



64 24k 3.2m

0

0

Jul 14, 2016

Reply

come a member

Login

Jul 12, 2016

Post Ask Question Reply

0

0

Reply



Top 10 Cloud Service Providers

[Join a member](#)

[Login](#)

[Post](#)

[Ask Question](#)

FEATURED ARTICLES

[Understand Validation In Blazor Apps](#)

[Send an Email Reminder Notification Based on an Expiration Date using Power Automate](#)

Get Started With Bot Development With Microsoft Power Virtual Agents - Step By Step - Part One



Top 10 Cloud Service Providers

Don't Believe What You See

Join a member

Login

Post

Ask Question

All 

TRENDING UP

- 01 Let's Welcome Angular 9 With its New Features
- 02 Learn Angular 8 Step By Step In 10 Days - Route or Component Navigation - Day Ten
- 03 Getting Started With Bot Framework Composer
- 04 RabbitMQ Service Bus Explained
- 05 Multiple Dynamic Greeting Message Using Bot Framework Composer
- 06 Angular 8 - CRUD Operations - Part Three
- 07 Type of Filters in MVC Application and Why They're Important

08 Setup A Task Scheduler To Run Application Periodically In C#

**C# Corner**

Top 10 Cloud Service Providers

10

Deployments Modes In .NET Core 3.1

[Become a member](#)[Login](#)[Post](#)[Ask Question](#)[View All](#) 



Top 10 Cloud Service Providers

[C# Tutorials](#)

[Common Interview Questions](#)

[Stories](#)

[Consultants](#)

[Ideas](#)

[Ce](#)

[Come a member](#)

[Login](#)

©2020 C# Corner. All contents are copyright of their authors.

[Post](#)

[Ask Question](#)