How to get the Display Name Attribute of an Enum member via MVC razor code?

Asked 7 years ago Active 3 months ago Viewed 227k times

183

62

I've got a property in my model called "Promotion" that its type is a flag enum called "UserPromotion". Members of my enum have display attributes set as follows:

```
[Flags]
public enum UserPromotion
{
    None = 0x0,

    [Display(Name = "Send Job Offers By Mail")]
    SendJobOffersByMail = 0x1,

    [Display(Name = "Send Job Offers By Sms")]
    SendJobOffersBySms = 0x2,

    [Display(Name = "Send Other Stuff By Sms")]
    SendPromotionalBySms = 0x4,

    [Display(Name = "Send Other Stuff By Mail")]
    SendPromotionalByMail = 0x8
}
```

Now I want to be able to create say a ul in my view to show the selected values of my "Promotion" property. This is what I have done so far but the problem is that how can I get the display names here?

```
<ul>
    @foreach (int aPromotion in @Enum.GetValues(typeof(UserPromotion)))
    {
        var currentPromotion = (int)Model.JobSeeker.Promotion;
        if ((currentPromotion & aPromotion) == aPromotion)
        {
        <li>Here I don't know how to get the display attribute of "currentPromotion".
</li>
        }
    }
</ul>
```

c# asp.net-mvc razor displayattribute

edited Apr 10 '15 at 0:55
dab
452 1 7 23

asked Oct 27 '12 at 11:38
Pejman
1,358 4 15 23

11 MVC5 does support DisplayName attribute on enums. – Bart Calixto Feb 18 '14 at 15:05

9 To be clearer: Only `System.ComponentModel.DataAnnotations.DisplayAttribute` . Not `System.ComponentModel.DisplayNameAttribute` . – kamranicus Oct 29 '15 at 2:49

1 Does this include use of reflection and therefore impact the performance? 'cos this is gonna be called a LOT of time. – Nico Jul 20 at 12:17

17 Answers

166

UPDATE

First solution was focused on getting display names from enum. Code below should be exact solution for your problem.

You can use this helper class for enums:

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Reflection;

public static class EnumHelper<T>
{
    public static IList<T> GetValues(Enum value)
    {
        var enumValues = new List<T>();

        foreach (FieldInfo fi in value.GetType().GetFields(BindingFlags.Static |
BindingFlags.Public))
        {
            enumValues.Add((T)Enum.Parse(value.GetType(), fi.Name, false));
        }
        return enumValues;
    }

    public static T Parse(string value)
    {
        return (T)Enum.Parse(typeof(T), value, true);
    }

    public static IList<string> GetNames(Enum value)
    {
        return value.GetType().GetFields(BindingFlags.Static |
BindingFlags.Public).Select(fi => fi.Name).ToList();
    }

    public static IList<string> GetDisplayValues(Enum value)
    {
        return GetNames(value).Select(obj => GetDisplayValue(Parse(obj))).ToList();
    }

    private static string lookupResource(Type resourceManagerProvider, string
resourceKey)
    {
        foreach (PropertyInfo staticProperty in
resourceManagerProvider.GetProperties(BindingFlags.Static | BindingFlags.NonPublic |
BindingFlags.Public))
        {
            if (staticProperty.PropertyType == typeof(System.Resources.ResourceManager))
            {
                System.Resources.ResourceManager resourceManager =
(System.Resources.ResourceManager)staticProperty.GetValue(null, null);
                return resourceManager.GetString(resourceKey);
            }
        }

        return resourceKey; // Fallback with the key name
    }

    public static string GetDisplayValue(T value)
    {
        var fieldInfo = value.GetType().GetField(value.ToString());

        var descriptionAttributes = fieldInfo.GetCustomAttributes(
            typeof(DisplayAttribute), false) as DisplayAttribute[];

        if (descriptionAttributes[0].ResourceType != null)
            return lookupResource(descriptionAttributes[0].ResourceType,
descriptionAttributes[0].Name);
```

```
        if (descriptionAttributes == null) return string.Empty;
        return (descriptionAttributes.Length > 0) ? descriptionAttributes[0].Name :
value.ToString();
    }
}
```

And then you can use it in your view as following:

```
<ul>
    @foreach (var value in @EnumHelper<UserPromotion>.GetValues(UserPromotion.None))
    {
         if (value == Model.JobSeeker.Promotion)
        {
            var description = EnumHelper<UserPromotion>.GetDisplayValue(value);
            <li>@Html.DisplayFor(e => description )</li>
        }
    }
</ul>
```

Hope it helps! :)

edited Aug 1 '17 at 9:10
MATT BAKER
2,123 1 19 37

answered Oct 27 '12 at 13:00
Hrvoje Stanisic
1,775 1 11 9

8 All of the answers use `.ToString` , but from stackoverflow.com/q/483794/179311, it says to use `Enum.GetName` instead. – bradlis7 Dec 30 '14 at 18:45

value.GetType().GetField(value.ToString()) was exactly what I was looking for ! – cdie Jun 13 '16 at 8:07

This answer is fine with some added null checking, but if you aren't using dotfuscation the answer at stackoverflow.com/a/4412730/852806 seems simpler. – HockeyJ Feb 20 '17 at 21:00

4 In `GetDisplayValue` you should first test `descriptionAttributes == null` before you try to access the array: `descriptionAttributes[0]` . Otherwise you may raise an exception and the line below where you check for null will never be true. – Robert S. Jul 31 '17 at 14:59

I would suggest minors changes: public static IList<T> GetValues(Enum value) could be public static IList<T> GetValues(T value). EnumHelper<T> to => public static class EnumHelper<T> where T : struct, IConvertible. Maybe static contructor? static EnumHelper() { if (!typeof(T).IsEnum) { throw new ArgumentException("T must be an enumerated type"); } } – Tom Feb 19 '18 at 8:58

One liner - Fluent syntax

148

```
public static class Extensions
{
    /// <summary>
    ///     A generic extension method that aids in reflecting
    ///     and retrieving any attribute that is applied to an `Enum`.
    /// </summary>
    public static TAttribute GetAttribute<TAttribute>(this Enum enumValue)
            where TAttribute : Attribute
    {
        return enumValue.GetType()
                        .GetMember(enumValue.ToString())
                        .First()
                        .GetCustomAttribute<TAttribute>();
    }
}
```

Example

```
public enum Season
{
   [Display(Name = "It's autumn")]
   Autumn,

   [Display(Name = "It's winter")]
   Winter,

   [Display(Name = "It's spring")]
   Spring,

   [Display(Name = "It's summer")]
   Summer
}

public class Foo
{
    public Season Season = Season.Summer;

    public void DisplayName()
    {
        var seasonDisplayName = Season.GetAttribute<DisplayAttribute>();
        Console.WriteLine("Which season is it?");
        Console.WriteLine (seasonDisplayName.Name);
    }
}
```

Output

> Which season is it?
> It's summer

edited Nov 30 '16 at 22:52

answered Aug 3 '14 at 21:04
Aydin
11.2k 2 23 38

2 Doesn't exist a definition of GetCustomAttribute – Tito Apr 29 '15 at 12:27

3 @Tito ensure that your project is targeting `.NET Framework 4.5` and that you're including the following namespaces `System.Net` `System.ComponentModel.DataAnnotations` – Aydin Apr 29 '15 at 12:32

5 using System.Reflection; using System.ComponentModel.DataAnnotations; Was needed for me. – Sinned Lolwut Jan 16 '16 at 13:32

what a terrible naming convention! – curiousBoy Feb 16 at 1:49

@curiousBoy How is `GetAttribute<TAttribute>` a terrible naming convention? It retrieves the attribute you specify and uses pascal casing as all public methods should. – Aydin Feb 17 at 2:35

120

Building on Aydin's great answer, here's an extension method that doesn't require any type parameters.

```
using System;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Reflection;

public static class EnumExtensions
{
    public static string GetDisplayName(this Enum enumValue)
    {
        return enumValue.GetType()
                        .GetMember(enumValue.ToString())
                        .First()
                        .GetCustomAttribute<DisplayAttribute>()
                        .GetName();
    }
}
```

NOTE: *GetName() should be used instead of the Name property. This ensures that the localized string will be returned if using the ResourceType attribute property.*

Example

To use it, just reference the enum value in your view.

```
@{
    UserPromotion promo = UserPromotion.SendJobOffersByMail;
}

Promotion: @promo.GetDisplayName()
```

Output

Promotion: Send Job Offers By Mail

edited May 23 '17 at 12:18
Community ♦
1 1

answered Oct 19 '14 at 21:03
Todd
9,172 3 24 24

4 Besure to add the following namespaces: using System; using System.ComponentModel.DataAnnotations; using System.Linq; using System.Reflection; – Peter Kerr Mar 11 '15 at 15:21

Slick solution, but I get {"Templates can be used only with field access, property access, single-dimension array index, or single-parameter custom indexer expressions."} – Casey Crookston Oct 2 '15 at 14:33

Looking at other SO answers for this error message (I am unfamiliar with it), it appears that you might be using this from within an Html helper method (like `@Html.DisplayFor(m => m.myEnum.GetDisplayName())` , which won't work, because they expect the evaluated expression to yield a property or something similar. You should use the bare enum value like in the example above. – Todd Oct 3 '15 at 21:09

5 I added a null reference check to the result of `GetCustomAttribute<DisplayAttribute>()` because for some Enums maybe this is not present. It falls back to `enumValue.ToString()` if the DisplayAttribute was not present. – H Dog May 26 '16 at 16:31

1 I used this to create a `List<SelectListItem>` that was populated by an Enum with all individual `DisplayAttribute.Name` annotations - this worked perfectly, thank you!! `public List<SelectListItem> MySelectListItem = new List<SelectListItem>(); foreach (MyEnum MyEnum in Enum.GetValues(typeof(MyEnum)).Cast<MyEnum>().Where(x => x != MyEnum.Default)) { MySelectListItem.Add(new SelectListItem() { Text = MyEnum.GetDisplayName(), Value = ((int)MyEnum).ToString() }); }` – Hopper Jun 9 '17 at 20:31

Based on Aydin's answer I would suggest a less "duplicatious" implementation (because we could easily get the `Type` from the `Enum`

57

value itself, instead of providing it as a parameter 😉:

```
public static string GetDisplayName(this Enum enumValue)
{
    return enumValue.GetType().GetMember(enumValue.ToString())
                   .First()
                   .GetCustomAttribute<DisplayAttribute>()
                   .Name;
}
```

EDIT (based upon @Vahagn Nahapetyan's comment)

```
public static string GetDisplayName(this Enum enumValue)
{
    return enumValue.GetType()?
                    .GetMember(enumValue.ToString())?
                    .First()?
                    .GetCustomAttribute<DisplayAttribute>()?
                    .Name;
}
```

Now we can use it very clean in this way:

```
public enum Season
{
    [Display(Name = "The Autumn")]
    Autumn,

    [Display(Name = "The Weather")]
    Winter,

    [Display(Name = "The Tease")]
    Spring,

    [Display(Name = "The Dream")]
    Summer
}

Season.Summer.GetDisplayName();
```

Which results in

> "The Dream"

edited Jun 6 '18 at 14:30

answered Feb 15 '15 at 11:34

Bernoulli IT
2,921 2 20 39

1 By far the simplest and easiest of all the answers. Thanks! – Casey Crookston Jan 11 '17 at 18:56

You should be careful with .First(). This will throw an exception for example if your enum name is "Equals" – Vahagn Nahapetyan Jun 5 '18 at 14:03

I understand the "danger" with First(). In this particular case it doesn't seem an issue. Because it is an extension method where `this` must be a valid (not null) Enum value. Otherwise calling the method would already throw (which is a responsibility of the calling code). This makes that `GetType()` will for sure provide the correct Enum Type in which `enumvalue` for sure will be a member. But GetCustomAttribute might return a null value so I provided a non-exceptional version of the method to return null when the chain of method calls has a null return value somewhere. Thanks! – Bernoulli IT Jun 6 '18 at 14:33

1 For the second variant of your code, it seems like there is no need to use null-conditional operator after GetMember because this method always returns an array of MemberInfo and never returns null. And for me it seems that it is better to use FirstOrDefault instead of just First. Then the using of null-conditional operator after FirstOrDefault will be seen consistent. – Alex34758 Jul 23 '18 at 10:27

27

If you are using MVC 5.1 or upper there is simplier and clearer way: just use data annotation (from `System.ComponentModel.DataAnnotations` namespace) like below:

```
public enum Color
{
    [Display(Name = "Dark red")]
    DarkRed,
    [Display(Name = "Very dark red")]
    VeryDarkRed,
    [Display(Name = "Red or just black?")]
    ReallyDarkRed
}
```

And in view, just put it into proper html helper:

```
@Html.EnumDropDownListFor(model => model.Color)
```

answered Sep 9 '15 at 12:38

1_bug
3,446 2 31 38

@SegmentationFault why? Can you describe your problem? Which version of .NET/MVC do you use? What error have you got? Please be more specific. – 1_bug Aug 31 '16 at 12:17

5 Because it only works for Dropdowns, not anywhere else. – Segmentation Fault Sep 1 '16 at 6:47

2 Doesn't seem to exist in .net core – Lonefish Jan 19 '17 at 15:40

3 .net core uses Html.GetEnumSelectList(typeof(YourEnum)) @Lonefish – Patrick Mcvay Mar 29 '17 at 17:20

2 if we want to use the @Html.DisplayFor(yourEnumField) we can put a Enum.cshtml in the DisplayTemplates directory (in shared directory). in this file we need to put just 2 lines. the first is: "@model Enum" the second is: "@GetDisplayName(Model)." the GetDisplayName method needs to be as in @Bernoulli IT answare – Developer May 8 '17 at 0:47

11

You could use Type.GetMember Method, then get the attribute info using reflection:

```
// display attribute of "currentPromotion"

var type = typeof(UserPromotion);
var memberInfo = type.GetMember(currentPromotion.ToString());
var attributes = memberInfo[0].GetCustomAttributes(typeof(DisplayAttribute), false);
var description = ((DisplayAttribute)attributes[0]).Name;
```

There were a few similar posts here:

Getting attributes of Enum's value

How to make MVC3 DisplayFor show the value of an Enum's Display-Attribute?

edited May 23 '17 at 10:31
Community ♦
1 1

answered Oct 27 '12 at 12:10
maximpa
1,863 10 13

6

```
<ul>
    @foreach (int aPromotion in @Enum.GetValues(typeof(UserPromotion)))
    {
        var currentPromotion = (int)Model.JobSeeker.Promotion;
        if ((currentPromotion & aPromotion) == aPromotion)
        {
        <li>@Html.DisplayFor(e => currentPromotion)</li>
        }
    }
</ul>
```

answered Oct 27 '12 at 11:54

Dmytro
8,789 22 60 113

6

Building on [Todd's great answer]() which built on [Aydin's great answer](), here's a *generic* extension method that doesn't require any type parameters.

```
/// <summary>
/// Gets human-readable version of enum.
/// </summary>
/// <returns>DisplayAttribute.Name property of given enum.</returns>
public static string GetDisplayName<T>(this T enumValue) where T : IComparable,
IFormattable, IConvertible
{
    if (!typeof(T).IsEnum)
        throw new ArgumentException("Argument must be of type Enum");

    DisplayAttribute displayAttribute = enumValue.GetType()
                                                 .GetMember(enumValue.ToString())
                                                 .First()
                                                 .GetCustomAttribute<DisplayAttribute>
();

    string displayName = displayAttribute?.GetName();

    return displayName ?? enumValue.ToString();
}
```

I needed this for my project because something like the below code, where not every member of the enum has a `DisplayAttribute`, does not work with Todd's solution:

```
public class MyClass
{
    public enum MyEnum
    {
        [Display(Name="ONE")]
        One,
        // No DisplayAttribute
        Two
    }
    public void UseMyEnum()
    {
        MyEnum foo = MyEnum.One;
        MyEnum bar = MyEnum.Two;
        Console.WriteLine(foo.GetDisplayName());
        Console.WriteLine(bar.GetDisplayName());
    }
}
// Output:
//
// ONE
// Two
```

If this is a complicated solution to a simple problem, please let me know, but this was the fix I used.

edited May 20 '18 at 22:33

answered Jun 16 '17 at 19:15
Sinjai
467 4 21

4

You need to use a bit of reflection in order to access that attribute:

```
var type = typeof(UserPromotion);
var member = type.GetMember(Model.JobSeeker.Promotion.ToString());
var attributes = member[0].GetCustomAttributes(typeof(DisplayAttribute), false);
var name = ((DisplayAttribute)attributes[0]).Name;
```

I recommend wrapping this method in a extension method or perform this in a view model.

answered Oct 27 '12 at 11:49
alexn
43.9k 11 99 136

3

I'm sorry to do this, but I couldn't use any of the other answers as-is and haven't time to duke it out in the comments.

Uses C# 6 syntax.

```
static class EnumExtensions
{
    /// returns the localized Name, if a [Display(Name="Localised Name")] attribute is
applied to the enum member
    /// returns null if there isnt an attribute
    public static string DisplayNameOrEnumName(this Enum value)
    // => value.DisplayNameOrDefault() ?? value.ToString()
    {
        // More efficient form of ^ based on http://stackoverflow.com/a/17034624/11635
        var enumType = value.GetType();
        var enumMemberName = Enum.GetName(enumType, value);
        return enumType
            .GetEnumMemberAttribute<DisplayAttribute>(enumMemberName)
            ?.GetName() // Potentially localized
            ?? enumMemberName; // Or fall back to the enum name
    }

    /// returns the localized Name, if a [Display] attribute is applied to the enum
member
    /// returns null if there is no attribute
    public static string DisplayNameOrDefault(this Enum value) =>
        value.GetEnumMemberAttribute<DisplayAttribute>()?.GetName();

    static TAttribute GetEnumMemberAttribute<TAttribute>(this Enum value) where
TAttribute : Attribute =>
        value.GetType().GetEnumMemberAttribute<TAttribute>(value.ToString());

    static TAttribute GetEnumMemberAttribute<TAttribute>(this Type enumType, string
enumMemberName) where TAttribute : Attribute =>
        enumType.GetMember(enumMemberName).Single().GetCustomAttribute<TAttribute>();
}
```

edited Feb 8 '16 at 16:06

answered Feb 8 '16 at 15:46

Ruben Bartelink
46.7k 17 150 206

2

Building further on Aydin's and Todd's answers, here is an extension method that also lets you get the name from a resource file

```
using AppResources;
using System;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Reflection;
using System.Resources;

public static class EnumExtensions
{
    public static string GetDisplayName(this Enum enumValue)
    {
        var enumMember= enumValue.GetType()
                        .GetMember(enumValue.ToString());

        DisplayAttribute displayAttrib = null;
        if (enumMember.Any()) {
            displayAttrib = enumMember
                        .First()
                        .GetCustomAttribute<DisplayAttribute>();
        }

        string name = null;
        Type resource = null;

        if (displayAttrib != null)
        {
            name = displayAttrib.Name;
            resource = displayAttrib.ResourceType;
        }

        return String.IsNullOrEmpty(name) ? enumValue.ToString()
            : resource == null ?  name
            : new ResourceManager(resource).GetString(name);
    }
}
```

and use it like

```
public enum Season
{
    [Display(ResourceType = typeof(Resource), Name = Season_Summer")]
    Summer
}
```

edited Jul 9 '15 at 16:31

answered Mar 11 '15 at 15:49

Peter Kerr
974 13 24

I'm trying to get this working for my project but I get an error with the "new ResourceManager(resource).GetString(name);" line. I had asked a question (stackoverflow.com/questions/31319251/…) and I was sent here. When I view the "ResourceManager(resource)" while running it returns "Resources.Enums.resource". Any help would be greatly appreciated. Thank you! – Karinne Jul 9 '15 at 14:32

What error message do you get? – Peter Kerr Jul 9 '15 at 16:28

Updated the code to better handle nulls when you don't have Display Name set for some of the enum values - might help – Peter Kerr Jul 9 '15 at 16:32

That still didn't work. I updated my question on stackoverflow.com/questions/31319251/… with the error message. Thanks for the help! – Karinne Jul 10 '15 at 11:48

2

With Core 2.1,

```
public static string GetDisplayName(Enum enumValue)
{
  return enumValue.GetType()?
 .GetMember(enumValue.ToString())?[0]?
 .GetCustomAttribute<DisplayAttribute>()?
 .Name;
}
```

answered Aug 27 '18 at 11:35

Deniz aydın

31 2

2

combining all edge-cases together from above:

- enum members with base object members' names (`Equals` , `ToString`)
- optional `Display` attribute

here is my code:

```
public enum Enum
{
    [Display(Name = "What a weird name!")]
    ToString,

    Equals
}
```

```
public static class EnumHelpers
{
    public static string GetDisplayName(this Enum enumValue)
    {
        var enumType = enumValue.GetType();

        return enumType
                .GetMember(enumValue.ToString())
                .Where(x => x.MemberType == MemberTypes.Field &&
((FieldInfo)x).FieldType == enumType)
                .First()
                .GetCustomAttribute<DisplayAttribute>()?.Name ?? enumValue.ToString();
    }
}

void Main()
{
    Assert.Equals("What a weird name!", Enum.ToString.GetDisplayName());
    Assert.Equals("Equals", Enum.Equals.GetDisplayName());
}
```

answered Jul 18 at 15:23

avs099
7,986 4 46 97

0

Based on previous answers I've created this comfortable helper to support all DisplayAttribute properties in a readable way:

```
public static class EnumExtensions
    {
        public static DisplayAttributeValues GetDisplayAttributeValues(this Enum
enumValue)
        {
            var displayAttribute =
enumValue.GetType().GetMember(enumValue.ToString()).First().GetCustomAttribute<DisplayAttr
();

            return new DisplayAttributeValues(enumValue, displayAttribute);
        }

        public sealed class DisplayAttributeValues
        {
            private readonly Enum enumValue;
            private readonly DisplayAttribute displayAttribute;
            public DisplayAttributeValues(Enum enumValue, DisplayAttribute
displayAttribute)
            {
                this.enumValue = enumValue;
                this.displayAttribute = displayAttribute;
            }

            public bool? AutoGenerateField =>
this.displayAttribute?.GetAutoGenerateField();
            public bool? AutoGenerateFilter =>
this.displayAttribute?.GetAutoGenerateFilter();
            public int? Order => this.displayAttribute?.GetOrder();
            public string Description => this.displayAttribute != null ?
this.displayAttribute.GetDescription() : string.Empty;
            public string GroupName => this.displayAttribute != null ?
this.displayAttribute.GetGroupName() : string.Empty;
            public string Name => this.displayAttribute != null ?
this.displayAttribute.GetName() : this.enumValue.ToString();
            public string Prompt => this.displayAttribute != null ?
this.displayAttribute.GetPrompt() : string.Empty;
            public string ShortName => this.displayAttribute != null ?
this.displayAttribute.GetShortName() : this.enumValue.ToString();
        }
    }
```

answered Oct 12 '15 at 23:09

Kryszal
1,258 12 19

0

I tried doing this as an edit but it was rejected; I can't see why.

The above will throw an exception if you call it with an Enum that has a mix of custom attributes and plain items, e.g.

```
public enum CommentType
{
    All = 1,
    Rent = 2,
    Insurance = 3,
    [Display(Name="Service Charge")]
    ServiceCharge = 4
}
```

So I've modified the code ever so slightly to check for custom attributes before trying to access them, and use the name if none are found.

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Reflection;

public static class EnumHelper<T>
{
    public static IList<T> GetValues(Enum value)
    {
        var enumValues = new List<T>();

        foreach (FieldInfo fi in value.GetType().GetFields(BindingFlags.Static |
BindingFlags.Public))
        {
            enumValues.Add((T)Enum.Parse(value.GetType(), fi.Name, false));
        }
        return enumValues;
    }

    public static T Parse(string value)
    {
        return (T)Enum.Parse(typeof(T), value, true);
    }

    public static IList<string> GetNames(Enum value)
    {
        return value.GetType().GetFields(BindingFlags.Static |
BindingFlags.Public).Select(fi => fi.Name).ToList();
    }

    public static IList<string> GetDisplayValues(Enum value)
    {
        return GetNames(value).Select(obj => GetDisplayValue(Parse(obj))).ToList();
    }

    private static string lookupResource(Type resourceManagerProvider, string
resourceKey)
    {
        foreach (PropertyInfo staticProperty in
resourceManagerProvider.GetProperties(BindingFlags.Static | BindingFlags.NonPublic |
BindingFlags.Public))
        {
            if (staticProperty.PropertyType == typeof(System.Resources.ResourceManager))
            {
                System.Resources.ResourceManager resourceManager =
(System.Resources.ResourceManager)staticProperty.GetValue(null, null);
                return resourceManager.GetString(resourceKey);
            }
        }

        return resourceKey; // Fallback with the key name
    }

    public static string GetDisplayValue(T value)
    {
        var fieldInfo = value.GetType().GetField(value.ToString());

        var descriptionAttributes = fieldInfo.GetCustomAttributes(
            typeof(DisplayAttribute), false) as DisplayAttribute[];

        if (descriptionAttributes.Any() && descriptionAttributes[0].ResourceType !=
null)
            return lookupResource(descriptionAttributes[0].ResourceType,
descriptionAttributes[0].Name);

        if (descriptionAttributes == null) return string.Empty;
        return (descriptionAttributes.Length > 0) ? descriptionAttributes[0].Name :
value.ToString();
    }
}
```

answered Apr 25 '18 at 14:20

Red

1,184 8 27

0

Using MVC5 you could use:

```
public enum UserPromotion
{
   None = 0x0,

   [Display(Name = "Send Job Offers By Mail")]
   SendJobOffersByMail = 0x1,

   [Display(Name = "Send Job Offers By Sms")]
   SendJobOffersBySms = 0x2,

   [Display(Name = "Send Other Stuff By Sms")]
```

```
    SendPromotionalBySms = 0x4,

    [Display(Name = "Send Other Stuff By Mail")]
    SendPromotionalByMail = 0x8
 }
```

then if you want to create a dropdown selector you can use:

```
@Html.EnumDropdownListFor(expression: model => model.PromotionSelector, optionLabel:
"Select")
```

answered Apr 8 at 13:48

M.Hazara
90 7

0

I want to contribute with culture-dependent GetDisplayName enum extension. Hope this will be usefull for anyone googling this answer like me previously:

"standart" way as Aydin Adn and Todd mentioned:

```
    public static string GetDisplayName(this Enum enumValue)
    {
        return enumValue
            .GetType()
            .GetMember(enumValue.ToString())
            .First()
            .GetCustomAttribute<DisplayAttribute>()
            .GetName();
    }
```

"Culture-dependent" way:

```
    public static string GetDisplayName(this Enum enumValue, CultureInfo ci)
    {
        var displayAttr = enumValue
            .GetType()
            .GetMember(enumValue.ToString())
            .First()
            .GetCustomAttribute<DisplayAttribute>();
        var resMan = displayAttr.ResourceType?.GetProperty(@"ResourceManager",
BindingFlags.Static | BindingFlags.Public | BindingFlags.NonPublic).GetValue(null, null)
as ResourceManager;

        return resMan?.GetString(displayAttr.Name, ci) ?? displayAttr.GetName();
    }
```

edited Sep 11 '18 at 14:09

answered Sep 11 '18 at 14:00

Pavel
188 2 9

protected by Community ♦ May 15 '15 at 6:39

Thank you for your interest in this question. Because it has attracted low-quality or spam answers that had to be removed, posting an answer now requires 10 reputation on this site (the association bonus does not count).

Would you like to answer one of these unanswered questions instead?