How to get C# Enum description from value? [duplicate]

Asked 9 years, 6 months ago Active 4 months ago Viewed 403k times

362

83

Possible Duplicate:
Getting attributes of Enum's value

I have an enum with Description attributes like this:

```
public enum MyEnum
{
    Name1 = 1,
    [Description("Here is another")]
    HereIsAnother = 2,
    [Description("Last one")]
    LastOne = 3
}
```

I found this bit of code for retrieving the description based on an Enum

```
public static string GetEnumDescription(Enum value)
{
    FieldInfo fi = value.GetType().GetField(value.ToString());

    DescriptionAttribute[] attributes =
fi.GetCustomAttributes(typeof(DescriptionAttribute), false) as DescriptionAttribute[];

    if (attributes != null && attributes.Any())
    {
        return attributes.First().Description;
    }

    return value.ToString();
}
```

This allows me to write code like:

```
var myEnumDescriptions = from MyEnum n in Enum.GetValues(typeof(MyEnum))
                         select new { ID = (int)n, Name =
Enumerations.GetEnumDescription(n) };
```

What I want to do is if I know the enum value (e.g. 1) - how can I retrieve the description? In other words, how can I convert an integer into an "Enum value" to pass to my GetDescription method?

c# enums

edited Jun 20 at 11:47
Thomas M. Krystyan
3 2

asked Apr 16 '10 at 1:44
davekaro
2,733 2 25 22

marked as duplicate by nawfal, Frank van Puffelen, Perception, Frank Shearar, Sjoerd Jan 28 '13 at 15:07

This question has been asked before and already has an answer. If those answers do not fully address your question, please ask a new question.

1 (attributes != null) will always be true and else is redundant. – Jeff Jul 17 '14 at 23:00

2 namespace required for Description is System.ComponentModel – John M Dec 21 '15 at 15:10

Try this solution codereview.stackexchange.com/questions/157871/... – Developer Sep 5 '18 at 8:26

5 Answers

341

```
int value = 1;
string description = Enumerations.GetEnumDescription((MyEnum)value);
```

The default underlying data type for an `enum` in C# is an `int`, you can just cast it.

answered Apr 16 '10 at 1:48
Nicholas Piasecki
22.1k 4 69 87

75 Why am I not finding any Enumerations class in the .Net framework? – Spencer Ruport Dec 26 '13 at 17:56

74 The Enumerations class is something that the person who asked the question wrote himself, and the GetEnumDescription() function is in the question. – Nicholas Piasecki Dec 29 '13 at 17:03

12 A better approach could be to use extension method instead. To do this, convert `Enumerations.GetEnumDescription` 's `Enum value` parameter to `this Enum value` and then call it like `string description = ((MyEnum)value).GetEnumDescription()` – gkc Nov 25 '14 at 14:09

6 string description=Enum.GetName(typeof(MyEnum), value); – atik sarker Dec 14 '14 at 9:09

3 It's in the question. It's also in the comment above, where I say it's in the question. Just read the question, and there you are. – Nicholas Piasecki Aug 3 '16 at 2:12

88

Update

The Unconstrained Melody library is no longer maintained; Support was dropped in favour of Enums.NET.

In Enums.NET you'd use:

```
string description = ((MyEnum)value).AsString(EnumFormat.Description);
```

Original post

I implemented this in a generic, type-safe way in Unconstrained Melody - you'd use:

```
string description = Enums.GetDescription((MyEnum)value);
```

This:

- Ensures (with generic type constraints) that the value really is an enum value
- Avoids the boxing in your current solution
- Caches all the descriptions to avoid using reflection on every call
- Has a bunch of other methods, including the ability to parse the value from the description

I realise the core answer was just the cast from an `int` to `MyEnum` , but if you're doing a lot of enum work it's worth thinking about using Unconstrained Melody :)

edited Dec 17 '18 at 17:37
sclarke81
1,405 15 20

answered Apr 16 '10 at 2:53
Jon Skeet
1142k 721 8243 8622

Isn't "value" an int? So, doesn't Enums.GetDescription((MyEnum)value) just cast the int to MyEnum? – davekaro Apr 16 '10 at 12:20

@davekaro: It casts the int to MyEnum - but you wouldn't be able to call it with any non-enum, including an "Enum" reference. Basically it's like your code, but with some generics magic. – Jon Skeet Apr 16 '10 at 13:06

1 @JonSkeet I don't see this method in Enums.cs in code.google.com/p/unconstrained-melody/downloads/… – tom Sep 13 '13 at 19:14

2 @tom: It may not be in the latest "released" version, but it's in the source: code.google.com/p/unconstrained-melody/source/browse/trunk/… – Jon Skeet Sep 14 '13 at 8:15

1 @AlexZhukovskiy: I suspect so, although I haven't tried. You may need to use `import="dnxcore450"` or whatever. When .NET Core 1.0 ships, I'll try to remember to update the package to ensure it works with netstandard1.0. – Jon Skeet May 27 '16 at 10:48

77

I put the code together from the accepted answer in a generic extension method, so it could be used for all kinds of objects:

```
public static string DescriptionAttr<T>(this T source)
{
    FieldInfo fi = source.GetType().GetField(source.ToString());

    DescriptionAttribute[] attributes = (DescriptionAttribute[])fi.GetCustomAttributes(
        typeof(DescriptionAttribute), false);

    if (attributes != null && attributes.Length > 0) return attributes[0].Description;
    else return source.ToString();
}
```

Using an enum like in the original post, or any other class whose property is decorated with the Description attribute, the code can be consumed like this:

```
string enumDesc = MyEnum.HereIsAnother.DescriptionAttr();
string classDesc = myInstance.SomeProperty.DescriptionAttr();
```

answered Jun 11 '12 at 20:02

Irish
937 6 8

4 string classDesc = myInstance.SomeProperty.DescriptionAttr(); That will not work! Let say you have class Test { public int TestInt {get; set;} }. So if you will call new Test().TestInt.DescriptionAttr() you will get null reference exception - 0.GetType().GetField("0") – Vladimirs Dec 12 '13 at 13:34

29

To make this easier to use, I wrote a generic extension:

```
public static string ToDescription<TEnum>(this TEnum EnumValue) where TEnum : struct
{
    return Enumerations.GetEnumDescription((Enum)(object)((TEnum)EnumValue));
}
```

now I can write:

```
        MyEnum my = MyEnum.HereIsAnother;
        string description = my.ToDescription();
        System.Diagnostics.Debug.Print(description);
```

Note: replace "Enumerations" above with your class name

answered Dec 5 '11 at 14:57

David
1,229 13 22

7

You can't easily do this in a generic way: you can only convert an integer to a specific type of enum. As Nicholas has shown, this is a trivial cast if you only care about one kind of enum, but if you want to write a generic method that can handle different kinds of enums, things get a bit more complicated. You want a method along the lines of:

```
public static string GetEnumDescription<TEnum>(int value)
{
  return GetEnumDescription((Enum)((TEnum)value));  // error!
}
```

but this results in a compiler error that "int can't be converted to TEnum" (and if you work around this, that "TEnum can't be converted to Enum"). So you need to fool the compiler by inserting casts to object:

```
public static string GetEnumDescription<TEnum>(int value)
{
  return GetEnumDescription((Enum)(object)((TEnum)(object)value));  // ugly, but works
}
```

You can now call this to get a description for whatever type of enum is at hand:

```
GetEnumDescription<MyEnum>(1);
GetEnumDescription<YourEnum>(2);
```

answered Apr 16 '10 at 1:57

itowlson

67.3k 12 143 145

How is "GetEnumDescription<MyEnum>(1);" any better than GetEnumDescription((MyEnum)1); ? – davekaro Apr 16 '10 at 2:09

@davekaro: Implemented like this, it's not all that much better, but a more robust implementation based on generics could do this without the explicit cast, so you don't risk unhandled exceptions if the number doesn't actually match any of the enum values. – Aaronaught Apr 16 '10 at 2:15

Interesting. Just to clarify for future readers: One is not going to get an unhandled exception on an explicit cast if the number doesn't match one of the enum values (you could say "MyEnum value = (MyEnum)5;" and that line will execute just fine, but you would bomb in the first line of GetEnumDescription() as implemented in the original question (because GetField() will return null as it can find no matching field with that value). (To guard against that, we'd need to check Enum.IsDefined() first and return null or an empty string, or just throw an ArgumentOutOfRangeException ourselves.) – Nicholas Piasecki Apr 16 '10 at 2:48