How to loop through all enum values in C#? [duplicate]

Ask Question

1249

192

This question already has an answer here:
How do I enumerate an enum in C#? *26 answers*

```
public enum Foos
{
    A,
    B,
    C
}
```

Is there a way to loop through the possible values of `Foos` ?

Basically?

```
foreach(Foo in Foos)
```

c# .net enums language-features

edited Sep 14 '18 at 11:22
Cœur
19.3k 10 116 155

asked Jun 9 '09 at 20:25
divinci
7,854 10 37 53

marked as duplicate by gliderkite, Andrew Whitaker, Jon Lin, Jehof, j0k Oct 23 '12 at 7:21

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.

8 Answers

1760

Yes you can use the `GetValues` method:

```
var values = Enum.GetValues(typeof(Foos));
```

Or the typed version:

```
var values = Enum.GetValues(typeof(Foos)).Cast<Foos>();
```

I long ago added a helper function to my private library for just such an occasion:

```
public static class EnumUtil {
    public static IEnumerable<T> GetValues<T>() {
        return Enum.GetValues(typeof(T)).Cast<T>();
    }
}
```

Usage:

```
var values = EnumUtil.GetValues<Foos>();
```

edited Mar 10 at 0:47

Evgeni Sergeev
12.5k 14 77 93

answered Jun 9 '09 at 20:28

JaredPar
583k 121 1077 1354

128 You can cast the array directly: `(T[])Enum.GetValues(typeof(T))` – Şafak Gür Nov 22 '12 at 9:51

38 The good thing about @ŞafakGür's comment is that (1) you don't have to go through an extra iterator (`.Cast<Foos>`), and (2) you don't need to box all the values and unbox them again. Şafak's cast will remain valid as long as they don't change the array type returned to some other type (like `object[]`). But we can be completely sure they won't because (a) it would lose performance, (b) there are already millions of codelines using Şafak's cast, and they would all break with a runtime exception. – Jeppe Stig Nielsen Apr 15 '13 at 17:38

2 Of course, how many enums are going to contain more than a dozen or two values? I imagine that in most cases boxing/unboxing is a negligible hit, so the cleanest solution is the highest priority. – Jon Coombs Mar 24 '14 at 16:58

10 @JCoombs I find this clean enough: `public static IReadOnlyList<T> GetValues<T>() { return (T[])Enum.GetValues(typeof(T)); }` . But yeah, performance difference is negligible in common usage. I just don't like the idea of creating an iterator when I already have an iterable (enumerable) object to return. – Şafak Gür Jul 31 '14 at 8:33

7 Unfortunately, this does not answer the question posed. The question was how to loop through the values of an enum. SLaks answered the question. – JAB Sep 10 '14 at 15:22

6

Yes. Use `GetValues()` method in `System.Enum` class.

edited Apr 9 '18 at 21:24

Lonely Neuron
3,060 3 18 34

answered Jun 9 '09 at 20:28

Pablo Santa Cruz

136k 27 202 254

23

```
static void Main(string[] args)
{
    foreach (int value in Enum.GetValues(typeof(DaysOfWeek)))
    {
        Console.WriteLine(((DaysOfWeek)value).ToString());
    }

    foreach (string value in Enum.GetNames(typeof(DaysOfWeek)))
    {
        Console.WriteLine(value);
    }
    Console.ReadLine();
}

public enum DaysOfWeek
{
    monday,
    tuesday,
    wednesday
}
```

edited Mar 8 '17 at 15:01

Tshilidzi Mudau

2,871 23 34

answered Jun 9 '09 at 20:32

dbones

3,109 2 27 45

UPDATED

Some time on, I see a comment that brings me back to my old

32

answer, and I think I'd do it differently now. These days I'd write:

```
private static IEnumerable<T> GetEnumValues<T>()
{
    // Can't use type constraints on value types, so have to do checi
    if (typeof(T).BaseType != typeof(Enum))
    {
        throw new ArgumentException("T must be of type System.Enum")
    }

    return Enum.GetValues(typeof(T)).Cast<T>();
}
```

edited Feb 4 '13 at 10:48

answered Jun 9 '09 at 20:30

Neil Barnwell

29.3k 24 130 202

1 Why is using LINQ "more correct"? Please c.f. `You can cast the array directly: (T[])Enum.GetValues(typeof(T))` @SafakGür, this version has less overhead IMO. – Sebastian Sep 22 '13 at 8:24

8 make it simple GetEnumValues<T>() where T : Enum – Saboor Awan Apr 3 '14 at 10:27

1 @SaboorAwan is not possible to use System.Enum as a type parameter constraint. Compiler says: `Constraint cannot be special class 'Enum'` – Carlos Muñoz Mar 9 '16 at 17:26

Yes that's why I have that comment and type checking thing in my implementation; I'd already thought of that. :) – Neil Barnwell Mar 16 '16 at 21:03

1 Quick note. In C# 7.3 you can now use `Enum` (as well as `unmanaged` and `delegate`) as generic constraints. – WBuck May 23 '18 at 12:52

723

```
foreach(Foos foo in Enum.GetValues(typeof(Foos)))
```

edited Dec 1 '11 at 11:27

answered Jun 9 '09 at 20:28

SLaks
693k 140 1653 1774

4 This is a great solution. By using "Foos" instead of "var" the type inference system was able to use the right version of GetValues which returned the correct object type. Nice! – Robert Patterson Apr 23 '16 at 17:06

3 @RobertPatterson By using `Foos` nothing is magically inferred. It is an explicit cast. – Chiel ten Brinke Jun 16 '17 at 13:35

2 @daveD I'd like to think people can handle writing a foreach block on their own. – Sinjai Aug 21 '17 at 16:23

53

```
foreach (Foos foo in Enum.GetValues(typeof(Foos)))
{
    ...
}
```

edited Jun 9 '09 at 20:43

answered Jun 9 '09 at 20:29

adrianbanks
67.1k 18 147 185

8

```
Enum.GetValues(typeof(Foos))
```

answered Jun 9 '09 at 20:30

Vasu Balakrishnan
1,610 9 12

108

```
foreach (EMyEnum val in Enum.GetValues(typeof(EMyEnum)))
{
   Console.WriteLine(val);
}
```

Credit to Jon Skeet here: http://bytes.com/groups/net-c/266447-how-loop-each-items-enum

answered Jun 9 '09 at 20:28

Inisheer
17k 9 43 81