How to create an array of enums

Asked 9 years, 4 months ago Active 9 years, 4 months ago Viewed 45k times

9

I have about 30 different flagged enums that I would like to put into an array for indexing and quick access. Let me also claify that I do not have 1 enum with 30 values but I have 30 enums with differing amounts of values.

The goal would be to add them to an array at a specifed index. This way I can write a functions in which I can pass the array index into for setting particuler values of the enum.

Updated: Here is an example of what I am wanting to do.

enum main(enum1 = 0, enum2 = 1, enumn = n-1) - this has indexs which would match the index of the associated enum

[flag] enum1(value1=0, value2=1, value3=2, value4=4...)

[flag] enum2("")

[flag] enum2("")

since I am using flagable enums I have a class like the following

```
public static class CEnumWorker
{
   public static enum1 myEnum1 = enum1.value1;
   public static enum2 myEnum2 = enum2.value1;
   public static enumN myEnumN = enumN.value1;

   //I would then have functions that set the flags on the enums. I would like to access
the enums through an array or other method so that I do not have to build a large switch
statement to know which enum I am wanting to manipulate
}
```

c# arrays enums

edited Jul 1 '10 at 19:05

asked Jul 1 '10 at 17:37
user381347
48 1 1 4

10 Maybe a code sample would help...but this doesn't make one bit of sense to me. – Justin Niessner Jul 1 '10 at 17:39

3 why do you wish to put different types in the same array? – fbstj Jul 1 '10 at 17:40

2 "For indexing and quick accessing"? By whom or by what? Do you want to put a lot of integers, strings and floats into an array for indexing and quick access? Why do you want to treat these enums as related entities? I strongly suspect that whatever you want to do, tossing enums into an array is the wrong solution. – Pontus Gagge Jul 1 '10 at 17:54

1 this is the lamest idea ever and you completely miss the point of enums. It sounds like you need a Dictionary, not a list of enums. – rochal Jul 1 '10 at 17:55

4 Answers

19

Since you have 30 different types of enums, you can't create a strongly typed array for them. The best you could do would be an array of System.Enum:

```
Enum[] enums = new Enum[] { enum1.Value1, enum2.Value2, etc };
```

You would then have to cast when pulling an enum out of the array if you need the strongly typed enum value.

edited Jul 1 '10 at 20:35

answered Jul 1 '10 at 17:42
Wesley Wiser
7,280 4 36 56

There's really no benefit to this versus typing the array as `object` . `ValueType` is (unintuitively) a reference type, so the values will still be boxed. – Adam Robinson Jul 1 '10 at 17:44

1 Thus, why don't directly Enum[] ? – digEmAll Jul 1 '10 at 17:45

@digEmAll - you would still have to cast, but at least the array couldn't contain anything that isn't an enum. – Nelson Rothermel Jul 1 '10 at 17:49

@Nelson: Yes, cast is unavoidable (in fact I still don't get the reason for such kind of array) but at least you have a bit more compile-time safeness than an object array... – digEmAll Jul 1 '10 at 17:54

I must have been having a really big brain fart as I looked into using Enum[] a couple of times today and just never fully worked it out. Thank you for all yoru help. – user381347 Jul 1 '10 at 19:32

2

If I understand correctly, you would have to do:

```
object[] enums = new object[30];
enums[0] = Enum1.Value1;
enums[1] = Enum2.AnotherValue;
```

But then you would have to access like this (not strongly typed, and easy to reference the wrong index):

```
if ((Enum1)enums[0] == Enum1.Value1)
...
```

In .NET 4, you could use the Tuple:

```
var enums = new Tuple<Enum1, Enum2>(Enum1.Value1, Enum2.AnotherValue);

if (enums.Item1 == Enum1.Value1)
...
```

...but I wouldn't recommend it for so many (30) enums, and I think there's even a limit of 8 or so. You can also create your own class:

```
class Enums
{
  public Enum1 Enum1 { get; set; }
  public Enum2 Enum2 { get; set; }
}

Enums enums = new Enums();
enums.Enum1 = Enum1.Value1;
enums.Enum2 = Enum2.AnotherValue;

if (enums.Enum1 == Enum1.Value1)
...
```

I would recommend the last way because you're using a reference type, you're not dependent on index position, you can give the properties whatever name you want, and it's strongly typed. The only thing you "lose" is the ability to easily iterate through the list, but you can achieve that with Reflection if you really need to.

edited Jul 1 '10 at 17:52 answered Jul 1 '10 at 17:47

Nelson Rothermel
5,507 8 51 75

Mabye make it an struct instead of a class. – Dykam Jul 1 '10 at 17:55

There is a limit of 8 generic parameters in the tuple type. By convention the last generic parameter is a tuple containing the addional values so you could create a tuple with more than 8 using something like: var a = new Tuple<int, int, int, int, int, int, int, Tuple<int, int>>(); but that wouldn't be very nice to deal with. – Wesley Wiser Jul 1 '10 at 17:55

@Dykam: A struct is generally used for *one* value to prevent a big object with multiple copies. If you had a few enums, maybe, but I think 30 enums should really be a reference type. – Nelson Rothermel Jul 1 '10 at 17:57

@wawa: I hadn't thought about that... :) It's like having deeply nested arrays of arrays. Ick! – Nelson Rothermel Jul 1 '10 at 17:58

1 @Nelson: yeah its pretty disgusting unless your using a language like F# whose compiler takes care of that for you. – Wesley Wiser Jul 1 '10 at 18:04

0

Enum provides two mechanisms for converting an integer to an enum value - the GetValues() method and plain casting:

```
enum EnumA { A1, A2, A1234 }
enum EnumB { B00, B01, B02, B04 }

class Program
{
    static void Main(string[] args)
    {
        EnumA a = ((EnumA[])Enum.GetValues(typeof(EnumA)))[0];

        Console.WriteLine(a);

        EnumB boa = (EnumB)3;

        Console.WriteLine(boa);
    }
}
```

I'm not quite sure why you want to create your own arrays if you can use one of these mechanisms to get an enum from an int.

answered Jul 1 '10 at 17:56
Pete Kirkham
44.1k 4 82 153

0

You could always use good old `object[]` , but that means doing a lot of casting.

answered Jul 1 '10 at 17:40

Joel Coehoorn
323k 100 510 744