

Meet The Overflow, a newsletter by developers, for developers. Fascinating questions, illuminating answers, and entertaining links from around the web. [Learn more](#)

How to enumerate an enum?

Asked 11 years ago Active yesterday Viewed 713k times

▲ How can you enumerate an `enum` in C#?

3574 E.g. the following code does not compile:

▼

★
622

```
public enum Suit
{
    Spades,
    Hearts,
    Clubs,
    Diamonds
}

public void EnumerateAllSuitsDemoMethod()
{
    foreach (Suit suit in Suit)
    {
        DoSomething(suit);
    }
}
```

And gives the following compile-time error:

'Suit' is a 'type' but is used like a 'variable'

It fails on the `suit` keyword, the second one.

c# .net loops enums enumeration

edited Sep 24 at 15:07

asked Sep 19 '08 at 20:24



By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).

13 See also ... stackoverflow.com/questions/972307/... – [SteveC](#) Aug 4 '09 at 14:10

1 You may want to check out [the ins and outs of C# enums](#), which discusses this as well as other useful enum tidbits – [ChaseMedallion](#) May 14 '18 at 12:53

1 LOL had EXACTLY the same question (guess you want to do sth with poker?) :D – [julian bechtold](#) Feb 5 at 10:49

27 Answers

4403  

```
foreach (Suit suit in (Suit[]) Enum.GetValues(typeof(Suit)))
{
}
```

Note: The cast to `(Suit[])` is not strictly necessary, [but does make the code 0.5 ns faster](#).

edited Feb 5 at 15:18



[Ian Boyd](#)

129k

203

725

1041

answered Sep 19 '08 at 20:37



[jop](#)

70.3k

10

50

52

92 This doesn't work if you have duplicate values in the enumerator list. – [Jessy](#) Jun 17 '12 at 3:50

10 I just want to point out that this, unfortunately won't work in silverlight, since the silverlight library don't comprise `enum.GetValues`. You have to use reflection in this case. – [Giacomo Tagliabue](#) Oct 17 '13 at 5:50

136 @Jessy this *does* work in case of duplicate situations like `enum E {A = 0, B = 0}`. `Enum.GetValues` results in two values being returned, though they are the same. `E.A == E.B` is true, so there is not distinction. If you want individual names, then you should look for `Enum.GetNames`. – [nawfal](#) Nov 7 '13 at 9:25

13 Then if you have duplicates/synonyms in your enum, and you want the other behavior, you can use Linq's `Distinct` extension (since .NET 3.5), so `foreach (var suit in ((Suit[])Enum.GetValues(typeof(Suit))).Distinct()) { }`. – [Jeppe Stig Nielsen](#) Jun 12 '14 at 8:46

39 I made the mistake of trying to use `var` for the type. The compiler will make the variable an `object` instead of the enum. List the enum type explicitly. – [jpmc26](#) Jan 8 '16 at 22:57

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).

660



```
public enum Suits
{
    Spades,
    Hearts,
    Clubs,
    Diamonds,
    NumSuits
}

public void PrintAllSuits()
{
    foreach (string name in Enum.GetNames(typeof(Suits)))
    {
        System.Console.WriteLine(name);
    }
}
```

By the way, incrementing the value is not a good way to enumerate the values of an enum. You should do this instead.

I would use `Enum.GetValues(typeof(Suit))` instead.

```
public enum Suits
{
    Spades,
    Hearts,
    Clubs,
    Diamonds,
    NumSuits
}

public void PrintAllSuits()
{
    foreach (var suit in Enum.GetValues(typeof(Suits)))
    {
        System.Console.WriteLine(suit.ToString());
    }
}
```

Navigation icons: back, forward, search, etc.

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).

VB Syntax here: [link](#) – AndruWitta Sep 9 '18 at 2:11 

I took your version with a small following changes from my side: `Enum.GetValues(typeof(Suits)).OfType<Suits>().ToArray()` . In that case I can iterate array of `Suits` enum items, not strings. – Barabas Jun 24 at 12:57 

I made some extensions for easy enum usage, maybe someone can use it...

327

```
public static class EnumExtensions
{
    /// <summary>
    /// Gets all items for an enum value.
    /// </summary>
    /// <typeparam name="T"></typeparam>
    /// <param name="value">The value.</param>
    /// <returns></returns>
    public static IEnumerable<T> GetAllItems<T>(this Enum value)
    {
        foreach (object item in Enum.GetValues(typeof(T)))
        {
            yield return (T)item;
        }
    }

    /// <summary>
    /// Gets all items for an enum type.
    /// </summary>
    /// <typeparam name="T"></typeparam>
    /// <param name="value">The value.</param>
    /// <returns></returns>
    public static IEnumerable<T> GetAllItems<T>() where T : struct
    {
        foreach (object item in Enum.GetValues(typeof(T)))
        {
            yield return (T)item;
        }
    }

    /// <summary>
    /// Gets all combined items from an enum value.
    /// </summary>
```

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).

```

/// <example>
/// Displays ValueA and ValueB.
/// <code>
/// EnumExample dummy = EnumExample.Combi;
/// foreach (var item in dummy.GetAllSelectedItems<EnumExample>())
/// {
///     Console.WriteLine(item);
/// }
/// </code>
/// </example>
public static IEnumerable<T> GetAllSelectedItems<T>(this Enum value)
{
    int valueAsInt = Convert.ToInt32(value, CultureInfo.InvariantCulture);

    foreach (object item in Enum.GetValues(typeof(T)))
    {
        int itemAsInt = Convert.ToInt32(item, CultureInfo.InvariantCulture);

        if (itemAsInt == (valueAsInt & itemAsInt))
        {
            yield return (T)item;
        }
    }
}

/// <summary>
/// Determines whether the enum value contains a specific value.
/// </summary>
/// <param name="value">The value.</param>
/// <param name="request">The request.</param>
/// <returns>
///     <c>true</c> if value contains the specified value; otherwise, <c>false</c>.
/// </returns>
/// <example>
/// <code>
/// EnumExample dummy = EnumExample.Combi;
/// if (dummy.Contains<EnumExample>(EnumExample.ValueA))
/// {
///     Console.WriteLine("dummy contains EnumExample.ValueA");
/// }
/// </code>
/// </example>
public static bool Contains<T>(this Enum value, T request)
{
    int valueAsInt = Convert.ToInt32(value, CultureInfo.InvariantCulture);
    int requestAsInt = Convert.ToInt32(request, CultureInfo.InvariantCulture);

```

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

```

        return true;
    }

    return false;
}
}

```

The enum itself must be decorated with the [FlagsAttribute](#)

```

[Flags]
public enum EnumExample
{
    ValueA = 1,
    ValueB = 2,
    ValueC = 4,
    ValueD = 8,
    Combi = ValueA | ValueB
}

```

edited Dec 7 '10 at 16:53

answered Jun 3 '09 at 12:03



[bob](#)

5,674 1 27 26

-
- 12 A one liner for the first extension method; it's no more lazy. `return Enum.GetValues(typeof(T)).Cast<T>();` – [Leyu](#) Jun 22 '10 at 9:29
-
- 2 Alternatively you could use `OfType` too: `Enum.GetValues(typeof(T)).OfType<T>()`. It's too bad there is not a generic version of `GetValues<T>()` then it would be even more slick. – [jpierson](#) Jan 10 '11 at 22:38
-
- 3 Maybe someone could show how to use these extensions? The compiler do not show extension methods on enum `EnumExample`. – [Tomas](#) Feb 12 '13 at 8:01
-
- 1 can anyone add example how to utilize those functions? – [Ashwini Verma](#) Sep 23 '13 at 13:01
-
- 3 +1 for reusable code: examples - save these extension methods in a library and reference it `[Flags]public enum mytypes{name1, name2 }; List<string> myTypeNames = mytypes.GetAllItems();` – [Krishna](#) Oct 22 '13 at 6:55
-



Some versions of the .NET framework do not support `Enum.GetValues` . Here's a good workaround from [Ideas 2.0: Enum.GetValues in Compact Framework](#):

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).

```

    FieldInfo[] fields = enumeration.GetType().GetFields(BindingFlags.Static |
    BindingFlags.Public);
    Enum[] enumerations = new Enum[fields.Length];

    for (var i = 0; i < fields.Length; i++)
        enumerations[i] = (Enum) fields[i].GetValue(enumeration);

    return enumerations;
}

```

As with any code that involves [reflection](#), you should take steps to ensure it runs only once and results are cached.

edited Mar 8 at 4:39



Mikael Dúi Bolinder

1,469 1 11 33

answered Sep 3 '09 at 18:48



Ekevo

2,080 1 16 29

17 Why not use the yield keyword here instead instantiating a list? – [Eric Mickelsen](#) Jan 16 '11 at 22:18

1 or shorter: `return type.GetFields().Where(x => x.IsLiteral).Select(x => x.GetValue(null)).Cast<Enum>();` – [nawfal](#) Nov 7 '13 at 9:21

7 @nawfal: Linq isn't available .Net CF 2.0. – [Gabriel GM](#) May 23 '14 at 15:16

@Ekevo How to bind this enum values to a DropDownList in MVC? – [Jack](#) Dec 16 '15 at 18:36

Why is no one using `Cast<T>` ?

108

```
var suits = Enum.GetValues(typeof(Suit)).Cast<Suit>();
```

There you go `IEnumerable<Suit>` .

edited Jul 25 '14 at 12:12

answered Mar 17 '13 at 4:15



sircodesalot

7,098 5 38 67

1 This is pretty cool if you can use LINQ... – [MemphiZ](#) Jan 19 '17 at 17:59

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).



96



I think this is more efficient than other suggestions because `GetValues()` is not called each time you have a loop. It is also more concise. And you get a compile-time error not a runtime exception if `Suit` is not an `enum`.

```
EnumLoop<Suit>.ForEach((suit) => {
    DoSomethingWith(suit);
});
```

`EnumLoop` has this completely generic definition:

```
class EnumLoop<Key> where Key : struct, IConvertible {
    static readonly Key[] arr = (Key[])Enum.GetValues(typeof(Key));
    static internal void ForEach(Action<Key> act) {
        for (int i = 0; i < arr.Length; i++) {
            act(arr[i]);
        }
    }
}
```

edited Mar 17 '16 at 19:41



Ahmed Salman Tahir
1,687 1 14 26

answered Feb 2 '12 at 13:36



James
3,441 21 34

- 6 Careful with using generics like this. If you try to use `EnumLoop` with some type that is not an enum, it will compile fine, but throw an exception at runtime. – [svick](#) Feb 6 '12 at 12:09
- 7 Thank you svick. Runtime exceptions will actually occur with the other answers on this page... except this one because I have added "where Key : struct, IConvertible" so that you get a compile time error in most cases. – [James](#) Feb 7 '12 at 12:13
- 3 No, `GetValues()` is called only once in the foreach. – [Alex Blokha](#) Jul 30 '12 at 11:25
- 4 James, I would discourage your class because clever is nice to write but in production code that many people will maintain and update, clever is extra work. If it makes a major saving or will be used a lot - so the savings is big and people will become familiar with it - it is worth it, but in most cases it slows down people trying to read and update the code and introduces a possible source bugs in the future. Less code is better :) less complexity is even better. – [Grant M](#) Jan 19 '15 at 4:00
- 2 @GrantM Why? This code is neither complex, and it's incredibly short. On top of that, writing the class once will allow even shorter iterations of code with using is as per his example. This is extremely clean, if you can't update that code, you probably can't update any of the companies code. – [Dimitris](#) Mar 10 '17 at 10:00

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).

You won't get `Enum.GetValues()` in Silverlight.

76 [Original Blog Post by Einar Ingebrigtsen](#):

```
public class EnumHelper
{
    public static T[] GetValues<T>()
    {
        Type enumType = typeof(T);

        if (!enumType.IsEnum)
        {
            throw new ArgumentException("Type '" + enumType.Name + "' is not an enum");
        }

        List<T> values = new List<T>();

        var fields = from field in enumType.GetFields()
                     where field.IsLiteral
                     select field;

        foreach (FieldInfo field in fields)
        {
            object value = field.GetValue(enumType);
            values.Add((T)value);
        }

        return values.ToArray();
    }

    public static object[] GetValues(Type enumType)
    {
        if (!enumType.IsEnum)
        {
            throw new ArgumentException("Type '" + enumType.Name + "' is not an enum");
        }

        List<object> values = new List<object>();

        var fields = from field in enumType.GetFields()
                     where field.IsLiteral
                     select field;
    }
}
```

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).

```

        values.Add(value);
    }

    return values.ToArray();
}

```

edited Jan 15 '16 at 14:11

Chris
85 1 7

answered Nov 17 '10 at 1:29

Aubrey Taylor
761 5 2

2 Nice solution, but some refactoring will be better! :) – [nawfal](#) Nov 7 '13 at 9:10

I am using .NET framework 4.0 & silverlight enum.getvalues work, the code I used is ----> enum.GetValues(typeof(enum)) – [Ananda](#) Dec 30 '15 at 6:41

2 Starting with C# 7.3 (Visual Studio 2017 ≥ v15.7), one can use where T: Enum – [Yahoo Serious](#) Jul 27 '18 at 14:59

Just to add my solution, which works in compact framework (3.5) and supports type checking **at compile time**:

58

```

public static List<T> GetEnumValues<T>() where T : new() {
    T valueType = new T();
    return typeof(T).GetFields()
        .Select(fieldInfo => (T)fieldInfo.GetValue(valueType))
        .Distinct()
        .ToList();
}

public static List<String> GetEnumNames<T>() {
    return typeof(T).GetFields()
        .Select(info => info.Name)
        .Distinct()
        .ToList();
}

```

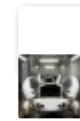
- If anyone knows how to get rid of the `T valueType = new T()`, I'd be happy to see a solution.

A call would look like this:

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).

edited Jan 30 '14 at 4:43

answered Jul 7 '10 at 13:37



nawfal

46.9k 39 269 316



Mallox

1,244 11 12

2 what about using `T valueType = default(T) ?` – Oliver Jul 7 '10 at 14:17

Great, I didn't even know that keyword. Always nice to learn something new. Thank you! Does it always return a reference to the same object, or does it create a new instance each time the default statement is called? I haven't found anything on the net about this so far, but if it creates a new instance every time, it kind of defeats the purpose I was looking for (having a one-liner ^^). – Mallox Jul 8 '10 at 6:48

1 Wouldn't this create a new instance for every iteration over the enumeration? – Mallox Jun 28 '11 at 15:04

1 -1 for "supports type checking at compile time:". What type checking? This would work for any `new() T`. Also, you dont need `new T()` at all, you can select just the static fields alone and do `.GetValue(null)`. See Aubrey's answer. – nawfal Jan 30 '14 at 4:47

2 Starting with C# 7.3 (Visual Studio 2017 ≥ v15.7), one can use `where T: Enum` – Yahoo Serious Jul 27 '18 at 14:59

I think you can use

52

```
Enum.GetNames(Suit)
```

edited Jul 25 '11 at 7:46

answered Sep 19 '08 at 20:37



bluish

15.2k 19 96 155



Tom Carr

1,199 6 10

6 `Enum.GetValues(Suits)` – Ian Boyd Sep 22 '08 at 14:43

50

```
public void PrintAllSuits()
{
    foreach(string suit in Enum.GetNames(typeof(Suits)))
    {
        Console.WriteLine(suit);
    }
}
```

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).

edited Jan 9 '17 at 20:45



Termininja

4,696

12

32

39

answered Sep 19 '08 at 21:05



Joshua Drake

2,352

2

31

52

-
- 2 That enumerates a string, don't forget to convert those things back to an enumeration value so the enumeration can be enumerated. – [Ian Boyd](#) Jun 1 '10 at 17:46
-
- 1 I see from your edit that you want to actually operate on the enums themselves, the above code addressed your original post. – [Joshua Drake](#) Jun 4 '10 at 15:22
-

49

```
foreach (Suit suit in Enum.GetValues(typeof(Suit))) { }
```

I've heard vague rumours that this is terifically slow. Anyone know? – [Orion Edwards](#) Oct 15 '08 at 1:31 7

I think caching the array would speed it up considerably. It looks like you're getting a new array (through reflection) every time. Rather:

```
Array enums = Enum.GetValues(typeof(Suit));
foreach (Suit suitEnum in enums)
{
    DoSomething(suitEnum);
}
```

That's at least a little faster, ja?

edited Oct 6 '17 at 7:20



Alexander Schmidt

3,617

3

30

68

answered Nov 16 '09 at 17:19



Limited Atonement

4,671

2

39

52

-
- 5 The compiler should take care of this, though. – [Stephan Bijzitter](#) Feb 23 '15 at 12:57
-
- @StephanBijzitter Wow, you're reading pretty far down on this one :-)) I agree, the compiler should make my solution unnecessary. – [Limited Atonement](#) Feb 23 '15 at 17:52
-
- 1 This is not necessary. Looking at the compiled code in ILSpy, the compiler definitely already does this. Why is this answer upvoted at all, much less 35 times? – [mhennv1384](#) May 8 '18 at 16:14
-

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).

Three ways:

29

1. `Enum.GetValues(type)` *//since .NET 1.1, not in silverlight or compact framewok*
2. `type.GetEnumValues()` *//only on .NET 4 and above*
3. `type.GetFields().Where(x => x.IsLiteral).Select(x => x.GetValue(null))` *//works everywhere*

Not sure why was `GetEnumValues` introduced on type instance, it isn't very readable at all for me.

Having a helper class like `Enum<T>` is what is most readable and memorable for me:

```
public static class Enum<T> where T : struct, IComparable, IFormattable, IConvertible
{
    public static IEnumerable<T> GetValues()
    {
        return (T[])Enum.GetValues(typeof(T));
    }

    public static IEnumerable<string> GetNames()
    {
        return Enum.GetNames(typeof(T));
    }
}
```

Now you call:

```
Enum<Suit>.GetValues();
//or
Enum.GetValues(typeof(Suit)); //pretty consistent style
```

One can also use sort of caching if performance matters, but I don't expect this to be an issue at all

```
public static class Enum<T> where T : struct, IComparable, IFormattable, IConvertible
{
    private static Dictionary<T, string> _names = new Dictionary<T, string>();
```

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

```

public static IEnumerable<T> GetValues()
{
    return values ?? (values = (T[])Enum.GetValues(typeof(T)));
}

public static IEnumerable<string> GetNames()
{
    return names ?? (names = Enum.GetNames(typeof(T)));
}
}

```

edited Jan 9 '17 at 23:43



Termininja

4,696

12

32

39

answered Aug 12 '13 at 15:36



nawfal

46.9k

39

269

316

This is a nice summary of methods. I think you should merge your other answer into this though. The truth is that enum are special and looping through them is often (usually) just as valid as enumeration because you know that the values will never change. IOW, If you have an enum that is changing all the time then you've chosen the wrong data construct to begin with. – [krowe2](#) Dec 19 '17 at 17:03

What the hell I'll throw my two pence in, just by combining the top answers I through together a very simple extension

28

```

public static class EnumExtensions
{
    /// <summary>
    /// Gets all items for an enum value.
    /// </summary>
    /// <typeparam name="T"></typeparam>
    /// <param name="value">The value.</param>
    /// <returns></returns>
    public static IEnumerable<T> GetAllItems<T>(this T value) where T : Enum
    {
        return (T[])Enum.GetValues(typeof(T));
    }
}

```

Clean simple and by @Jeppe-Stig-Nielsen s comment fast.

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).

- 4 Starting with C# 7.3 (Visual Studio 2017 ≥ v15.7), one can use `where T: Enum` – [Yahoo Serious](#) Jul 27 '18 at 14:57
- not work in previous versions of c# – [reza akhlaghi](#) yesterday

I use `ToString()` then split and parse the spit array in flags.

23

```
[Flags]
public enum ABC {
    a = 1,
    b = 2,
    c = 4
};

public IEnumerable<ABC> Getselected (ABC flags)
{
    var values = flags.ToString().Split(',');
    var enums = values.Select(x => (ABC)Enum.Parse(typeof(ABC), x.Trim()));
    return enums;
}

ABC temp= ABC.a | ABC.b;
var list = getSelected (temp);
foreach (var item in list)
{
    Console.WriteLine(item.ToString() + " ID=" + (int)item);
}
```

edited Nov 18 '15 at 15:51

answered Jul 26 '12 at 9:22



[Mickey Perlstein](#)

1,418 2 18 29

There are two ways to iterate an `Enum` :

22

```
1. var values = Enum.GetValues(typeof(myenum))
2. var values = Enum.GetNames(typeof(myenum))
```

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).

Use it in `foreach` loop as below:

```
foreach(var value in values)
{
    //Do operations here
}
```

edited Jan 13 '17 at 7:12



Termininja

4,696 12 32 39

answered Jan 22 '16 at 18:50



Kyro Ren

5,848 2 28 48

2 Who is down voting....it works perfectly... and at least add a comment what they think is wrong? – [Kyro Ren](#) Apr 27 '16 at 9:21

5 downvoting without even giving the reason is really not fair – [Learning-Overthinker-Confused](#) Jun 20 '16 at 6:19

1 Maybe 'cos this is already covered in many answers? Let's not make answers redundant. – [nawfal](#) May 17 '17 at 15:38

@nawfal yes may be covered in other answers, though not concluded well in most of them. – [Kyro Ren](#) May 18 '17 at 5:39

I do not hold the opinion this is better, or even good, just stating yet another solution.

16

If enum values range strictly from 0 to n - 1, a generic alternative:

```
public void EnumerateEnum<T>()
{
    int length = Enum.GetValues(typeof(T)).Length;
    for (var i = 0; i < length; i++)
    {
        var @enum = (T)(object)i;
    }
}
```

If enum values are contiguous and you can provide the first and last element of the enum, then:

```
public void EnumerateEnum()
```

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).


```

        var @enum = i;
    }
}

```

but that's not strictly enumerating, just looping. The second method is much faster than any other approach though...

answered Jan 25 '13 at 0:05



[nawfal](#)

46.9k

39

269

316

If you need speed and type checking at build and run time, this helper method is better than using LINQ to cast each element:

16

```

public static T[] GetEnumValues<T>() where T : struct, IComparable, IFormattable,
IConvertible
{
    if (typeof(T).BaseType != typeof(Enum))
    {
        throw new ArgumentException(string.Format("{0} is not of type System.Enum",
typeof(T)));
    }
    return Enum.GetValues(typeof(T)) as T[];
}

```

And you can use it like below:

```
static readonly YourEnum[] _values = GetEnumValues<YourEnum>();
```

Of course you can return `IEnumerable<T>`, but that buys you nothing here.

edited Jan 17 '16 at 12:11



[Peter Mortensen](#)

14.5k

19

89

118

answered Nov 15 '13 at 20:13



[dmihailescu](#)

1,292

15

14

2 Starting with C# 7.3 (Visual Studio 2017 ≥ v15.7), one can use `where T : Enum` — [Yahoo Serious](#) Jul 27 '18 at 14:57

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).

13

```
var resman = ViewModelResources.TimeFrame.ResourceManager;

ViewBag.TimeFrames = from MapOverlayTimeFrames timeFrame
    in Enum.GetValues(typeof(MapOverlayTimeFrames))
    select new SelectListItem
    {
        Value = timeFrame.ToString(),
        Text = resman.GetString(timeFrame.ToString()) ?? timeFrame.ToString()
    };
```

edited Oct 2 '15 at 14:00



Ian Boyd

129k 203 725 1041

answered Oct 25 '12 at 21:20



jhilden

8,672 4 38 61

10

```
foreach (Suit suit in Enum.GetValues(typeof(Suit)))
{
}
```

(The current accepted answer has a cast that I don't think is needed (although I may be wrong).)

edited Jan 17 '16 at 12:12



Peter Mortensen

14.5k 19 89 118

answered Jan 20 '14 at 10:37



matt burns

18.5k 5 81 95

This question appears in Chapter 10 of "[C# Step by Step 2013](#)"

10

The author uses a double for-loop to iterate through a pair of Enumerators (to create a full deck of cards):

```
class Pack
{
    public const int NumSuits = 4;
    public const int CardsPerSuit = 13;
    private PlayingCard[,] cardPack;

    public Pack()
```

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).

```

    {
        for (Value value = Value.Two; value <= Value.Ace; value++)
        {
            cardPack[(int)suit, (int)value] = new PlayingCard(suit, value);
        }
    }
}

```

In this case, `Suit` and `Value` are both enumerations:

```

enum Suit { Clubs, Diamonds, Hearts, Spades }
enum Value { Two, Three, Four, Five, Six, Seven, Eight, Nine, Ten, Jack, Queen, King, Ace}

```

and `PlayingCard` is a card object with a defined `suit` and `Value` :

```

class PlayingCard
{
    private readonly Suit suit;
    private readonly Value value;

    public PlayingCard(Suit s, Value v)
    {
        this.suit = s;
        this.value = v;
    }
}

```

edited Jan 13 '17 at 7:22



Termininja

4,696 12 32 39

answered Jun 28 '15 at 6:12



Ross Gatih

366 2 6

will this work if the values in enum are not sequential? – [Aamir Masood](#) Feb 1 '17 at 9:50

▲ I know it is a bit messy but if you are fan of one-liners, here is one:

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).

edited May 15 '15 at 11:31

answered Apr 8 '14 at 15:44



anar khalilov

12.6k 8 39 55

1 Is that [lisp](#)? – Mikael Dúi Bolinder Mar 1 at 15:02

What if you know the type will be an `enum`, but you don't know what the exact type is at compile time?

8

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

    public static IEnumerable getListOfEnum(Type type)
    {
        MethodInfo getValuesMethod =
        typeof(EnumHelper).GetMethod("GetValues").MakeGenericMethod(type);
        return (IEnumerable)getValuesMethod.Invoke(null, null);
    }
}
```

The method `getListOfEnum` uses reflection to take any enum type and returns an `IEnumerable` of all enum values.

Usage:

```
Type myType = someEnumValue.GetType();

IEnumerable resultEnumerable = getListOfEnum(myType);

foreach (var item in resultEnumerable)
{
    Console.WriteLine(String.Format("Item: {0} Value: {1}", item.ToString(), (int)item));
}
```

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).

A simple and generic way to convert an enum to something you can interact:

8

```
public static Dictionary<int, string> ToList<T>() where T : struct
{
    return ((IEnumerable<T>)Enum
        .GetValues(typeof(T)))
        .ToDictionary(
            item => Convert.ToInt32(item),
            item => item.ToString());
}
```

And then:

```
var enums = EnumHelper.ToList<MyEnum>();
```

edited Oct 25 '16 at 14:12



Massimiliano Kraus

2,559 4 16 34

answered Sep 12 '14 at 18:06



Gabriel

802 10 21

A Dictionary is not a good idea: if you have an Enum like enum E { A = 0, B = 0 }, the 0 value is added 2 times generating an ArgumentException (you cannot add the same Key on a Dictionary 2 or more times!). – [Massimiliano Kraus](#) Oct 25 '16 at 13:00

Why return a Dictionary<,> from a method named ToList ? Also why not return Dictionary<T, string> ? – [Aluan Haddad](#) Dec 11 '17 at 11:53

Add method public static IEnumerable<T> GetValues<T>() to your class, like

5

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

call and pass your enum. now you can iterate through it using foreach

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).

```

{
    // custom method
    var foos = GetValues<Suit>();
    foreach (var foo in foos)
    {
        // Do something
    }
}

```

edited Oct 6 '17 at 7:18



Alexander Schmidt

3,617 3 30 68

answered Mar 31 '17 at 7:08



MUT

381 2 16



2

enum types are called "enumeration types" not because they are containers that "enumerate" values (which they aren't), but because they are defined by *enumerating* the possible values for a variable of that type.

(Actually, that's a bit more complicated than that - enum types are considered to have an "underlying" integer type, which means each enum value corresponds to an integer value (this is typically implicit, but can be manually specified). C# was designed in a way so that you could stuff *any* integer of that type into the enum variable, even if it isn't a "named" value.)

The [System.Enum.GetNames method](#) can be used to retrieve an array of strings which are the names of the enum values, as the name suggests.

EDIT: Should have suggested the [System.Enum.GetValues](#) method instead. Oops.

edited Jan 16 '18 at 20:14

answered Oct 11 '17 at 4:50



Emily Chen

120 8

2 Although your answer is correct in itself, it doesn't really address the OP's original question. The `GetNames` method returns, indeed, a string array, but the OP requires an enumerator through the values. – [Silviu Preda](#) Jan 15 '18 at 9:30

@SilviuPreda: Edited. It should have been `GetValues` instead of `GetNames`. – [Emily Chen](#) Jan 16 '18 at 20:17

Also you can bind to the public static members of the enum directly by using reflection.

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).

`.ToList().ForEach(x => DoSomething(x.Name));`

answered Jan 9 '17 at 23:39



[Termininja](#)

4,696

12

32

39

i try many ways and get result from this code : for get a list of int from an enum use this; thats works!

0

```
List<int> listEnumValues = new List<int>();
YourEnumType[] myEnumMembers = (YourEnumType[])Enum.GetValues(typeof(YourEnumType));
foreach ( YourEnumType enumMember in myEnumMembers)
{
    listEnumValues.Add(enumMember.GetHashCode());
}
```

answered yesterday



[reza akhlaghi](#)

443

1

7

15

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).