

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

What does the [Flags] Enum Attribute mean in C#?

Asked 11 years, 1 month ago Active 8 months ago Viewed 451k times

From time to time I see an enum like the following:

1357



436

```
[Flags]
public enum Options
{
    None      = 0,
    Option1   = 1,
    Option2   = 2,
    Option3   = 4,
    Option4   = 8
}
```

I don't understand what exactly the [Flags] -attribute does.

Anyone have a good explanation or example they could post?

c#

enums

flags

edited Oct 11 '14 at 10:48



Vogel612

4,624 5 41 64

asked Aug 12 '08 at 4:09



Brian Leahy

17.5k 9 41 60

It's also worth noting, in addition to the accepted answer, that VB.NET actually *requires* [Flags] - at least according to the .NET guys: social.msdn.microsoft.com/forums/en-US/csharp/language/thread/... – Rushyo Jul 30 '12 at 15:38

8 Note, not required in VB these days. Save behaviour as C# - just changes the ToString() output. Note, you can also do logical OR, WITHIN the Enum itself. Very cool. Cat = 1, Dog = 2, CatAndDog = Cat || Dog. – Chalky Aug 4 '15 at 9:13

14 @Chalky You mean CatAndDog = Cat & Dog (the logical OR instead of the Conditional). I assume? – DdW Sep 12 '16 at 12:20

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

11 Answers



2046



The `[Flags]` attribute should be used whenever the enumerable represents a collection of possible values, rather than a single value. Such collections are often used with bitwise operators, for example:

```
var allowedColors = MyColor.Red | MyColor.Green | MyColor.Blue;
```

Note that the `[Flags]` attribute **doesn't** enable this by itself - all it does is allow a nice representation by the `.ToString()` method:

```
enum Suits { Spades = 1, Clubs = 2, Diamonds = 4, Hearts = 8 }
[Flags] enum SuitsFlags { Spades = 1, Clubs = 2, Diamonds = 4, Hearts = 8 }

...

var str1 = (Suits.Spades | Suits.Diamonds).ToString();
           // "5"
var str2 = (SuitsFlags.Spades | SuitsFlags.Diamonds).ToString();
           // "Spades, Diamonds"
```

It is also important to note that `[Flags]` **does not** automatically make the enum values powers of two. If you omit the numeric values, the enum will not work as one might expect in bitwise operations, because by default the values start with 0 and increment.

Incorrect declaration:

```
[Flags]
public enum MyColors
{
    Yellow, // 0
    Green,  // 1
    Red,    // 2
    Blue    // 3
}
```

The values, if declared this way, will be Yellow = 0, Green = 1, Red = 2, Blue = 3. This will render it useless as flags.

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

```
[Flags]
public enum MyColors
{
    Yellow = 1,
    Green = 2,
    Red = 4,
    Blue = 8
}
```

To retrieve the distinct values in your property, one can do this:

```
if (myProperties.AllowedColors.HasFlag(MyColor.Yellow))
{
    // Yellow is allowed...
}
```

or prior to .NET 4:

```
if((myProperties.AllowedColors & MyColor.Yellow) == MyColor.Yellow)
{
    // Yellow is allowed...
}

if((myProperties.AllowedColors & MyColor.Green) == MyColor.Green)
{
    // Green is allowed...
}
```

Under the covers

This works because you used powers of two in your enumeration. Under the covers, your enumeration values look like this in binary ones and zeros:

```
Yellow: 00000001
Green:  00000010
Red:    00000100
Blue:   00001000
```

Similarly, after you've set your property *AllowedColors* to Red, Green and Blue using the binary bitwise OR `|` operator, *AllowedColors*

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

```
myProperties.AllowedColors: 00001110
```

So when you retrieve the value you are actually performing bitwise AND & on the values:

```
myProperties.AllowedColors: 00001110
MyColor.Green: 00000010
-----
00000010 // Hey, this is the same as MyColor.Green!
```

The None = 0 value

And regarding the use of 0 in your enumeration, quoting from MSDN:

```
[Flags]
public enum MyColors
{
    None = 0,
    ....
}
```

Use None as the name of the flag enumerated constant whose value is zero. **You cannot use the None enumerated constant in a bitwise AND operation to test for a flag because the result is always zero.** However, you can perform a logical, not a bitwise, comparison between the numeric value and the None enumerated constant to determine whether any bits in the numeric value are set.

You can find more info about the flags attribute and its usage at [msdn](#) and [designing flags at msdn](#)

edited Feb 8 at 3:29



Matt Jenkins

1,540 1 19 28

answered Aug 12 '08 at 5:10



andnil


23.4k 2 25 24

145 Flags itself does nothing. Also, C# does not require Flags per se. But the ToString implementation of your enum uses Flags, and so does Enum.IsDefined, Enum.Parse, etc. Try to remove Flags and look at the result of MyColor.Yellow | MyColor.Red; without it you get "5", with Flags you get "Yellow, Red". Some other parts of the framework also use [Flags] (e.g., XML Serialization). – Ruben Aug 17 '09 at 17:30

7 // Yellow has been set..., I find a bit misleading. Nothing has been set, it means that Yellow is a member of your AllowedColors, perhaps better would be "Yellow is set" or "Yellow is in AllowedColors". – Matt Aug 17 '09 at 17:30

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

Nick Westgate Apr 9 '13 at 12:06

- 1 @borrrden, hell yeahh! I found this: msdn.microsoft.com/en-us/library/system.enum.aspx - see "Remarks" part: "Enum is the base class for all enumerations in the .NET Framework." and "The enumeration does not explicitly inherit from Enum; the inheritance relationship is handled implicitly by the compiler." So, when you write: public enum bla bla bla - this is a value type. But, the HasFlag method wants you to give him an instance of System.Enum which is a class (reference type:) – [Aleksei Chepovoi](#) Jun 28 '13 at 14:59 
- 1 Enum.IsDefined does not take the FlagsAttribute into account. None of the following return true, even with the attribute: Yellow | Green, "Yellow, Green", 3 – [TheXenocide](#) Dec 6 '13 at 21:29

You can also do this

753

```
[Flags]
public enum MyEnum
{
    None    = 0,
    First   = 1 << 0,
    Second  = 1 << 1,
    Third   = 1 << 2,
    Fourth  = 1 << 3
}
```

I find the bit-shifting easier than typing 4,8,16,32 and so on. It has no impact on your code because it's all done at compile time


edited Jun 18 '14 at 2:45

answered Aug 12 '08 at 4:37



[Orion Edwards](#)

88.2k 52 199 279

- 16 That's what I mean, I prefer to have the full int in source code. If I have a column on a table in the database called MyEnum that stores a value of one of the enums, and a record has 131,072, I would need to get out my calculator to figure out that that corresponds to the enum with the value 1<<17. As opposed to just seeing the value 131,072 written in the source. – [JeremyWeir](#) May 29 '12 at 17:11
- 6 @jwg I agree, it's silly to be overly worried about the runtime performance of this, but all the same, I do think its nice to know that this isn't going to be inserting bitshifts anywhere you use the enum. More of a 'that's neat' thing rather than anything related to performance – [Orion Edwards](#) Feb 15 '13 at 19:25 
- 17 @JeremyWeir - Several bits are going to be set in a flag enumeration value. So your method of data analysis is what is improper. Run a procedure to Represent your integer value in binary. 131,072 [D] = 0000 0000 0000 0001 0000 0000 0000 0000 [B] [32].. With the 17th bit set, an enum value assigned 1<<17 is easily determinable. 0110 0011 0011 0101 0101 0011 0101 1011 [b] [32]... the enum values assign 1<<31, 1<<30, 1<<28, 1<<25

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

- 13 Also pointing out you can bit-shift from "previous" enum values rather than directly from numbers, i.e. `Third = Second << 1` rather than `Third = 1 << 2` -- see [more complete description below](#) – drzaus May 16 '13 at 15:05
- 23 I like this, but once you get to the upper limits of the underlying enum type, the compiler doesn't warn against bit shifts such as `1 << 31 == -2147483648`, `1 << 32 == 1`, `1 << 33 == 2`, and so on. By contrast, if you say `ThirtySecond = 2147483648` for an int type enum, the compiler throws an error. – aaaantoinne Jun 3 '14 at 20:58



108



Combining answers <https://stackoverflow.com/a/8462/1037948> (declaration via bit-shifting) and <https://stackoverflow.com/a/9117/1037948> (using combinations in declaration) you can bit-shift previous values rather than using numbers. Not necessarily recommending it, but just pointing out you can.

Rather than:

```
[Flags]
public enum Options : byte
{
    None    = 0,
    One     = 1 << 0,    // 1
    Two     = 1 << 1,    // 2
    Three   = 1 << 2,    // 4
    Four    = 1 << 3,    // 8

    // combinations
    OneAndTwo = One | Two,
    OneTwoAndThree = One | Two | Three,
}
```

You can declare

```
[Flags]
public enum Options : byte
{
    None    = 0,
    One     = 1 << 0,    // 1
    // now that value 1 is available, start shifting from there
    Two     = One << 1,    // 2
    Three   = Two << 1,    // 4
    Four    = Three << 1,  // 8
}
```

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

```
OneTwoAndThree = One | Two | Three,
}
```

Confirming with LinqPad:

```
foreach(var e in Enum.GetValues(typeof(Options))) {
    string.Format("{0} = {1}", e.ToString(), (byte)e).Dump();
}
```

Results in:

```
None = 0
One = 1
Two = 2
OneAndTwo = 3
Three = 4
OneTwoAndThree = 7
Four = 8
```

edited May 23 '17 at 10:31



Community ♦

1 1

answered May 16 '13 at 15:03



drzaus

16.6k 7 93 155

-
- 21 The combinations are a good recommendation, but I think the chained bit-shift would be prone to copy-and-paste errors such as Two = One << 1, Three = One << 1, etc... The incrementing integers of the form 1 << n are safer and the intent is clearer. – [Rupert Rawnsley](#) Mar 20 '14 at 13:00
-
- 2 @RupertRawnsley to quote my answer: > Not necessarily recommending it, but just pointing out you can – [drzaus](#) Mar 20 '14 at 16:13
-

Please see the following for an example which shows the declaration and potential usage:

48

```
namespace Flags
{
    class Program
    {
        [Flags]
```

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

```

        Bar = 0x2,
        Baz = 0x4
    }

    static void Main(string[] args)
    {
        MyFlags fooBar = MyFlags.Foo | MyFlags.Bar;

        if ((fooBar & MyFlags.Foo) == MyFlags.Foo)
        {
            Console.WriteLine("Item has Foo flag set");
        }
    }
}

```

edited Aug 14 '16 at 2:23



Jaider

9,983

5

49

73

answered Aug 12 '08 at 4:32



OJ.

25.6k

5

49

67

This example works even if you leave out [Flags]. It's the [Flags] part I'm trying to learn about. – [gbarry](#) Aug 6 at 23:21

I [asked recently](#) about something similar.

34

If you use flags you can add an extension method to enums to make checking the contained flags easier (see post for detail)

This allows you to do:

```

[Flags]
public enum PossibleOptions : byte
{
    None = 0,
    OptionOne = 1,
    OptionTwo = 2,
    OptionThree = 4,
    OptionFour = 8,

    //combinations can be in the enum too
    OptionOneAndTwo = OptionOne | OptionTwo.
}

```

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

Then you can do:

```
PossibleOptions opt = PossibleOptions.OptionOneTwoAndThree

if( opt.IsSet( PossibleOptions.OptionOne ) ) {
    //optionOne is one of those set
}
```

I find this easier to read than the most ways of checking the included flags.

edited May 23 '17 at 12:34



Community ♦

1 1

answered Aug 12 '08 at 18:40



Keith

100k 63 249 365

IsSet is an extension method I assume? – Robert MacLean Jun 2 '09 at 10:09

Yeah - read the other question that I link to for details: stackoverflow.com/questions/7244 – Keith Jun 2 '09 at 12:16

69 .NET 4 adds a HasFlag method to enumerations, so you can do opt.HasFlag(PossibleOptions.OptionOne) without having to write your own extensions – Orion Edwards Jul 19 '10 at 20:53

1 Note that HasFlag is [much slower](#) than doing bitwise operations. – Wai Ha Lee Sep 12 '18 at 15:45

1 @WaiHaLee You can use CodeJam.EnumHelper.IsFlagSet extension method which is compiled to a fast version using Expression: github.com/rsdn/CodeJam/wiki/M_CodeJam_EnumHelper_IsFlagSet__1 – NN_ Apr 29 at 20:09

In extension to the accepted answer, in C#7 the enum flags can be written using binary literals:

28

```
[Flags]
public enum MyColors
{
    None    = 0b0000,
    Yellow  = 0b0001,
    Green   = 0b0010,
    Red     = 0b0100,
    Blue    = 0b1000
}
```

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

answered Jul 20 '17 at 11:09

[Thorkil Holm-Jacobsen](#)

4,423 3 19 37

@Nidonocu

22

To add another flag to an existing set of values, use the OR assignment operator.

```

Mode = Mode.Read;
//Add Mode.Write
Mode |= Mode.Write;
Assert.True(((Mode & Mode.Write) == Mode.Write)
    && ((Mode & Mode.Read) == Mode.Read));

```

answered Aug 12 '08 at 15:37

[steve_c](#)

4,959 4 26 38

To add Mode.Write :

17

```

Mode = Mode | Mode.Write;

```

edited Apr 14 '16 at 5:46

[shA.t](#)

13.6k 4 41 79

answered Aug 12 '08 at 5:59

[David Wengier](#)

9,027 5 34 42

55 or Mode |= Mode.Write – [abatishchev](#) Feb 24 '09 at 11:29

There's something overly verbose to me about the `if ((x & y) == y)...` construct, especially if `x` AND `y` are both compound sets of flags and you only want to know if there's **any** overlap

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

[1] See Jaime's comment. If we were authentically *bitmasking*, we'd only need to check that the result was positive. But since `enums` can be negative, even, strangely, when combined with [the \[Flags\] attribute](#), it's defensive to code for `!= 0` rather than `> 0`.

Building off of @andnil's setup...

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace BitFlagPlay
{
    class Program
    {
        [Flags]
        public enum MyColor
        {
            Yellow = 0x01,
            Green = 0x02,
            Red = 0x04,
            Blue = 0x08
        }

        static void Main(string[] args)
        {
            var myColor = MyColor.Yellow | MyColor.Blue;
            var acceptableColors = MyColor.Yellow | MyColor.Red;

            Console.WriteLine((myColor & MyColor.Blue) != 0);    // True
            Console.WriteLine((myColor & MyColor.Red) != 0);     // False
            Console.WriteLine((myColor & acceptableColors) != 0); // True
            // ... though only Yellow is shared.

            Console.WriteLine((myColor & MyColor.Green) != 0);   // Wait a minute...

            Console.Read();
        }
    }
}
```

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



Enum may be based on a signed type, so you should use "`!= 0`" instead of "`> 0`". – [raven](#) Aug 16 '13 at 9:18

@JaimePardos -- As long as we keep them honest bytes, as I do in this example, there's no concept of negative. Just 0 to 255. As [MSDN warns](#), "Use caution if you define a negative number as a flag enumerated constant because... [that] might make your code confusing and encourage coding errors." It's strange to think in terms of "negative bitflags"! ;^) I'll edit more in a bit. But you're right, if we do use negative values in our enum, we'd need to check for `!= 0`. – [ruffin](#) Aug 16 '13 at 13:23



Flags allow you to use bitmasking inside your enumeration. This allows you to combine enumeration values, while retaining which ones are specified.

11



```
[Flags]
public enum DashboardItemPresentationProperties : long
{
    None = 0,
    HideCollapse = 1,
    HideDelete = 2,
    HideEdit = 4,
    HideOpenInNewWindow = 8,
    HideResetSource = 16,
    HideMenu = 32
}
```

edited Feb 7 '16 at 18:21



[Markus Safar](#)

5,053 4 20 38

answered Aug 12 '08 at 4:24



[Jay Mooney](#)

1,257 1 14 21



When working with flags I often declare additional None and All items. These are helpful to check whether all flags are set or no flag is set.

11



```
[Flags]
enum SuitsFlags {

    None = 0,
```

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

```
Hearts = 1 << 3,  
All = ~(~0 << 4)  
}
```

Usage:

```
Spades | Clubs | Diamonds | Hearts == All // true  
Spades & Clubs == None // true
```

answered Oct 23 '18 at 8:43



Jpsy

13.3k

3

78

89

protected by [cassiomolin](#) Feb 21 at 13:45

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?