# Cast int to enum in C#

Ask Question

2813

409

How can an `int` be cast to an `enum` in C#?

c# enums casting

edited Oct 29 '14 at 17:57
alexy13
2,136 3 28 52

asked Aug 27 '08 at 3:58
lomaxx
57.2k 54 131 172

37 For the other way round: get-int-value-from-enum – nawfal Jun 9 '13 at 11:53

## 24 Answers

3394

From a string:

```
YourEnum foo = (YourEnum) Enum.Parse(typeof(YourEnum), yourString);
// the foo.ToString().Contains(",") check is necessary for enumerat
[Flags] attribute
if (!Enum.IsDefined(typeof(YourEnum), foo) && !foo.ToString().Conta
  throw new InvalidOperationException($"{yourString} is not an unde
YourEnum enumeration.")
```

From an int:

```
YourEnum foo = (YourEnum)yourInt;
```

**Update:**

From number you can also

```
YourEnum foo = (YourEnum)Enum.ToObject(typeof(YourEnum) , yourInt);
```

edited Jan 5 '17 at 8:56
Micha Wiedenmann
**10.6k** 13 64 106

answered Aug 27 '08 at 3:59
FlySwat
**116k** 63 231 300

22 @FlySwat, what if `YourEnum` is dynamic and will only be known at runtime, and what I want is to convert to `Enum` ? – Shimmy Feb 19 '12 at 9:56

192 Be aware that Enum.Parse will NOT work if your code is obfuscated. At run time after obfuscation the string is compared to the enum names, and at this point the names of the enums aren't what you would expect them to be. Your parse will fail where they succeeded before as a result. – jropella Apr 26 '13 at 18:03

130 **BEWARE** If you use the "from a string" syntax above and pass in an invalid string that is a number (e.g. "2342342" -- assuming that's not a value of your enum), it will actually allow that without throwing an error! Your enum will have that value (2342342) even though it's not a valid choice in the enum itself. – JoeCool Jun 25 '13 at 15:14

106 I think this answer is a bit dated now. For string, you should really be using `var result = Enum.TryParse(yourString, out yourEnum)`

nowadays (and checking the result to determine if the conversion failed). – Justin T Conroy Nov 26 '13 at 21:40

16 It is also possible to have `Enum.Parse` be case-insensitive by adding a `true` parameter value to the call: `YourEnum foo = (YourEnum) Enum.Parse(typeof(YourEnum), yourString, true);` – Erik Schierboom Feb 5 '14 at 12:18

1

Here's an extension method that casts `Int32` to `Enum`.

It honors bitwise flags even when the value is higher than the maximum possible. For example if you have an enum with possibilities *1*, *2*, and *4*, but the int is *9*, it understands that as *1* in absence of an *8*. This lets you make data updates ahead of code updates.

```
   public static TEnum ToEnum<TEnum>(this int val) where TEnum : stru
IFormattable, IConvertible
    {
        if (!typeof(TEnum).IsEnum)
        {
            return default(TEnum);
        }

        if (Enum.IsDefined(typeof(TEnum), val))
        {//if a straightforward single value, return that
            return (TEnum)Enum.ToObject(typeof(TEnum), val);
        }

        var candidates = Enum
            .GetValues(typeof(TEnum))
            .Cast<int>()
            .ToList();

        var isBitwise = candidates
            .Select((n, i) => {
                if (i < 2) return n == 0 || n == 1;
                return n / 2 == candidates[i - 1];
            })
            .All(y => y);
```

Home

PUBLIC

```
            var maxPossible = candidates.Sum();

            if (
                Enum.TryParse(val.ToString(), out TEnum asEnum)
                && (val <= maxPossible || !isBitwise)
            ){//if it can be parsed as a bitwise enum with multiple flag
              //or is not bitwise, return the result of TryParse
                return asEnum;
            }

            //If the value is higher than all possible combinations,
            //remove the high imaginary values not accounted for in the 
            var excess = Enumerable
                .Range(0, 32)
                .Select(n => (int)Math.Pow(2, n))
                .Where(n => n <= val && n > 0 && !candidates.Contains(n)
                .Sum();

            return Enum.TryParse((val - excess).ToString(), out asEnum)
    default(TEnum);
        }
```

answered Feb 22 at 1:31

Chad Hedgcock
7,309 3 22 34

---

0

You simply use **Explicit conversion** Cast int to enum or enum to int

```
    class Program
        {
            static void Main(string[] args)
            {
                Console.WriteLine((int)Number.three); //Output=3

                Console.WriteLine((Number)3);// Outout three
                Console.Read();
            }

            public enum Number
            {
                Zero = 0,
                One = 1,
                Two = 2,
                three = 3
            }
        }
```

answered Feb 1 at 10:15

Shivam Mishra
**36** 6

28

If you have an integer that acts as a bitmask and could represent one or more values in a [Flags] enumeration, you can use this code to parse the individual flag values into a list:

```
for (var flagIterator = 0; flagIterator < 32; flagIterator++)
{
    // Determine the bit value (1,2,4,...,Int32.MinValue)
    int bitValue = 1 << flagIterator;

    // Check to see if the current flag exists in the bit mask
    if ((intValue & bitValue) != 0)
    {
        // If the current flag exists in the enumeration, then we ca
the list
        // if the enumeration has that flag defined
        if (Enum.IsDefined(typeof(MyEnum), bitValue))
            Console.WriteLine((MyEnum)bitValue);
    }
}
```

Note that this assumes that the underlying type of the `enum` is a signed 32-bit integer. If it were a different numerical type, you'd have to change the hardcoded 32 to reflect the bits in that type (or programatically derive it using `Enum.GetUnderlyingType()` )

edited Jan 2 at 15:50

answered Apr 13 '11 at 20:13

Evan M
**1,885** 21 29

Is this loop never terminate? flagIterator = 0x00000001, flagIterator = 0x00000002, flagIterator = 0x00000004, ..., flagIterator = 0x40000000, flagIterator = 0x80000000, flagIterator = 0x00000000. In other words, the value will always be lower than 0x80000000 because it overflow to zero after the case where bit D31 = 1. Then, it remain 0 forever because shifting left the value 0 gives 0 – Christian Gingras Jan 2 at 5:32

Great catch @christiangingras, thank you! I've modified the answer to account for that, and it should take into account when the highest bit is set (i.e. 0x80000000/Int32.MinValue) – Evan M Jan 2 at 15:51

---

3

the easy and clear way for casting an int to enum in c#:

```
public class Program
   {
       public enum Color : int
       {
           Blue = 0,
           Black = 1,
           Green = 2,
           Gray = 3,
           Yellow =4
       }

       public static void Main(string[] args)
       {
           //from string
           Console.WriteLine((Color) Enum.Parse(typeof(Color), "Gre

           //from int
           Console.WriteLine((Color)2);
```

```
            //From number you can also
            Console.WriteLine((Color)Enum.ToObject(typeof(Color) ,2)
        }
    }
```

answered Dec 8 '18 at 5:06

Mohammad Aziz Nabizada
**104** 4

---

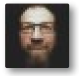
22

This is an flags enumeration aware safe convert method:

```
public static bool TryConvertToEnum<T>(this int instance, out T resu
  where T: Enum
{
  var enumType = typeof (T);
  var success = Enum.IsDefined(enumType, instance);
  if (success)
  {
    result = (T)Enum.ToObject(enumType, instance);
  }
  else
  {
    result = default(T);
  }
  return success;
}
```

edited Nov 12 '18 at 18:40

answered Mar 30 '15 at 10:08


Daniel Fisher lennybacon
**1,617** 16 24

1 This can now be improved with C# 7.3 by constraining to `Enum` instead of `struct`, meaning we don't have to rely on the runtime check! – Scott Nov 9 '18 at 17:03

11

Following is slightly better extension method

```
public static string ToEnumString<TEnum>(this int enumValue)
        {
            var enumString = enumValue.ToString();
            if (Enum.IsDefined(typeof(TEnum), enumValue))
            {
                enumString = ((TEnum) Enum.ToObject(typeof (TEnum),
enumValue)).ToString();
            }
            return enumString;
        }
```

edited Sep 6 '18 at 10:22

answered Dec 16 '16 at 6:59

Kamran Shahid
1,671 2 25 46

105

Take the following example:

```
int one = 1;
MyEnum e = (MyEnum)one;
```

edited Dec 25 '16 at 9:39

Media

**589** 2 12 33

answered Aug 27 '08 at 4:00

abigblackman
**1,081** 1 7 4

---

6

It can help you to convert any input data to user desired **enum**. Suppose you have an enum like below which by default **int**. Please add a **Default** value at first of your enum. Which is used at helpers medthod when there is no match found with input value.

```
public enum FriendType
{
    Default,
    Audio,
    Video,
    Image
}

public static class EnumHelper<T>
{
    public static T ConvertToEnum(dynamic value)
    {
        var result = default(T);
        var tempType = 0;

        //see Note below
        if (value != null &&
            int.TryParse(value.ToString(), out  tempType) &&
            Enum.IsDefined(typeof(T), tempType))
        {
            result = (T)Enum.ToObject(typeof(T), tempType);
        }
        return result;
    }
}
```

**N.B:** Here I try to parse value into int, because enum is by default **int** If you define enum like this which is **byte** type.

```
public enum MediaType : byte
{
    Default,
    Audio,
    Video,
    Image
}
```

You need to change parsing at helper method from

```
int.TryParse(value.ToString(), out  tempType)
```

to

```
byte.TryParse(value.ToString(), out  tempType)
```

I check my method for following inputs

```
EnumHelper<FriendType>.ConvertToEnum(null);
EnumHelper<FriendType>.ConvertToEnum("");
EnumHelper<FriendType>.ConvertToEnum("-1");
EnumHelper<FriendType>.ConvertToEnum("6");
EnumHelper<FriendType>.ConvertToEnum("");
EnumHelper<FriendType>.ConvertToEnum("2");
EnumHelper<FriendType>.ConvertToEnum(-1);
EnumHelper<FriendType>.ConvertToEnum(0);
EnumHelper<FriendType>.ConvertToEnum(1);
EnumHelper<FriendType>.ConvertToEnum(9);
```

sorry for my english

answered Nov 17 '16 at 12:49

reza.cse08
**3,679** 28 27

---

Slightly getting away from the original question, but I found an

18

answer to Stack Overflow question *Get int value from enum* useful.
Create a static class with `public const int` properties, allowing you to easily collect together a bunch of related `int` constants, and then not have to cast them to `int` when using them.

```
public static class Question
{
    public static readonly int Role = 2;
    public static readonly int ProjectFunding = 3;
    public static readonly int TotalEmployee = 4;
    public static readonly int NumberOfServers = 5;
    public static readonly int TopBusinessConcern = 6;
}
```

Obviously, some of the enum type functionality will be lost, but for storing a bunch of database id constants, it seems like a pretty tidy solution.

edited May 23 '17 at 10:31
Community ♦
1 1

answered Jul 17 '14 at 14:39
Ted
1,546 1 22 38

5 enums superseded the use of integer constants like this since they provide more type safety – Paul Richards Sep 1 '14 at 7:41

1 Paul, this is a method of collecting together related int constants (e.g. Database id constants) so they can be used directly without having to cast them to int every time they're used. Their type *is* integer, not for example, DatabaseIdsEnum. – Ted Sep 1 '14 at 9:37

1 There is at least one situation that I have found in which enum type safety can be unintentionally bypassed. – Thierry Sep 10 '14 at 17:33

42

For numeric values, this is safer as it will return an object no matter what:

```
public static class EnumEx
{
    static public bool TryConvert<T>(int value, out T result)
    {
        result = default(T);
        bool success = Enum.IsDefined(typeof(T), value);
        if (success)
        {
            result = (T)Enum.ToObject(typeof(T), value);
        }
        return success;
    }
}
```

edited Jan 7 '16 at 20:09
Peter Mortensen
**13.9k** 19 87 113

answered Feb 21 '13 at 15:22
Sébastien Duval
**491** 6 6

This does not work with flag enums – Daniel Fisher lennybacon Mar 30 '15 at 10:00

214

Alternatively, use an extension method instead of a one-liner:

```
public static T ToEnum<T>(this string enumString)
{
    return (T) Enum.Parse(typeof (T), enumString);
}
```

***Usage:***

```
Color colorEnum = "Red".ToEnum<Color>();
```

OR

```
string color = "Red";
var colorEnum = color.ToEnum<Color>();
```

edited Jan 7 '16 at 20:07
Peter Mortensen
**13.9k** 19 87 113

answered Nov 11 '11 at 13:27
Abdul Munim
**15.8k** 5 44 57

7 For processing user input, it's probably a good idea to call the overload of Enum.Parse that is allows you to specify that the comparison NOT be case sensitive (i.e. a user typing "red" (lowercase) would crash the above code without this change.) – BrainSlugs83 Jun 4 '13 at 20:56

33 Cool, except that it is not the question. – nawfal Jun 8 '13 at 21:41

6 Handy, but the question specifically asks about ints. – BJury May 27 '15 at 10:18

1 this also works if the string is an integer, e.g. "2" – TruthOf42 Oct 6 '16 at 19:19

2 This will throw an exception if enumString is null (had a similar issue yesterday). Consider using TryParse instead of Parse. TryParse will also check if T is an Enum Type – Justin Oct 18 '16 at 15:03

I don't know anymore where I get the part of this enum extension,

8

but it is from stackoverflow. I am sorry for this! But I took this one and modified it for enums with Flags. For enums with Flags I did this:

```
    public static class Enum<T> where T : struct
    {
       private static readonly IEnumerable<T> All = Enum.GetValues(typ
       private static readonly Dictionary<int, T> Values = All.ToDicti
  Convert.ToInt32(k));

       public static T? CastOrNull(int value)
       {
          T foundValue;
          if (Values.TryGetValue(value, out foundValue))
          {
             return foundValue;
          }

          // For enums with Flags-Attribut.
          try
          {
             bool isFlag = typeof(T).GetCustomAttributes(typeof(FlagsA
  false).Length > 0;
             if (isFlag)
             {
                int existingIntValue = 0;

                foreach (T t in Enum.GetValues(typeof(T)))
                {
                   if ((value & Convert.ToInt32(t)) > 0)
                   {
                      existingIntValue |= Convert.ToInt32(t);
                   }
                }
                if (existingIntValue == 0)
                {
                   return null;
                }

                return (T)(Enum.Parse(typeof(T), existingIntValue.ToSt
             }
          }
          catch (Exception)
          {
             return null;
          }
          return null;
```

```
      }
   }
```

Example:

```
[Flags]
public enum PetType
{
  None = 0, Dog = 1, Cat = 2, Fish = 4, Bird = 8, Reptile = 16, Othe
};

integer values
1=Dog;
13= Dog | Fish | Bird;
96= Other;
128= Null;
```

answered Jan 7 '16 at 11:40

Franki1986
**526** 5 19

---

13

From a string: (Enum.Parse is out of Date, use Enum.TryParse)

```
enum Importance
{}

Importance importance;

if (Enum.TryParse(value, out importance))
{
}
```

answered Nov 21 '14 at 0:32

Will Yu
**456** 5 12

3 The question specifically asks about integers. – BJury May 27 '15 at 10:13

3 Will Yu please edit your answer to let everyone know Enum.TryParse will work on a string of the value or name of the enum (I couldn't resist) – JeremyWeir Feb 10 '16 at 5:37

Jeremy, Weir working on that (couldn't resist either). – huysentruitw Jan 29 '18 at 12:20

14

This parses integers or strings to a target enum with partial matching in dot.NET 4.0 using generics like in Tawani's utility class above. I am using it to convert command-line switch variables which may be incomplete. Since an enum cannot be null, you should logically provide a default value. It can be called like this:

```
var result = EnumParser<MyEnum>.Parse(valueToParse, MyEnum.FirstValu
```

Here's the code:

```
using System;

public class EnumParser<T> where T : struct
{
    public static T Parse(int toParse, T defaultVal)
    {
        return Parse(toParse + "", defaultVal);
    }
    public static T Parse(string toParse, T defaultVal)
    {
        T enumVal = defaultVal;
        if (defaultVal is Enum && !String.IsNullOrEmpty(toParse))
        {
            int index;
            if (int.TryParse(toParse, out index))
            {
                Enum.TryParse(index + "", out enumVal);
            }
            else
            {
                if (!Enum.TryParse<T>(toParse + "", true, out enumVa
                {
                    MatchPartialName(toParse, ref enumVal);
                }
            }
        }
        return enumVal;
    }

    public static void MatchPartialName(string toParse, ref T enumVa
    {
        foreach (string member in enumVal.GetType().GetEnumNames())
        {
            if (member.ToLower().Contains(toParse.ToLower()))
            {
                if (Enum.TryParse<T>(member + "", out enumVal))
                {
                    break;
                }
            }
        }
    }
}
```

**FYI:** The question was about integers, which nobody mentioned will also explicitly convert in Enum.TryParse()

edited Jul 30 '14 at 22:16

answered Jul 30 '14 at 20:02

CZahrobsky
570 4 7

---

10

In my case, I needed to return the enum from a WCF service. I also needed a friendly name, not just the enum.ToString().

Here's my WCF Class.

```
[DataContract]
public class EnumMember
{
    [DataMember]
    public string Description { get; set; }

    [DataMember]
    public int Value { get; set; }

    public static List<EnumMember> ConvertToList<T>()
    {
        Type type = typeof(T);

        if (!type.IsEnum)
        {
            throw new ArgumentException("T must be of type enumerati
        }

        var members = new List<EnumMember>();

        foreach (string item in System.Enum.GetNames(type))
        {
            var enumType = System.Enum.Parse(type, item);

            members.Add(
                new EnumMember() { Description = enumType.GetDescrip
((IConvertible)enumType).ToInt32(null) });
        }

        return members;
    }
}
```

Here's the Extension method that gets the Description from the Enum.

```
    public static string GetDescriptionValue<T>(this T source)
    {
        FieldInfo fileInfo = source.GetType().GetField(source.ToStri
        DescriptionAttribute[] attributes =
(DescriptionAttribute[])fileInfo.GetCustomAttributes(typeof(Descript
false);

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

Implementation:

```
return EnumMember.ConvertToList<YourType>();
```

answered Jul 2 '14 at 14:58

LawMan
**2,529** 21 24

---

137

I think to get a complete answer, people have to know how enums work internally in .NET.

**How stuff works**

An enum in .NET is a structure that maps a set of values (fields) to a basic type (the default is `int`). However, you can actually choose the integral type that your enum maps to:

```
public enum Foo : short
```

In this case the enum is mapped to the `short` data type, which means it will be stored in memory as a short and will behave as a short when you cast and use it.

If you look at it from a IL point of view, a (normal, int) enum looks like this:

```
.class public auto ansi serializable sealed BarFlag extends System.E
{
    .custom instance void System.FlagsAttribute::.ctor()
    .custom instance void ComVisibleAttribute::.ctor(bool) = { bool(

    .field public static literal valuetype BarFlag AllFlags = int32(
    .field public static literal valuetype BarFlag Foo1 = int32(1)
    .field public static literal valuetype BarFlag Foo2 = int32(0x20

    // and so on for all flags or enum values

    .field public specialname rtspecialname int32 value__
}
```

What should get your attention here is that the `value__` is stored separately from the enum values. In the case of the enum `Foo` above, the type of `value__` is int16. This basically means that you can store whatever you want in an enum, **as long as the types match**.

At this point I'd like to point out that `System.Enum` is a value type, which basically means that `BarFlag` will take up 4 bytes in memory and `Foo` will take up 2 -- e.g. the size of the underlying type (it's actually more complicated than that, but hey...).

**The answer**

So, if you have an integer that you want to map to an enum, the runtime only has to do 2 things: copy the 4 bytes and name it something else (the name of the enum). Copying is implicit because the data is stored as value type - this basically means that if you use unmanaged code, you can simply interchange enums and integers without copying data.

To make it safe, I think it's a best practice to **know that the underlying types are the same or implicitly convertible** and to ensure the enum values exist (they aren't checked by default!).

To see how this works, try the following code:

```
public enum MyEnum : int
{
    Foo = 1,
    Bar = 2,
    Mek = 5
}

static void Main(string[] args)
{
    var e1 = (MyEnum)5;
    var e2 = (MyEnum)6;

    Console.WriteLine("{0} {1}", e1, e2);
    Console.ReadLine();
}
```

Note that casting to `e2` also works! From the compiler perspective above this makes sense: the `value__` field is simply filled with either 5 or 6 and when `Console.WriteLine` calls `ToString()` , the name of `e1` is resolved while the name of `e2` is not.

If that's not what you intended, use `Enum.IsDefined(typeof(MyEnum), 6)` to check if the value you are casting maps to a defined enum.

Also note that I'm explicit about the underlying type of the enum, even though the compiler actually checks this. I'm doing this to ensure I don't run into any surprises down the road. To see these surprises in action, you can use the following code (actually I've seen this happen a lot in database code):

```
public enum MyEnum : short
{
    Mek = 5
}

static void Main(string[] args)
{
    var e1 = (MyEnum)32769; // will not compile, out of bounds for a

    object o = 5;
    var e2 = (MyEnum)o;     // will throw at runtime, because o is o
```

```
        Console.WriteLine("{0} {1}", e1, e2);
        Console.ReadLine();
    }
```

answered Apr 3 '14 at 7:39

atlaste
**23.4k** 2 41 59

7 I realize this is an old post, but how do you gain this level of knowledge in c#? Is this from reading through the C# specification? – Rolan Nov 15 '15 at 0:19

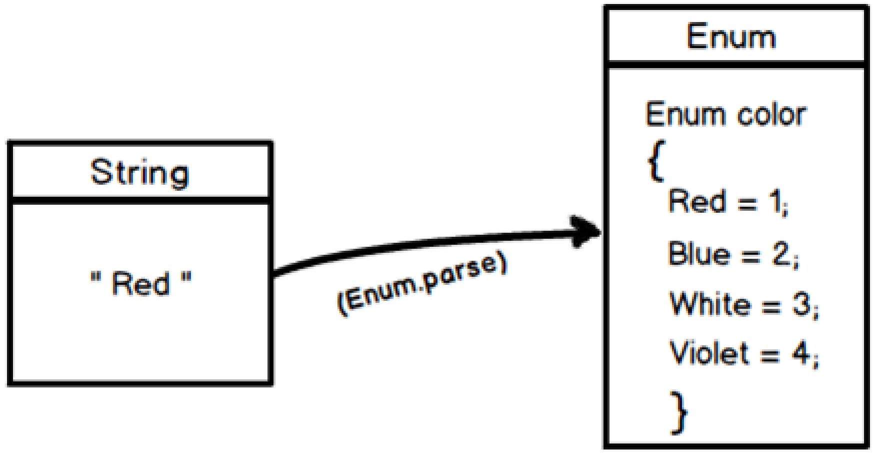
17 @Rolan I sometimes wish more people would ask that. :-) To be honest I don't really know; I try to understand how things work and get information wherever I can get it. I did read the C# standard, but I also regularly decompile code with Reflector (I even look at the x86 assembler code a lot) and do tons of little experiments. Also, knowing about other languages helps in this case; I've been doing CS for about 30 years now, and at some point certain things become 'logical' - f.ex. an enum should integral types, because otherwise interop will break (or your performance will go down the drain). – atlaste Nov 15 '15 at 11:38

7 I believe the key to doing software engineering properly is knowing how stuff works. For me that means that if you write a piece of code, you know how it roughly translates to f.ex. processor operations and memory fetches / writes. If you ask how to get to that level, I'd suggest building a ton of small test cases, making them tougher as you go, try to predict the outcome every time, and test them afterwards (incl. decompilation, etc). After figuring out all the details and all the characteristics, you can check if you got it right in the (dull) standard. At least, that would be my approach. – atlaste Nov 15 '15 at 11:43

Fantastic answer, thanks! In your last code sample, it throws an exception at runtime because o is an object. You can cast an int variable to a short as long as it falls within the short range. – gravidThoughts Aug 11 '16 at 17:04

@gravidThoughts Thanks. Actually it's an unboxing operation, so it won't do any implicit conversions like the ones you describe. Casting is sometimes confusing in C# if you don't know the details... Anyhow, because `int` != `short` , it will throw (unboxing fails). If you do `object o = (short)5;` , it will work, because then the types will match. It's not about the range, it's really about the type. – atlaste Aug 12 '16 at 6:43

20



To convert a string to ENUM or int to ENUM constant we need to use Enum.Parse function. Here is a youtube video [https://www.youtube.com/watch?v=4nhx4VwdRDk](https://www.youtube.com/watch?v=4nhx4VwdRDk) which actually demonstrate's with string and the same applies for int.

The code goes as shown below where "red" is the string and "MyColors" is the color ENUM which has the color constants.

```
MyColors EnumColors = (MyColors)Enum.Parse(typeof(MyColors), "Red");
```

answered Feb 5 '14 at 12:15
Shivprasad Koirala
17.2k 6 61 56

Different ways to cast **to and from** `Enum`

8

```
enum orientation : byte
{
 north = 1,
 south = 2,
 east = 3,
 west = 4
}

class Program
{
  static void Main(string[] args)
  {
    orientation myDirection = orientation.north;
    Console.WriteLine(“myDirection = {0}”, myDirection); //output myl
    Console.WriteLine((byte)myDirection); //output 1

    string strDir = Convert.ToString(myDirection);
        Console.WriteLine(strDir); //output north

    string myString = “north”; //to convert string to Enum
    myDirection = (orientation)Enum.Parse(typeof(orientation),myStri


 }
}
```

answered Jan 8 '14 at 15:18
gmail user
**1,696** 4 26 38

---

40

If you're ready for the 4.0 .NET Framework, there's a new *Enum.TryParse()* function that's very useful and plays well with the [Flags] attribute. See *Enum.TryParse Method (String, TEnum%)*

edited Aug 31 '12 at 20:33
Peter Mortensen
**13.9k** 19 87 113

answered Nov 1 '11 at 14:58

Ryan Russon
**821** 8 15

19 That's useful when converting from a string. But not when converting from an int. – CodesInChaos Nov 1 '11 at 15:08

---

25

Sometimes you have an object to the `MyEnum` type. Like

```
var MyEnumType = typeof(MyEnumType);
```

Then:

```
Enum.ToObject(typeof(MyEnum), 3)
```

edited Aug 31 '12 at 20:30

Peter Mortensen
**13.9k** 19 87 113

answered Jul 2 '10 at 14:41

L. D.
**259** 3 3

---

60

I am using this piece of code to cast int to my enum:

```
if (typeof(YourEnum).IsEnumDefined(valueToCast)) return (YourEnum)va
else { //handle it here, if its not defined }
```

I find it the best solution.

answered Oct 21 '11 at 10:05

MSkuta
815 8 14

1 Does not work with flags enums – Daniel Fisher lennybacon Mar 30 '15 at 10:09

1 this is good. i was surprised there's not an exception when casting an invalid value to an int-backed enum. – orion elenzil Nov 20 '15 at 0:50

This actually is not so different than the top-rated answer. That answer also discusses using Enum.IsDefined after you've casted the string to the Enum type. So even if the string was casted without error, Enum.IsDefined will still catch it – mmcrae Dec 20 '17 at 16:17

48

Below is a nice utility class for Enums

```
public static class EnumHelper
{
    public static int[] ToIntArray<T>(T[] value)
    {
        int[] result = new int[value.Length];
        for (int i = 0; i < value.Length; i++)
            result[i] = Convert.ToInt32(value[i]);
        return result;
    }

    public static T[] FromIntArray<T>(int[] value)
    {
        T[] result = new T[value.Length];
        for (int i = 0; i < value.Length; i++)
            result[i] = (T)Enum.ToObject(typeof(T),value[i]);
        return result;
    }


    internal static T Parse<T>(string value, T defaultValue)
    {
        if (Enum.IsDefined(typeof(T), value))
            return (T) Enum.Parse(typeof (T), value);
        int num;
        if(int.TryParse(value,out num))
        {
            if (Enum.IsDefined(typeof(T), num))
                return (T)Enum.ToObject(typeof(T), num);
        }

        return defaultValue;
    }
}
```

answered Sep 7 '10 at 4:42

Tawani
**6,234** 19 72 100

---

782

Just cast it:

```
MyEnum e = (MyEnum)3;
```

You can check if it's in range using Enum.IsDefined:

```
if (Enum.IsDefined(typeof(MyEnum), 3)) { ... }
```

answered Aug 27 '08 at 4:01

Matt Hamilton
**165k** 56 358 307

202 Beware you can't use Enum.IsDefined if you use the Flags attribute and the value is a combination of flags for example: Keys.L | Keys.Control – dtroy Jul 31 '09 at 4:49

13 Regarding `Enum.IsDefined` , be aware that it can be dangerous: msdn.microsoft.com/en-us/library/ms229025(VS.90).aspx – adrian Dec 4 '13 at 11:26

1 I prefer this definition: **"Returns an indication whether a constant**

**with a specified value exists in a specified enumeration"** from MSDN – Pap Aug 18 '14 at 19:13

1 ...Because your definition can be misleading, because you are saying: **"...check if it's in range..."** which implies within a range of numbers with starting and ending limits... – Pap Aug 18 '14 at 19:20

1 @mac9416 I've tried to give a succinct example at gist.github.com/alowdon/f7354cda97bac70b44e1c04bc0991bcc - basically by using `IsDefined` to check input values, you leave yourself vulnerable to people adding new enum values later which would pass an `IsDefined` check (since the new value exists in the new code), but which might not work with the original code you wrote. It's therefore safer to explicitly specify the enum values that your code is able to handle. – adrian Nov 12 '18 at 22:43

**protected** by NullPoiнтея Jun 10 '13 at 5:16

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?