Convert a string to an enum in C#

Asked 11 years ago Active 1 month ago Viewed 587k times

754

142

What's the best way to convert a string to an enumeration value in C#?

I have an HTML select tag containing the values of an enumeration. When the page is posted, I want to pick up the value (which will be in the form of a string) and convert it to the enumeration value.

In an ideal world, I could do something like this:

```
StatusEnum MyStatus = StatusEnum.Parse("Active");
```

but that isn't a valid code.

c# string enums

edited Mar 7 at 8:29
Mehdi
317 1 11

asked Aug 19 '08 at 12:51
Ben Mills
10.8k 12 35 37

22 Answers

1288

In .NET Core and .NET >4 there is a generic parse method:

```
Enum.TryParse("Active", out StatusEnum myStatus);
```

This also includes C#7's new inline `out` variables, so this does the try-parse, conversion to the explicit enum type and initialises+populates the `myStatus` variable.

If you have access to C#7 and the latest .NET this is the best way.

In .NET it's rather ugly (until 4 or above):

```
StatusEnum MyStatus = (StatusEnum) Enum.Parse(typeof(StatusEnum), "Active", true);
```

I tend to simplify this with:

```
public static T ParseEnum<T>(string value)
{
    return (T) Enum.Parse(typeof(T), value, true);
}
```

Then I can do:

```
StatusEnum MyStatus = EnumUtil.ParseEnum<StatusEnum>("Active");
```

One option suggested in the comments is to add an extension, which is simple enough:

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

StatusEnum MyStatus = "Active".ToEnum<StatusEnum>();
```

Finally, you may want to have a default enum to use if the string cannot be parsed:

```
public static T ToEnum<T>(this string value, T defaultValue)
{
    if (string.IsNullOrEmpty(value))
    {
        return defaultValue;
    }

    T result;
    return Enum.TryParse<T>(value, true, out result) ? result : defaultValue;
}
```

```
StatusEnum MyStatus = "Active".ToEnum(StatusEnum.None);
```

However, I would be careful adding an extension method like this to `string` as (without namespace control) it will appear on all instances of `string` whether they hold an enum or not (so `1234.ToString().ToEnum(StatusEnum.None)` would be valid but nonsensical) . It's often be best to avoid cluttering Microsoft's core classes with extra methods that only apply in very specific contexts unless your entire development team has a very good understanding of what those extensions do.

edited Feb 16 '17 at 16:58

answered Aug 19 '08 at 12:54
Keith
99.3k 62 246 364

13 If performace is important (which always is) chk answer given by Mckenzieg1 below : stackoverflow.com/questions/16100/… – Nash Jul 19 '09 at 19:04

25 @avinashr is right about @McKenzieG1's answer, but it isn't ALWAYS important. For instance it would be a pointless micro optimisation to worry about enum parsing if you were making a DB call for each parse. – Keith Jul 19 '09 at 20:19

4 @H.M. I don't think an extension is appropriate here - it's a bit of a special case and an extension would apply to *every* string. If you really wanted to do it though it would be a trivial change. – Keith Aug 22 '13 at 10:17

7 How about Enum.TryParse? – Elaine Jun 3 '14 at 8:39

12 very nice. you need a where T : struct in your last example. – bbrik Dec 22 '15 at 17:13

295

Use `Enum.TryParse<T>(String, T)` (≥ .NET 4.0):

```
StatusEnum myStatus;
Enum.TryParse("Active", out myStatus);
```

It can be simplified even further with C# 7.0's parameter type inlining:

```
Enum.TryParse("Active", out StatusEnum myStatus);
```

edited Oct 14 '17 at 16:20

answered Dec 5 '13 at 8:22

42 Add the middle boolean parameter for case-sensitivity and this is the safest and most elegant solution by far. – DanM7 Feb 28 '14 at 22:05

15 Come on, how many of you implemented that selected answer from 2008 to only scroll down and find this is the better (modern) answer. – TEK Mar 18 '16 at 14:47

1 Good additional points here: stackoverflow.com/a/34267134/541420 – Erwin Mayer Mar 21 '16 at 14:57

@TEK I actually prefer the 2008 answer. – Zero3 Nov 24 '16 at 10:20

I don't get it. `Parse` throws explanatory exceptions for what went wrong with the conversion (value was `null`, empty or no corresponding enum constant), which is way better than `TryParse`'s boolean return value (which suppresses the concrete error) – yair Jan 4 '18 at 18:42

180

Note that the performance of `Enum.Parse()` is awful, because it is implemented via reflection. (The same is true of `Enum.ToString`, which goes the other way.)

If you need to convert strings to Enums in performance-sensitive code, your best bet is to create a `Dictionary<String,YourEnum>` at startup and use that to do your conversions.

edited Jul 26 at 1:19
Noctis
9,940 3 32 70

answered Sep 2 '08 at 2:27
McKenzieG1
9,951 7 31 39

6 I've measured 3ms to convert a string to an Enum on the first run, on a desktop computer. (Just to illustrate the level of awfullness). – Matthieu Charbonnier Nov 26 '17 at 21:00

8 Wow 3ms is orders of magnitude of terrible – John Stock Dec 14 '17 at 3:25

can you add a code sample around this, so we get an idea on how to replace and use – transformer Jan 5 '18 at 7:32

@transformer here we go dotnetfiddle.net/sDEvpT – Shekhar Reddy Feb 23 '18 at 8:00

86

You're looking for Enum.Parse.

```
SomeEnum enum = (SomeEnum)Enum.Parse(typeof(SomeEnum), "EnumValue");
```

27

You can use extension methods now:

```
public static T ToEnum<T>(this string value, bool ignoreCase = true)
{
    return (T) Enum.Parse(typeof (T), value, ignoreCase);
}
```

And you can call them by the below code (here, `FilterType` is an enum type):

```
FilterType filterType = type.ToEnum<FilterType>();
```

edited Aug 30 '15 at 12:40
Peter Mortensen
14.4k 19 88 117

answered Feb 10 '14 at 10:12
Foyzul Karim
2,507 3 37 57

1 I have updated this to take the value as object and cast it to string inside this method. This way I can take an int value .ToEnum instead of strings only. – SollyM Feb 14 '14 at 12:05

2 @SollyM I'd say that's a horrible idea cause then this extension method will apply to *all* object types. Two extension methods, one for string and one for int, would be cleaner and much safer in my opinion. – Svish Dec 9 '14 at 11:56

@Svish, that's true. The only reason I did this is because our code is used internally only and I wanted to avoid writing 2 extensions. And since the only time we convert to Enum is with string or int, I didn't see it being a problem otherwise. – SollyM Dec 10 '14 at 14:36

3 @SollyM Internal or not, I'm still the one maintaining and using my code :P I would be annoyed if I got up a ToEnum in every intellisense menu, and like you say, since the only time you convert to an enum is from string or int, you can be pretty sure you'll only need those two methods. And two methods aren't that much more than one, especially when they are this small and of the utility type :P – Svish Dec 11 '14 at 11:09

18

```
object Enum.Parse(System.Type enumType, string value, bool ignoreCase);
```

So if you had an enum named mood it would look like this:

```
        Happy,
        Sad
    }

    // ...
    Mood m = (Mood) Enum.Parse(typeof(Mood), "Happy", true);
    Console.WriteLine("My mood is: {0}", m.ToString());
```

answered Aug 19 '08 at 12:58
brendan
22.7k 16 62 105

brillant answer ! – Sinan ÇALIŞKAN Jul 12 '16 at 9:47

17

BEWARE:

```
enum Example
{
    One = 1,
    Two = 2,
    Three = 3
}
```

`Enum.(Try)Parse()` **accepts multiple, comma-separated arguments, and combines them with binary 'or'** `|` . You cannot disable this and in my opinion you almost never want it.

```
var x = Enum.Parse("One,Two"); // x is now Three
```

Even if `Three` was not defined, `x` would still get int value `3` . That's even worse: Enum.Parse() can give you a value that is not even defined for the enum!

I would not want to experience the consequences of users, willingly or unwillingly, triggering this behavior.

Additionally, as mentioned by others, performance is less than ideal for large enums, namely linear in the number of possible values.

I suggest the following:

```
    public static bool TryParse<T>(string value, out T result)
        where T : struct
    {
        var cacheKey = "Enum_" + typeof(T).FullName;

        // [Use MemoryCache to retrieve or create&store a dictionary for this enum,
permanently or temporarily.
        // [Implementation off-topic.]
        var enumDictionary = CacheHelper.GetCacheItem(cacheKey, CreateEnumDictionary<T>,
EnumCacheExpiration);

        return enumDictionary.TryGetValue(value.Trim(), out result);
    }

    private static Dictionary<string, T> CreateEnumDictionary<T>()
    {
        return Enum.GetValues(typeof(T))
            .Cast<T>()
            .ToDictionary(value => value.ToString(), value => value,
StringComparer.OrdinalIgnoreCase);
    }
```

edited Mar 22 '16 at 8:21

answered Dec 14 '15 at 12:31
Timo
3,711 2 27 37

4 In fact this is very useful to know that `Enum.(Try)Parse accepts multiple, comma-separated arguments, and combines them with binary 'or'` . Means you can set up your enum values as powers of 2 and you have a very easy way to parse multiple boolean flags, eg. "UseSSL,NoRetries,Sync". In fact that's probably what it was designed for. – pcdev Aug 22 '18 at 8:00

15

Enum.Parse is your friend:

```
StatusEnum MyStatus = (StatusEnum)Enum.Parse(typeof(StatusEnum), "Active");
```

edited Aug 30 '15 at 12:37
Peter Mortensen
14.4k 19 88 117

answered Aug 19 '08 at 12:55
tags2k
30.7k 30 72 100

11

You can extend the accepted answer with a default value to avoid exceptions:

```
public static T ParseEnum<T>(string value, T defaultValue) where T : struct
{
    try
    {
        T enumValue;
        if (!Enum.TryParse(value, true, out enumValue))
        {
            return defaultValue;
        }
        return enumValue;
    }
    catch (Exception)
    {
        return defaultValue;
    }
}
```

Then you call it like:

```
StatusEnum MyStatus = EnumUtil.ParseEnum("Active", StatusEnum.None);
```

If the default value is not an enum the Enum.TryParse would fail and throw an exception which is catched.

After years of using this function in our code on many places maybe it's good to add the information that this operation costs performance!

edited Jul 2 at 9:45

answered Dec 9 '14 at 11:43
Nelly
432 5 14

I don't like default values. It can lead to unpredictable results. – Daniël Tulp May 26 '16 at 7:46

3 when will this ever throw an exception? – andleer May 28 '16 at 15:28

Right Solution for my problem, Thanks... – Nayan Hodar Jun 6 '18 at 6:35

8

```
public static TEnum ParseEnum<TEnum>(string value) where TEnum : struct
{
    TEnum tmp;
    if (!Enum.TryParse<TEnum>(value, true, out tmp))
    {
        tmp = new TEnum();
    }
    return tmp;
}
```

answered Aug 30 '12 at 15:07

gap
1,802 2 24 33

7

```
// str.ToEnum<EnumType>()
T static ToEnum<T>(this string str)
{
    return (T) Enum.Parse(typeof(T), str);
}
```

edited Sep 5 '14 at 15:06

answered Aug 19 '08 at 13:13

Mark Cidade
86.9k 29 210 227

5

Parses string to TEnum without try/catch and without TryParse() method from .NET 4.5

```
/// <summary>
/// Parses string to TEnum without try/catch and .NET 4.5 TryParse()
/// </summary>
public static bool TryParseToEnum<TEnum>(string probablyEnumAsString_, out TEnum
enumValue_) where TEnum : struct
{
    enumValue_ = (TEnum)Enum.GetValues(typeof(TEnum)).GetValue(0);
    if(!Enum.IsDefined(typeof(TEnum), probablyEnumAsString_))
        return false;
    return true;
}
```

edited Oct 30 '13 at 12:46

answered Oct 17 '13 at 16:04

jite.gs
51 1 3

1 You need to provide some description to your code. – Maxim Kolesnikov Oct 17 '13 at 16:24

1 Whether it is necessary to make a description if the code already contains a description? Ok, I did this :) – jite.gs Oct 30 '13 at 12:43

Thanks dude, that was good. – AmirReza-Farahlagha Oct 1 '18 at 11:36

3

Super simple code using TryParse:

```
var value = "Active";

StatusEnum status;
if (!Enum.TryParse<StatusEnum>(value, out status))
    status = StatusEnum.Unknown;
```

answered Nov 25 '16 at 2:30

Brian Rice
1,401 1 20 36

2

I like the extension method solution..

```
namespace System
{
    public static class StringExtensions
    {

        public static bool TryParseAsEnum<T>(this string value, out T output) where T :
struct
```

```
                var isEnum = Enum.TryParse(value, out result);

                output = isEnum ? result : default(T);

                return isEnum;
            }
        }
    }
```

Here below my implementation with tests.

```
 using static Microsoft.VisualStudio.TestTools.UnitTesting.Assert;
 using static System.Console;

 private enum Countries
     {
         NorthAmerica,
         Europe,
         Rusia,
         Brasil,
         China,
         Asia,
         Australia
     }

    [TestMethod]
         public void StringExtensions_On_TryParseAsEnum()
         {
             var countryName = "Rusia";

             Countries country;
             var isCountry = countryName.TryParseAsEnum(out country);

             WriteLine(country);

             IsTrue(isCountry);
             AreEqual(Countries.Rusia, country);

             countryName = "Don't exist";

             isCountry = countryName.TryParseAsEnum(out country);

             WriteLine(country);
```

```
enumeration
        }
```

answered Oct 1 '15 at 21:57

alhpe
629 9 13

1

```
public static T ParseEnum<T>(string value)            //function declaration
{
    return (T) Enum.Parse(typeof(T), value);
}

Importance imp = EnumUtil.ParseEnum<Importance>("Active");   //function call
```

====================A Complete Program====================

```
using System;

class Program
{
    enum PetType
    {
    None,
    Cat = 1,
    Dog = 2
    }

    static void Main()
    {

    // Possible user input:
    string value = "Dog";

    // Try to convert the string to an enum:
    PetType pet = (PetType)Enum.Parse(typeof(PetType), value);

    // See if the conversion succeeded:
    if (pet == PetType.Dog)
    {
}
-------------
Output

Equals dog.
```

answered Aug 18 '15 at 5:28
Rae Lee
801 8 10

1

I used class (strongly-typed version of Enum with parsing and performance improvements). I found it on GitHub, and it should work for .NET 3.5 too. It has some memory overhead since it buffers a dictionary.

```
StatusEnum MyStatus = Enum<StatusEnum>.Parse("Active");
```

The blogpost is *Enums – Better syntax, improved performance and TryParse in NET 3.5*.

And code: https://github.com/damieng/DamienGKit/blob/master/CSharp/DamienG.Library/System/EnumT.cs

edited Aug 30 '15 at 16:33
Peter Mortensen
14.4k 19 88 117

answered Jul 1 '15 at 9:39
Patrik Lindström
750 3 11 19

1

For performance this might help:

```
    private static Dictionary<Type, Dictionary<string, object>> dicEnum = new
Dictionary<Type, Dictionary<string, object>>();
    public static T ToEnum<T>(this string value, T defaultValue)
    {
        var t = typeof(T);
        Dictionary<string, object> dic;
        if (!dicEnum.ContainsKey(t))
        {
            dic = new Dictionary<string, object>();
            dicEnum.Add(t, dic);
            else
                dic = dicEnum[t];
            if (!dic.ContainsKey(value))
                return defaultValue;
            else
                return (T)dic[value];
        }
```

answered Jun 22 '16 at 14:10
Koray
891 1 14 27

1

I found that here the case with enum values that have EnumMember value was not considered. So here we go:

```
using System.Runtime.Serialization;

public static TEnum ToEnum<TEnum>(this string value, TEnum defaultValue) where TEnum :
struct
{
    if (string.IsNullOrEmpty(value))
    {
        return defaultValue;
    }

    TEnum result;
    var enumType = typeof(TEnum);
    foreach (var enumName in Enum.GetNames(enumType))
    {
        var fieldInfo = enumType.GetField(enumName);
        var enumMemberAttribute = ((EnumMemberAttribute[])
fieldInfo.GetCustomAttributes(typeof(EnumMemberAttribute), true)).FirstOrDefault();
        if (enumMemberAttribute?.Value == value)
        {
            return Enum.TryParse(enumName, true, out result) ? result : defaultValue;
        }
    }

    return Enum.TryParse(value, true, out result) ? result : defaultValue;
}
```

```
public enum OracleInstanceStatus
{
    Unknown = -1,
    Started = 1,
    Mounted = 2,
    Open = 3,
    [EnumMember(Value = "OPEN MIGRATE")]
    OpenMigrate = 4
}
```

answered Oct 4 '16 at 16:40

isxaker
3,446 5 40 68

1

You have to use Enum.Parse to get the object value from Enum, after that you have to change the object value to specific enum value. Casting to enum value can be do by using Convert.ChangeType. Please have a look on following code snippet

```
public T ConvertStringValueToEnum<T>(string valueToParse){
    return Convert.ChangeType(Enum.Parse(typeof(T), valueToParse, true), typeof(T));
}
```

edited Feb 12 '17 at 12:11

answered Feb 8 '17 at 11:33

Bartosz Gawron
81 5

1

Try this sample:

```
 public static T GetEnum<T>(string model)
    {
        var newModel = GetStringForEnum(model);

        if (!Enum.IsDefined(typeof(T), newModel))
        {
            return (T)Enum.Parse(typeof(T), "None", true);
```

```
private static Task<string> GetStringForEnum(string model)
{
    return Task.Run(() =>
    {
        Regex rgx = new Regex("[^a-zA-Z0-9 -]");
        var nonAlphanumericData = rgx.Matches(model);
        if (nonAlphanumericData.Count < 1)
        {
            return model;
        }
        foreach (var item in nonAlphanumericData)
        {
            model = model.Replace((string)item, "");
        }
        return model;
    });
}
```

In this sample you can send every string, and set your `Enum`. If your `Enum` had data that you wanted, return that as your `Enum` type.

edited Oct 17 '18 at 14:59
meagar ♦
188k 32 284 300

answered Oct 1 '18 at 9:37
AmirReza-Farahlagha
704 7 18

1 You are overwriting `newModel` on each line, so if it contains dashes, it will not be replaced. Also, you don't have to check if the string contains anything, you can just call `Replace` anyway: `var newModel = model.Replace("-", "").Replace(" ", "");` – Lars Kristensen Oct 1 '18 at 9:52

@LarsKristensen Yeah we can create an method that remove nonalphanumeric character. – AmirReza-Farahlagha Oct 1 '18 at 9:56

0

```
    <Extension()>
Public Function ToEnum(Of TEnum)(ByVal value As String, ByVal defaultValue As TEnum)
As TEnum
    If String.IsNullOrEmpty(value) Then
        Return defaultValue
    End If

    Return [Enum].Parse(GetType(TEnum), value, True)
End Function
```

AHMED RABEE
300 4 12

0

```
public TEnum ToEnum<TEnum>(this string value, TEnum defaultValue){
if (string.IsNullOrEmpty(value))
    return defaultValue;

return Enum.Parse(typeof(TEnum), value, true);}
```

answered May 22 at 7:19
AHMED RABEE
300 4 12

protected by Bhargav Rao ♦ Feb 8 '17 at 13:06

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?