

RE: passing enum value as an argument

Source:

<http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.general/2005-07/msg00676.html>

- *From:* "Madestro" <me_no_like_spam_juanDOTromero@bowneDOTcom>
 - *Date:* Wed, 13 Jul 2005 14:57:04 -0700
-

> You mentioned it in an earlier post, but not in the post which
> specifically said that a call to TakeFirstEnum(SecondEnum.Fred)
> succeeds. You didn't even mention it in the post before that. How was
> anyone supposed to work out that you were talking about the (hopefully
> unusual) situation of working with Option Strict off in a post made 16
> hours after your only previous reference to Option Strict?

Understandable. I should not expect you to remember and be more clear on my answers.

> No it wouldn't – because unless you declare a new type for each
> constant (in which case you've basically got an enum anyway) you would
> have to use the same type (eg Int32) for different types of constant,
> at which point you **have** lost type safety.

Actually you would have to declare the SAME type for each constant, and indeed you would have an Enum, and you WOULD have to use the same type on the function that receives them so you really DON'T lose type safety.

> I think you'll agree that 5 is outside the range of the enum, and yet
> it works. How do you explain that?

You could not be more right. I actually missread the documentation. The limit is in the value size. For example, if you have an Enum defined as integer, then you could still use other values so long as they are within the range of an integer.

Totally my fault, I take it back.

> It can't. So in this case, the compiler would force you to
> disambiguate. That doesn't mean there aren't many, many cases where
> there **isn't** any ambiguity, and where the compiler could infer the
> type.

The compiler may infer the type, but not the value. The value has to be resolved by a fully qualified name. The compiler cannot possibly resolve that.

> Please explain how the code which you claim won't work works perfectly
> fine then.

RE: passing enum value as an argument

I am not sure what code you refer to.

- > By the way, your test code also fails to compile because you haven't
- > declared the return type of your Test function, and because the enum
- > you declared was called "Juan" but you declared the parameter to the
- > function to be of type "JuanEnum". Here's your code tidied up. It
- > doesn't fail where you claimed it would, by the way.

I turned Option Strict OFF to test this, which is why the function had no return type.

JuanEnum is just a typo when copying back to here. I don't copy and paste.

Yes, the code will not fail. This relates to my first answer in this posting.

Thank you!

—

Juan Romero

The successful person has the habit of doing the things failures don't like to do.

E.M. Gray

"Jon Skeet [C# MVP]" wrote:

- > Madestro <me_no_like_spam_juanDOTromero@bowneDOTcom> wrote:
- >> Once again, please read the posts above. I did talk about Option Strict.
- >
- > You mentioned it in an earlier post, but not in the post which
- > specifically said that a call to TakeFirstEnum(SecondEnum.Fred)
- > succeeds. You didn't even mention it in the post before that. How was
- > anyone supposed to work out that you were talking about the (hopefully
- > unusual) situation of working with Option Strict off in a post made 16
- > hours after your only previous reference to Option Strict?
- >
- >> I clearly stated that you could CAST to the Enum type. Obviously IMPLICIT
- >> conversions will yield an error with Option Strict ON, that is the whole
- >> point of it.
- >
- > Indeed.
- >
- >> As for the evaluation at runtime, because the compiler knows the type, a
- >> constant or variable would provide THE SAME level of type safety.
- >
- > No it wouldn't – because unless you declare a new type for each
- > constant (in which case you've basically got an enum anyway) you would
- > have to use the same type (eg Int32) for different types of constant,
- > at which point you *have* lost type safety.
- >
- > In your first post in the thread, you said:

RE: passing enum value as an argument

RE: passing enum value as an argument

```
>
> <quote>
> That would defeat the purpose of having an Enum. you might as well
> create three variables then.
> </quote>
>
> What type would you make those variables have?
>
>> The range of values IS part of the safety. If you don't believe me, read the
>> documentation and/or attempt to assign an integer outside of the range and
>> you will see what happens. E.g.:
>>
>> Enum Juan
>> Zero 'By default this is initialized to 0
>> One 'By default this is initialized to 1
>> Two 'By default this is initialized to 2
>> End Enum
>>
>> Public sub New()
>> Test(2) -----> Option Strict OFF, this will be OK
>> Test(3) -----> Option Strict OFF, this will FAIL
>> Test(CType(2,Juan)) -----> Option Strict ON, This will be OK
>> Test(CType(3,Juan)) -----> Option Strict ON, this will FAIL
>> End Sub
>
> Unlike you, I've actually tried the code. Compile and run the
> following:
>
> Option Strict On
>
> Imports System
>
> Public Enum FirstEnum
> Foo
> Bar
> Baz
> End Enum
>
> Public Class Test
>
> Shared Sub TakeFirstEnum (value As FirstEnum)
> Console.WriteLine (value)
> End Sub
>
> Shared Sub Main()
> TakeFirstEnum(CType(5, FirstEnum))
> End Sub
> End Class
>
> Compile it from the command line:
> vbc Test.vb
```

RE: passing enum value as an argument

> Run it:
> 5
>
> I think you'll agree that 5 is outside the range of the enum, and yet
> it works. How do you explain that?
>
>> As you can see I am not *entirely* wrong. Try creating another Enum and pass
>> it to the function (with casting if using Option Strict ON) and you will see
>> that the value is accepted so long as it is in the range of the enum.
>
> No, it's accepted whether or not it's in the range of the enum.
>
>> As far as ambiguity, a language can NEVER allow you to use the name
>> directly. Consider the following scenario:
>>
>> Public Class One
>> Dim Juan as integer = 0
>> Enum Sample
>> Juan = 1
>> Javier = 2
>> End Enum
>> Public Sub New()
>> console.writeline(Juan) -----> * This will print "1"
>> End Sub
>> End Class
>>
>> How will the compiler resolve this variable?
>
> It can't. So in this case, the compiler would force you to
> disambiguate. That doesn't mean there aren't many, many cases where
> there *isn't* any ambiguity, and where the compiler could infer the
> type.
>
> <snip>
>
>> Finally, I never post anything until I try it myself.
>
> Please explain how the code which you claim won't work works perfectly
> fine then.
>
> By the way, your test code also fails to compile because you haven't
> declared the return type of your Test function, and because the enum
> you declared was called "Juan" but you declared the parameter to the
> function to be of type "JuanEnum". Here's your code tidied up. It
> doesn't fail where you claimed it would, by the way.
>
> Option Strict On
>
> Imports System
>
> Public Enum Juan

RE: passing enum value as an argument

```
> Zero 'By default this is initialized to 0
> One 'By default this is initialized to 1
> Two 'By default this is initialized to 2
> End Enum
>
>
> Public Class Foo
>
> Public Shared Sub Test(ByVal e as Juan)
> End Sub
>
>
> Shared Sub Main()
> Test(ctype(2,Juan))
> Test(ctype(3,Juan))
> End Sub
> End Class
>
>
> ---
> Jon Skeet – <skeet@xxxxxxxx>
> http://www.pobox.com/~skeet
> If replying to the group, please do not mail me too
>
.
```

• *Follow-Ups:*

- ◆ ***RE: passing enum value as an argument***
◇ *From:* Jon Skeet [C# MVP]

• *References:*

- ◆ ***passing enum value as an argument***
◇ *From:* Glenn Venzke
- ◆ ***RE: passing enum value as an argument***
◇ *From:* Madestro
- ◆ ***RE: passing enum value as an argument***
◇ *From:* Jon Skeet [C# MVP]
- ◆ ***RE: passing enum value as an argument***
◇ *From:* Madestro
- ◆ ***RE: passing enum value as an argument***
◇ *From:* Jon Skeet [C# MVP]
- ◆ ***RE: passing enum value as an argument***
◇ *From:* Madestro
- ◆ ***RE: passing enum value as an argument***
◇ *From:* Jon Skeet [C# MVP]
- ◆ ***RE: passing enum value as an argument***
◇ *From:* Madestro
- ◆ ***RE: passing enum value as an argument***
◇ *From:* Jon Skeet [C# MVP]

RE: passing enum value as an argument

◆ ***RE: passing enum value as an argument***

◇ *From:* Madestro

◆ ***RE: passing enum value as an argument***

◇ *From:* Jon Skeet [C# MVP]

- Prev by Date: ***Re: How easy managed codes be reversed engineered?***
- Next by Date: ***Re: How easy managed codes be reversed engineered?***
- Previous by thread: ***RE: passing enum value as an argument***
- Next by thread: ***RE: passing enum value as an argument***
- Index(es):
 - ◆ ***Date***
 - ◆ ***Thread***