C# numeric constants

Asked 16 years, 3 months ago Modified 11 years, 6 months ago Viewed 13k times

19

I have the following C# code:

```
byte rule = 0;
...
rule = rule | 0x80;
```

which produces the error:

> *Cannot implicitly convert type 'int' to 'byte'. An explicit conversion exists (are you missing a cast?)*

[Update: first version of the question was wrong ... I misread the compiler output]

Adding the cast **doesn't** fix the problem:

```
rule = rule | (byte) 0x80;
```

I need to write it as:

```
rule |= 0x80;
```

Which just seems weird. Why is the `|=` operator any different to the `|` operator?

Is there any other way of telling the compiler to treat the constant as a byte?

@ **Giovanni Galbo** : yes and no. The code is dealing with the programming of the flash memory in an external device, and logically represents a single byte of memory. I could cast it later, but this seemed more obvious. I guess my C heritage is showing through too much!

@ **Jonathon Holland** : the 'as' syntax looks neater but unfortunately doesn't appear to work ... it produces:

> *The as operator must be used with a reference type or nullable type ('byte' is a non-nullable value type)*

c# casting

Share
Improve this question
Follow

edited Aug 15, 2012 at 8:02

Adi Lester
25.2k ● 12 ● 98 ● 113

asked Sep 4, 2008 at 1:57

Rob Walker
47.4k ● 15 ● 100 ● 137

10 Answers

Sorted by: Highest score (default)

34

C# does not have a literal suffix for byte. u = uint, l = long, ul = ulong, f = float, m = decimal, but no byte. You have to cast it.

Share Improve this answer Follow

answered Sep 4, 2008 at 2:03

Eric Z Beard
38.4k ● 27 ● 101 ● 147

2 user: it is not defined it the c# language specifications. There's an example here: msdn.microsoft.com/en-us/library/5bdb6693(VS.80).aspx You can declare and initialize a byte variable like this example: byte myByte = 255; In the preceding declaration, the integer literal 255 is implicitly converted from int to byte. If the integer literal exceeds the range of byte, a compilation error will occur. – blizpasta Aug 20, 2010 at 2:09

11

This works:

```
rule = (byte)(rule | 0x80);
```

Apparently the expression 'rule | 0x80' returns an int even if you define 0x80 as 'const byte 0x80'.

Share
Improve this answer
Follow

edited Sep 4, 2008 at 2:12

answered Sep 4, 2008 at 2:05

jmatthias
7,595 ● 7 ● 30 ● 37

9

The term you are looking for is "Literal" and unfortunately C# does not have a byte literal.

Here's a list of all C# literals.

Share

Improve this answer

Follow

edited Aug 15, 2012 at 8:04

Adi Lester

25.2k ● 12 ● 98 ● 113

answered Sep 4, 2008 at 2:04

FlySwat

175k ● 75 ● 248 ● 314

6

```
int rule = 0;
rule |= 0x80;
```

http://msdn.microsoft.com/en-us/library/kxszd0kx.aspx The | operator is defined for all value types. I think this will produced the intended result. The "|=" operator is an or then assign operator, which is simply shorthand for rule = rule | 0x80.

One of the niftier things about C# is that it lets you do crazy things like abuse value types simply based on their size. An 'int' is exactly the same as a byte, except the compiler will throw warnings if you try and use them as both at the same time. Simply sticking with one (in this case, int) works well. If you're concerned about 64bit readiness, you can specify int32, but all ints are int32s, even running in x64 mode.

Share

Improve this answer

Follow

edited Sep 4, 2008 at 2:20

answered Sep 4, 2008 at 2:10

David J. Sokol

3,556 ● 3 ● 32 ● 25

As you say, all ints are int32, doesn't the language (or CLR) specification make this guarantee? – Eloff May 4, 2010 at 22:59

4 int is a C# alias for Int32 and is definitely not the same as a byte. The JIT compiler is platform aware so we don't need to be. The native word size, addressing and the efficiency of operations against various native primitives are more likely to be involved in how it allocates managed data (including "primitives") in the underlying system. Consider that the same code can be compiled for .NET Micro as well as .NET CF (targeting ARM for phone and PowerPC for XBox 360). These systems may even express their numbers with different endianness which has major effects on binary operations. – TheXenocide Jul 2, 2012 at 13:57

3

According to the ECMA Specification, pg 72 there is no byte literal. Only integer literals for the types: int, uint, long, and ulong.

Share Improve this answer Follow

answered Sep 4, 2008 at 2:05

grom

16.1k ● 19 ● 65 ● 67

2

Almost five years on and nobody has actually answered the question.

A couple of answers claim that the problem is the lack of a byte literal, but this is irrelevant. If you calculate `(byte1 | byte2)` the result is of type `int`. Even if "b" were a literal suffix for byte the type of `(23b | 32b)` would still be `int`.

The accepted answer links to an MSDN article claiming that `operator|` is defined for all integral types, but this isn't true either.

`operator|` is not defined on `byte` so the compiler uses its usual overload resolution rules to pick the version that's defined on `int`. Hence, if you want to assign the result to a `byte` you need to cast it:

```
rule = (byte)(rule | 0x80);
```

The question remains, why does `rule |= 0x80;` work?

Because the C# specification has a special rule for compound assignment that allows you to omit the explicit conversion. In the compound assignment `x op= y` the rule is:

> if the selected operator is a predefined operator, if the return type of the selected operator is explicitly convertible to the type of x, and if y is implicitly convertible to the type of x or the operator is a shift operator, then the operation is evaluated as `x = (T)(x op y)`, where T is the type of x, except that x is evaluated only once.

Share Improve this answer Follow

answered Jun 2, 2013 at 16:35

arx

16.9k ● 2 ● 47 ● 64

1

Looks like you may just have to do it the ugly way: http://msdn.microsoft.com/en-us/library/5bdb6693.aspx.

Share Improve this answer Follow

answered Sep 4, 2008 at 2:01

John Rutherford

10.7k ● 7 ● 31 ● 32

Unfortunately, your only recourse is to do it just the way you have. There is no suffix to mark the literal as a byte. The | operator does not provide for implicit conversion as an

0

assignment (i.e. initialization) would.

Share Improve this answer Follow

answered Sep 4, 2008 at 2:05

Peter Meyer

26.1k ● 1 ● 35 ● 53

0

> Apparently the expression 'rule | 0x80' returns an int even if you define 0x80 as 'const byte 0x80'.

I think the rule is numbers like 0x80 defaults to int unless you include a literal suffix. So for the expression `rule | 0x80`, the result will be an int since 0x80 is an int and rule (which is a byte) can safely be converted to int.

Share Improve this answer Follow

answered Sep 4, 2008 at 2:26

jfs

16.7k ● 13 ● 63 ● 90

This is incorrect. Even if you cast 0x80 to a byte, the `|` operator gives you an int. Try: `rule = rule | (byte)0x80;` to see what I mean. You can, however, wrap the entire operation in parentheses and cast it all to byte: `rule = (byte)(rule | 0x80);` – Ethan May 31, 2012 at 19:15

0

According to the C standard, bytes ALWAYS promote to int in expressions, even constants. However, as long as both values are UNSIGNED, the high-order bits will be discarded so the operation should return the correct value.

Similarly, floats promote to double, etc.

Pull out of copy of K&R. It's all in there.

Share Improve this answer Follow

answered Jan 28, 2013 at 2:24

Bruce Bowman

1