

What is the difference between const and readonly in C#?

Asked 11 years, 1 month ago Active 12 days ago Viewed 379k times

What is the difference between `const` and `readonly` in C#?
When would you use one over the other?

1240

`c#` `.net` `const` `constants` `readonly`

307

edited Sep 26 at 22:24



Kolob Canyon

2,159 1 15 37

asked Sep 11 '08 at 8:02



Readonly

139k 96 192 196

2 The value will be initialized only once from the constructor of the class. – donstack Jan 24 '14 at 7:33

I had to look down several answers to find this link, but it's a good one. [Eric Lippert's take on immutability in C#](#) – Frank Bryce Dec 17 '15 at 16:06

12 "The readonly keyword is different from the const keyword. A const field can only be initialized at the declaration of the field. A readonly field can be initialized either at the declaration or in a constructor. Therefore, readonly fields can have different values depending on the constructor used. Also, while a const field is a compile-time constant, the readonly field can be used for runtime constants as in the following example..."
msdn.microsoft.com/en-us/library/acdd6hb7.aspx – cp.engr Sep 27 '16 at 15:06

32 Answers

1 2 next

Apart from the apparent difference of
1184

- having to declare the value at the time of a definition for a `const` VS `readonly` values can be computed dynamically but need to be assigned before the constructor exits.. after that it is frozen.
- 'const's are implicitly `static`. You use a `ClassName.ConstantName` notation to access them.

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

```
public class Const_V_ReadOnly
{
    public const int I_CONST_VALUE = 2;
    public readonly int I_RO_VALUE;
    public Const_V_ReadOnly()
    {
        I_RO_VALUE = 3;
    }
}
```

AssemblyB references AssemblyA and uses these values in code. When this is compiled,

- in the case of the `const` value, it is like a find-replace, the value 2 is 'baked into' the AssemblyB's IL. This means that if tomorrow I'll update `I_CONST_VALUE` to 20 in the future. *AssemblyB would still have 2 till I recompile it.*
- in the case of the `readonly` value, it is like a `ref` to a memory location. The value is not baked into AssemblyB's IL. This means that if the memory location is updated, AssemblyB gets the new value without recompilation. So if `I_RO_VALUE` is updated to 30, you only need to build AssemblyA. All clients do not need to be recompiled.

So if you are confident that the value of the constant won't change use a `const`.

```
public const int CM_IN_A_METER = 100;
```

But if you have a constant that may change (e.g. w.r.t. precision).. or when in doubt, use a `readonly`.

```
public readonly float PI = 3.14;
```

Update: Aku needs to get a mention coz he pointed this out first. Also I need to plug where I learned this.. [Effective C# - Bill Wagner](#)

edited Sep 22 '15 at 8:01



Irshad

2,542 5 22 41

answered Sep 11 '08 at 8:24



Gishu

106k 42 209 289

68 The `static` point seems to be the most important and useful point - `consts` are implicitly `static` – Lijo Jan 21 '13 at 14:00 ✎

24 The part about reference values is the most important one. `Const` values can be optimized away. – CodingBarfield Jun 5 '13 at 7:03

19 `readonly` variables can be changed outside the constructor (reflection). It's only the compiler that tries to hinder you from modifying the var outside

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

enforce this. The runtime also happens not to enforce that you don't change `string.Empty` to "Hello, world!", but I still wouldn't claim that this makes `string.Empty` modifiable, or that code shouldn't assume that `string.Empty` will always be a zero-length string. – user743382 Apr 14 '14 at 11:55

6 blogs.msmvps.com/jonskeet/2014/07/16/... is an interesting read only the overhead cost of readonly – CAD bloke Aug 5 '14 at 21:52



263



There is a gotcha with consts! If you reference a constant from another assembly, its value will be compiled right into the calling assembly. That way when you update the constant in the referenced assembly it won't change in the calling assembly!

edited Apr 10 '14 at 15:53



Appulus

12.6k

10

32

42

answered Sep 11 '08 at 8:15



aku

104k

30

162

199

7 On decompilation (Reflector, ILSpy, ..) a constant NEVER EVER is referenced by any one, no matter of same assembly or another assembly, so you cannot analyze the usage of a constant in compiled code at all. – springy76 Aug 19 '16 at 13:02



Constants

146



- Constants are static by default
- They must have a value at compilation-time (you can have e.g. $3.14 * 2$, but cannot call methods)
- Could be declared within functions
- Are copied into every assembly that uses them (every assembly gets a local copy of values)
- Can be used in attributes

Readonly instance fields

- Must have set value, by the time constructor exits
- Are evaluated when instance is created

Static readonly fields

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

- Must have an evaluated value by the time the static constructor is done
- It's not recommended to put ThreadStaticAttribute on these (static constructors will be executed in one thread only and will set the value for its thread; all other threads will have this value uninitialized)

edited Jun 9 '11 at 12:03



Shantanu Gupta

10.5k 50 150 260

answered Dec 2 '08 at 11:50



splattne

86.3k 46 197 244

Just to add, ReadOnly for reference types only makes the reference readonly not the values. For example:

54

```
public class Const_V_ReadOnly
{
    public const int I_CONST_VALUE = 2;
    public readonly char[] I_RO_VALUE = new Char[]{'a', 'b', 'c'};

    public UpdateReadOnly()
    {
        I_RO_VALUE[0] = 'V'; //perfectly legal and will update the value
        I_RO_VALUE = new char[]{'V'}; //will cause compiler error
    }
}
```

edited Sep 29 '08 at 21:19

answered Sep 11 '08 at 10:37

user1333

Is there any other reference type than `string` that you could use as a constant? – [springy76](#) Aug 19 '16 at 13:02

You can have `const` with reference types other than `string`, but the constant can only have the value `null` . – [Mike Rosoft](#) Apr 27 '18 at 14:28

[This explains it](#). Summary: `const` must be initialized at declaration time, `readonly` can be initialized on the constructor (and thus have a different value depending on the constructor used).

37

EDIT: See Gishu's gotcha above for the subtle difference

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



const : Can't be changed anywhere.

30



readonly : This value can only be changed in the constructor. Can't be changed in normal functions.

edited Nov 27 '13 at 22:04



Appulus

12.6k 10 32 42

answered May 21 '11 at 13:21



Deepthi

301 3 2



There is a small gotcha with readonly. A readonly field can be set multiple times within the constructor(s). Even if the value is set in two different chained constructors it is still allowed.

25



```
public class Sample {  
    private readonly string ro;  
  
    public Sample() {  
        ro = "set";  
    }  
  
    public Sample(string value) : this() {  
        ro = value; // this works even though it was set in the no-arg ctor  
    }  
}
```

answered Oct 19 '08 at 22:14



Mike Two

36.5k 7 76 94

Good to know!! Nice point. – Levit Mar 24 '16 at 7:51

24



```
public class MyClass
{
    public const double PI1 = 3.14159;
}
```

A `readonly` member is like a constant in that it represents an unchanging value. The difference is that a `readonly` member can be initialized at runtime, in a constructor, as well being able to be initialized as they are declared.

```
public class MyClass1
{
    public readonly double PI2 = 3.14159;

    //or

    public readonly double PI3;

    public MyClass2()
    {
        PI3 = 3.14159;
    }
}
```

const

- They can not be declared as `static` (they are implicitly static)
- The value of constant is evaluated at compile time
- constants are initialized at declaration only

readonly

- They can be either instance-level or static
- The value is evaluated at run time
- `readonly` can be initialized in declaration or by code in the constructor

edited Apr 27 '16 at 23:05



Michael Blackburn

2,680 1 20 18

answered Sep 17 '12 at 11:48



Sujit

2,194 9 33 50

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

20:21

Can you explain why `const` declarations can't be made inside methods? – [Minh Tran](#) Oct 29 '17 at 17:38



20



A `const` is a compile-time constant whereas `readonly` allows a value to be calculated at run-time and set in the constructor or field initializer. So, a '`const`' is always constant but '`readonly`' is read-only once it is assigned.

[Eric Lippert](#) of the C# team has more information on different types of immutability

answered Sep 11 '08 at 8:07



[Wheelie](#)

3,070 1 27 33



15



[Here's another link](#) demonstrating how `const` isn't version safe, or relevant for reference types.

Summary:

- The value of your `const` property is set at compile time and can't change at runtime
- `Const` can't be marked as static - the keyword denotes they are static, unlike `readonly` fields which can.
- `Const` can't be anything except value (primitive) types
- The `readonly` keyword marks the field as unchangeable. However the property can be changed inside the constructor of the class
- The `readonly` only keyword can also be combined with static to make it act in the same way as a `const` (atleast on the surface). There is a marked difference when you look at the IL between the two
- `const` fields are marked as "literal" in IL while `readonly` is "initonly"

edited Dec 1 '14 at 16:33

answered Jan 31 '09 at 11:42



[Chris S](#)

51.6k 47 211 235



Read Only : Value can be changed through Ctor at runtime. But not through member Function

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

answered Jan 12 '16 at 17:42



Yeasin Abedin Siam

909 2 12 22

thanks for not making me read 4 paragraphs just for these two take-aways... – Don Cheadle Feb 26 '18 at 19:11

Yet another gotcha: readonly values can be changed by "devious" code via reflection.

8

```
var fi = this.GetType()
    .BaseType
    .GetField("_someField",
        BindingFlags.Instance | BindingFlags.NonPublic);
fi.SetValue(this, 1);
```

[Can I change a private readonly inherited field in C# using reflection?](#)

edited May 23 '17 at 11:54



Community ♦

1 1

answered Oct 13 '09 at 2:26



Greg

19.3k 10 52 74

I believe a `const` value is the same for all objects (and must be initialized with a literal expression), whereas `readonly` can be different for each instantiation...

6

answered Sep 11 '08 at 8:03



Daren Thomas

45k 37 137 185

One of the team members in our office provided the following guidance on when to use `const`, `static`, and `readonly`:

5

- Use **const** when you have a variable of a type you can know at runtime (string literal, int, double, enums,...) that you want all instances or consumers of a class to have access to where the value should not change.


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

- Use **static readonly** when you have a variable of a type that you cannot know at runtime (objects) that you want all instances or consumers of a class to have access to where the value should not change.
- Use **readonly** when you have an instance level variable you will know at the time of object creation that should not change.

One final note: a const field is static, but the inverse is not true.

answered Oct 2 '08 at 20:43

community wiki
[Scott Lawrence](#)

1 I think you mean "converse." The inverse would be "a non-const field is not static." Which may or may not be true. The converse, "a static field is (always) const" is not true. – [Michael Blackburn](#) Apr 27 '16 at 21:02 

They are both constant, but a const is available also at compile time. This means that one aspect of the difference is that you can use const variables as input to attribute constructors, but not readonly variables.

5

Example:

```
public static class Text {
    public const string ConstDescription = "This can be used.";
    public readonly static string ReadonlyDescription = "Cannot be used.";
}

public class Foo
{
    [Description(Text.ConstDescription)]
    public int BarThatBuilds {
        { get; set; }
    }

    [Description(Text.ReadonlyDescription)]
    public int BarThatDoesNotBuild {
        { get; set; }
    }
}
```

edited Jan 7 '16 at 23:44

answered Sep 17 '08 at 13:02

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

4

Variables marked `const` are little more than strongly typed `#define` macros, at compile time `const` variable references are replaced with inline literal values. As a consequence only certain built-in primitive value types can be used in this way. Variables marked `readonly` can be set, in a constructor, at run-time and their read-only-ness is enforced during run-time as well. There is some minor performance cost associated with this but it means you can use `readonly` with any type (even reference types).

Also, `const` variables are inherently static, whereas `readonly` variables can be instance specific if desired.

edited Jan 5 '09 at 22:34



Jason Baker

113k 112 340 494

answered Sep 11 '08 at 8:41



Wedge

17.4k 7 41 67

Added that consts are *strongly typed* `#define` macros. Otherwise, we may scare off all the C or C++ people. :-)) – Jason Baker Jan 5 '09 at 22:35

3

Another *gotcha*.

Since `const` really only works with basic data types, if you want to work with a class, you may feel "forced" to use `ReadOnly`. However, beware of the trap! `ReadOnly` means that you can not replace the object with another object (you can't make it refer to another object). But any process that has a reference to the object is free to modify the values *inside* the object!

So don't be confused into thinking that `ReadOnly` implies a user can't change things. There is no simple syntax in C# to prevent an instantiation of a class from having its internal values changed (as far as I know).

answered Sep 17 '08 at 13:28



Mark T

1,845 3 24 39

Yes that is more of a general theme. If you have a get only property exposing an arraylist, you can still modify the arraylist. You cannot set a different arraylist to that property, but you cannot stop the user from altering the arraylist. – Gishu Sep 17 '08 at 13:31

There is notable difference between `const` and `readonly` fields in C#.Net

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



const is by default static and needs to be initialized with constant value, which can not be modified later on. Change of value is not allowed in constructors, too. It can not be used with all datatypes. For ex- DateTime. It can not be used with DateTime datatype.

```
public const DateTime dt = DateTime.Today; //throws compilation error
public const string Name = string.Empty; //throws compilation error
public readonly string Name = string.Empty; //No error, Legal
```

readonly can be declared as static, but not necessary. No need to initialize at the time of declaration. Its value can be assigned or changed using constructor. So, it gives advantage when used as instance class member. Two different instantiation may have different value of readonly field. For ex -

```
class A
{
    public readonly int Id;

    public A(int i)
    {
        Id = i;
    }
}
```

Then readonly field can be initialised with instant specific values, as follows:

```
A objOne = new A(5);
A objTwo = new A(10);
```

Here, instance objOne will have value of readonly field as 5 and objTwo has 10. Which is not possible using const.

answered Dec 15 '14 at 13:01



Chirag

3,044 1 22 23



CONST

3

1. const keyword can be applied to fields or local variables

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

3. No Memory Allocated Because const value is embedded in IL code itself after compilation. It is like find all occurrences of const variable and replace by its value. So the IL code after compilation will have hard-coded values in place of const variables
4. Const in C# are by default static.
5. The value is constant for all objects
6. There is dll versioning issue - This means that whenever we change a public const variable or property , (In fact, it is not supposed to be changed theoretically), any other dll or assembly which uses this variable has to be re-built
7. Only C# built-in types can be declared as constant
8. Const field can not be passed as ref or out parameter

ReadOnly

1. readonly keyword applies only to fields not local variables
2. We can assign readonly field at the time of declaration or in constructor, not in any other methods.
3. dynamic memory allocated for readonly fields and we can get the value at run time.
4. Readonly belongs to the object created so accessed through only instance of class. To make it class member we need to add static keyword before readonly.
5. The value may be different depending upon constructor used (as it belongs to object of the class)
6. If you declare a non-primitive types (reference type) as readonly only reference is immutable not the object it contains.
7. Since the value is obtained at run time, there is no dll versioning problem with readonly fields/ properties.
8. We can pass readonly field as ref or out parameters in the constructor context.

edited Sep 8 '18 at 6:55

answered Sep 8 '18 at 6:39



Muhammad VP

47 6



A constant will be compiled into the consumer as a literal value while the static string will serve as a reference to the value defined.

2

As an exercise, try creating an external library and consume it in a console application, then alter the values in the library and recompile it (without recompiling the consumer program), drop the DLL into the directory and run the EXE manually, you should find that the constant string does not change.

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



Brett Ryan

15.4k 25 109 140

I sincerely doubt that is true... I will go check. – [ljs](#) Sep 8 '09 at 12:23

this is one of the 50 specific ways to improve your C# - [amazon.co.uk/Effective-Specific-Ways-Improve-Your/dp/0321245660/...](https://www.amazon.co.uk/Effective-Specific-Ways-Improve-Your/dp/0321245660/) – [Russ Cam](#) Sep 8 '09 at 12:24

2 @kronoz - This answer *is* correct. – [Andrew Hare](#) Sep 8 '09 at 12:26

[my.safaribooksonline.com/0321245660/...](https://my.safaribooksonline.com/0321245660/) – [Russ Cam](#) Sep 8 '09 at 12:27

@Andrew Hare - yes, I just checked. I am very surprised, that is a real gotcha, I'm really very surprised by that, amazed that is the case...! – [ljs](#) Sep 8 '09 at 12:29

A `const` has to be **hard-coded**, where as `readonly` can be **set in the constructor** of the class.

2

answered Feb 20 '11 at 12:29



Greg

15.8k 14 70 95

Constant

2

We need to provide the value to the `const` field when it is defined. The compiler then saves the constant's value in the assembly's metadata. This means that a constant can be defined only for the primitive type like boolean, char, byte and so on. Constants are always considered static members, not instance members.

Readonly

Readonly fields can only be resolved at runtime. That means we can define a value for a value using the constructor for the type in which the field is declared. The verification is done by the compiler that readonly fields are not written to by any method other than the constructor.

More about both [explained here in this article](#)

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



1



Principally; you can assign a value to a static readonly field to a non-constant value at runtime, whereas a const has to be assigned a constant value.

answered Sep 8 '09 at 12:24



ljs

26.5k

31

99

122



1



Const and readonly are similar, but they are not exactly the same. A const field is a compile-time constant, meaning that that value can be computed at compile-time. A readonly field enables additional scenarios in which some code must be run during construction of the type. After construction, a readonly field cannot be changed.

For instance, const members can be used to define members like:

```
struct Test
{
    public const double Pi = 3.14;
    public const int Zero = 0;
}
```

since values like 3.14 and 0 are compile-time constants. However, consider the case where you define a type and want to provide some pre-fab instances of it. E.g., you might want to define a Color class and provide "constants" for common colors like Black, White, etc. It isn't possible to do this with const members, as the right hand sides are not compile-time constants. One could do this with regular static members:

```
public class Color
{
    public static Color Black = new Color(0, 0, 0);
    public static Color White = new Color(255, 255, 255);
    public static Color Red = new Color(255, 0, 0);
    public static Color Green = new Color(0, 255, 0);
    public static Color Blue = new Color(0, 0, 255);
    private byte red, green, blue;

    public Color(byte r, byte g, byte b) {
```

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

```
    }
}
```

but then there is nothing to keep a client of `Color` from mucking with it, perhaps by swapping the `Black` and `White` values. Needless to say, this would cause consternation for other clients of the `Color` class. The "readonly" feature addresses this scenario. By simply introducing the `readonly` keyword in the declarations, we preserve the flexible initialization while preventing client code from mucking around.

```
public class Color
{
    public static readonly Color Black = new Color(0, 0, 0);
    public static readonly Color White = new Color(255, 255, 255);
    public static readonly Color Red = new Color(255, 0, 0);
    public static readonly Color Green = new Color(0, 255, 0);
    public static readonly Color Blue = new Color(0, 0, 255);
    private byte red, green, blue;

    public Color(byte r, byte g, byte b) {
        red = r;
        green = g;
        blue = b;
    }
}
```

It is interesting to note that `const` members are always static, whereas a `readonly` member can be either static or not, just like a regular field.

It is possible to use a single keyword for these two purposes, but this leads to either versioning problems or performance problems. Assume for a moment that we used a single keyword for this (`const`) and a developer wrote:

```
public class A
{
    public static const C = 0;
}
```

and a different developer wrote code that relied on `A`:

```
public class B
{
    // ...
}
```

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

```
}  
}
```

Now, can the code that is generated rely on the fact that A.C is a compile-time constant? I.e., can the use of A.C simply be replaced by the value 0? If you say "yes" to this, then that means that the developer of A cannot change the way that A.C is initialized -- this ties the hands of the developer of A without permission. If you say "no" to this question then an important optimization is missed. Perhaps the author of A is positive that A.C will always be zero. The use of both const and readonly allows the developer of A to specify the intent. This makes for better versioning behavior and also better performance.

answered Sep 28 '13 at 8:19

**Ramesh Rajendran**

21k 22 99 169

ReadOnly :The value will be initialized only once from the constructor of the class.
const: can be initialized in any function but only once

1

answered Jan 24 '14 at 7:34

**donstack**

1,125 3 14 37

The difference is that the value of a static readonly field is set at run time, so it can have a different value for different executions of the program. However, the value of a const field is set to a compile time constant.

1

Remember: For reference types, in both cases (static and instance), the readonly modifier only prevents you from assigning a new reference to the field. It specifically does not make immutable the object pointed to by the reference.

For details, please refer to C# Frequently Asked Questions on this topic:

<http://blogs.msdn.com/csharpfaq/archive/2004/12/03/274791.aspx>

answered Apr 5 '14 at 14:39

**Yonatan Nir**

4,017 16 60 118

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

1

initialized only from the Static constructor of the class. Read only is used only when we want to assign the value at run time.

answered Apr 9 '15 at 21:09



Omar AMEZOUG

439 6 16

1

Const: Absolute constant value during the application life time.

Readonly: It can be changed in running time.

answered Apr 29 at 20:26



Bigeyes

619 1 9 24

1

- when to use `const` or `readonly`

- `const`

- **compile-time** constant: **absolute** constant, value is set during declaration, is in the IL code itself

- `readonly`

- **run-time** constant: can be set in the constructor/init via config file i.e. `App.config` , but once it initializes it can't be changed

answered Jun 4 at 20:09



Ryan Efendy

1,117 12 10

0

One thing to add to what people have said above. If you have an assembly containing a `readonly` value (e.g. `readonly MaxFooCount = 4;`), you can change the value that calling assemblies see by shipping a new version of that assembly with a different value (e.g. `readonly MaxFooCount = 5;`)

But with a `const`, it would be folded into the caller's code when the caller was compiled.

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

answered Sep 15 '08 at 10:14



Anthony

3,716 6 57 81

1

2

next

protected by [Community](#) ♦ Jun 19 '14 at 3:17

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?

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