**Constant Readonly Static: C#**

**Const:**  
You use the const keyword to declare a constant field or a constant local. Constant fields and locals aren't variables and may not be modified. Constants can be numbers, Boolean values, strings, or a null reference. Don’t create a constant to represent information that you expect to change at any time. For example, don’t use a constant field to store the price of a service, a product version number, or the brand name of a company. These values can change over time, and because compilers propagate constants, other code compiled with your libraries will have to be recompiled to see the changes. See also the [readonly](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/readonly) keyword. For example:

C#

const int x = 0;

public const double gravitationalConstant = 6.673e-11;

private const string productName = "Visual C#";

public const double Pi = 3.14159;

public const int SpeedOfLight = 300000; // km per sec.

Constant fields are defined at the time of declaration in the code snippet, because once they are defined they can't be modified. By default a constant is static, so you can't define them static from your side.

**Notes:**

* Compile-time constant
* Can't be declared static
* Can't be modified or changed
* Can be of any type of Access Modifier
* Local scope
* Needs to get initialized
* Declared at the time of declaration

**Readonly:**

The readonly keyword is a modifier that you can use on fields. When a field declaration includes a readonly modifier, assignments to the fields introduced by the declaration can only occur as part of the declaration or in a constructor in the same class.

**Example**

In this example, the value of the field year cannot be changed in the method ChangeYear, even though it is assigned a value in the class constructor:

C#

class Age

{

readonly int \_year;

Age(int year)

{

\_year = year;

}

void ChangeYear()

{

//\_year = 1967; // Compile error if uncommented.

}

}

A Readonly field can be initialized either at the time of declaration or within the constructor of the same class. We can also change the value of a Readonly at runtime or assign a value to it at runtime (but in a non-static constructor only).

**Notes:**

* Run-time constant
* It can be static
* Global scope
* Can be declared in the constructer class
* Generally public

**Static:**

Use the static modifier to declare a static member, which belongs to the type itself rather than to a specific object. The static modifier can be used with classes, fields, methods, properties, operators, events, and constructors, but it cannot be used with indexers, finalizers, or types other than classes.

static class CompanyEmployee

{

public static void DoSomething() { /\*...\*/ }

public static void DoSomethingElse() { /\*...\*/ }

}

A [static](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/static) class is basically the same as a non-static class, but there is one difference: a static class cannot be instantiated. In other words, you cannot use the [new](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/new) keyword to create a variable of the class type. Because there is no instance variable, you access the members of a static class by using the class name itself. For example, if you have a static class that is named UtilityClass that has a public method named MethodA, you call the method as shown in the following example:

C#

UtilityClass.MethodA();

The static keyword is used to declare a static member. If we are declare a class as a static class then in this case all the class members must be static too. The static keyword can be used effectively with classes, fields, operators, events, methods and so on effectively.

**Notes:**

* Can't be used with indexers
* Works with constructors too
* By default it is private
* Can be parameterized or public too
* If its applied to a class then all the class members need to be static

Constants

* 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

* Are evaluated when code execution hits class reference (when new instance is created or a static method is executed)
* 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)

**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. Constants are static by default
* 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"

**Notes:**

* 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.
* Use **static** when you have data that you want all instances or consumers of a class to have access to where the value can change.
* 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.