What's the difference between the 'ref' and 'out' keywords?

Ask Question

809

236

I'm creating a function where I need to pass an object so that it can be modified by the function. What is the difference between:

```
public void myFunction(ref MyClass someClass)
```

and

```
public void myFunction(out MyClass someClass)
```

Which should I use and why?

c# reference keyword out ref

edited Nov 8 '12 at 2:02

Mechanical snail

19.9k 10 74 100

asked Dec 23 '08 at 9:16

TK.

19.3k 45 107 141

55 You: ***I need to pass an object so that it can be modified*** It looks like `MyClass` would be a `class` type, i.e. a reference type. In that case, the object you pass can be modified by the `myFunction` even with no `ref` / `out` keyword. `myFunction` will receive a *new* reference that points to the *same* object, and it can modify that same object as much as it wants. The difference the `ref` keyword would make, would be that `myFunction` received the *same* reference to the same object. That would be important only if `myFunction` were to change the reference to point to *another* object. – Jeppe Stig Nielsen Jun 8 '13 at 13:51

2 I'm puzzled by the amount of confusing answers here, when @AnthonyKolesov's is quite perfect. – o0'. Dec 24 '13 at 13:05

Declaring an out method is useful when you want a method to return multiple values. One argument can be assigned to null. This enables methods to return values optionally. – Yevgraf Andreyevich Zhivago Aug 19 '14 at 2:50

Here explained with Example It more understandable :) dotnet-tricks.com/Tutorial/csharp/… – Prageeth godage Mar 8 '16 at 10:46

1 @JeppeStigNielsen's comment is, technically, the (only) correct answer to the OP's actual question. To pass an object into a method so that **the method can modify the object**, simply pass the (reference to the) object into the method by value. Changing the object within the method via the object argument **modifies the original object**, even though the method contains its own separate variable (which references the same object). – David R Tribble Jun 29 '18 at 21:32

23 Answers

`ref` tells the compiler that the object is initialized before entering the function, while `out` tells the compiler that the object will be initialized inside the function.

1066 So while `ref` is two-ways, `out` is out-only.

edited Dec 21 '12 at 13:54
miguel

73 1 7

answered Dec 23 '08 at 9:18

Rune Grimstad

30k 8 51 71

246 Another cool thing specific to out is that the function has to assign to the out parameter. It's not allowed to leave it unassigned. – Daniel Earwicker Dec 23 '08 at 11:30

6 is 'ref' only applicable to value type? Since reference type is always pass by ref. – faulty Dec 24 '08 at 9:42

3 Yes. Value types including structs – Rune Grimstad Dec 31 '08 at 9:54

15 @faulty: No, ref is not only applicable to value types. ref/out are like pointers in C/C++, they deal with the memory location of the object (indirectly in C#) instead of the direct object. – thr Mar 15 '10 at 6:53

47 @faulty: Counterintuitively, Reference types are always passed by value in C#, unless you use the ref specifier. If you set myval=somenewval, the effect is only in that function scope. The ref keyword would allow you to change myval to point to somenewval. – JasonTrue Apr 14 '10 at 23:21

5

ref means that the value in the ref parameter is already set, the method can read and modify it. Using the ref keyword is the same as saying that the caller is responsible for initializing the value of the parameter.

out tells the compiler that the initialization of object is the responsibility of the function, the function has to assign to the out

parameter. It's not allowed to leave it unassigned.

edited Oct 31 '18 at 9:25

itsmysterybox
1,141 3 9 20

answered Dec 6 '14 at 19:21

Farhan S.
493 6 21

3

Authoring Time:

(1) We create the calling method `Main()`

(2) it creates a List object (which is a reference type object) and stores it in the variable `myList` .

```
public sealed class Program
{
    public static Main()
    {
        List<int> myList = new List<int>();
```

During Runtime:

(3) Runtime allocates a memory on stack at #00, wide enough to store an address (#00 = `myList` , since variable names are really just aliases for memory locations)

(4) Runtime creates a list object on heap at memory location #FF(all these addresses are for example sakes)

(5) Runtime would then store the starting address #FF of the object at #00(or in words, stores the reference of the List object in the pointer `myList`)

Back to Authoring Time:

(6) We then pass the List object as argument `myParamList` to the called method `modifyMyList` and assign a new List object to it

```
List<int> myList = new List<int>();

List<int> newList = ModifyMyList(myList)

public List<int> ModifyMyList(List<int> myParamList){
     myParamList = new List<int>();
     return myParamList;
}
```

During Runtime:

(7) Runtime starts the call routine for the called method and as part of it, checks the type of parameters.

(8) Upon finding the reference type, it allocates a memory on stack at #04 for aliasing the parameter variable `myParamList` .

(9) It then stores the value #FF in it as well.

(10) Runtime creates a list object on heap at memory location #004 and replaces #FF in #04 with this value(or dereferenced the original List object and pointed to the new List object in this method)

The address in #00 is not altered and retains the reference to #FF(or the original `myList` pointer is not disturbed).

The ***ref*** keyword is a compiler directive to skip the generation of runtime code for (8) and (9) which means there will be no heap allocation for method parameters. It will use the original #00 pointer to operate on the object at #FF. If the original pointer is not initialised, the runtime will halt complaining it can’t proceed since the variable isn’t initialised

The ***out*** keyword is a compiler directive which pretty much is the same as ref with a slight modification at (9) and (10). The compiler expects the argument to be uninitialised and will continue with (8),

(4) and (5) to create an object on heap and to stores its starting address in the argument variable. No uninitialised error will be thrown and any previous reference stored will be lost.

edited May 30 '17 at 10:37

answered May 29 '17 at 16:10

supi
513 4 8

0

If you want to pass your parameter as a ref then you should initialize it before passing parameter to the function else compiler itself will show the error.But in case of out parameter you don't need to initialize the object parameter before passing it to the method.You can initialize the object in the calling method itself.

answered Dec 5 '16 at 9:57

Rakeshkumar Das
85 1 4

16

out:

In C#, a method can return only one value. If you like to return more than one value, you can use the out keyword. The out modifier return as return-by-reference. The simplest answer is that the keyword “out” is used to get the value from the method.

1. You don't need to initialize the value in the calling function.
2. You must assign the value in the called function, otherwise the compiler will report an error.

ref:

In C#, when you pass a value type such as int, float, double etc. as an argument to the method parameter, it is passed by value. Therefore, if you modify the parameter value, it does not affect argument in the method call. But if you mark the parameter with “ref” keyword, it will reflect in the actual variable.

1. You need to initialize the variable before you call the function.
2. It’s not mandatory to assign any value to the ref parameter in the method. If you don’t change the value, what is the need to mark it as “ref”?

answered Apr 1 '16 at 18:31

Nazmul Hasan
5,513 4 35 52

"In C#, a method can return only one value. If you like to return more than one value, you can use the out keyword." We can also use "ref" for returning value. So we can use both ref and out if we want to return multiple values from a method? – Ned May 23 '18 at 20:23

1 In the c# 7 you can return multiple values with ValueTuples. – Iman Bahrampour Dec 9 '18 at 5:25

7

For those that learn by example (like me) here's what Anthony Kolesov is saying.

I've created some minimal examples of ref, out, and others to illustrate the point. I'm not covering best practices, just examples to understand the differences.

https://gist.github.com/2upmedia/6d98a57b68d849ee7091

edited May 23 '17 at 12:18

Community ♦

1 1

answered Dec 25 '15 at 21:16

2upmedia

1,837 1 14 12

1

Ref: The ref keyword is used to pass an argument as a reference. This means that when value of that parameter is changed in the method, it gets reflected in the calling method. An argument that is passed using a ref keyword must be initialized in the calling method before it is passed to the called method.

Out: The out keyword is also used to pass an argument like ref keyword, but the argument can be passed without assigning any value to it. An argument that is passed using an out keyword must be initialized in the called method before it returns back to calling method.

```
public class Example
{
 public static void Main()
 {
 int val1 = 0; //must be initialized
 int val2; //optional

 Example1(ref val1);
 Console.WriteLine(val1);

 Example2(out val2);
 Console.WriteLine(val2);
 }

 static void Example1(ref int value)
 {
 value = 1;
 }
 static void Example2(out int value)
 {
 value = 2;
 }
```

```
}

/* Output     1     2
```

Ref and out in method overloading

Both ref and out cannot be used in method overloading simultaneously. However, ref and out are treated differently at run-time but they are treated same at compile time (CLR doesn't differentiates between the two while it created IL for ref and out).

answered Jul 26 '15 at 23:22

Dejan Ciev

124 4

-3

Mind well that the reference parameter which is passed inside the function is directly worked on.

For example,

```
public class MyClass
{
    public string Name { get; set; }
}

public void Foo()
{
    MyClass myObject = new MyClass();
    myObject.Name = "Dog";
    Bar(myObject);
    Console.WriteLine(myObject.Name); // Writes "Dog".
}

public void Bar(MyClass someObject)
{
    MyClass myTempObject = new MyClass();
    myTempObject.Name = "Cat";
```

```
        someObject = myTempObject;
    }
```

This will write Dog, not Cat. Hence you should directly work on someObject.

edited Jul 26 '15 at 20:30

Peter Mortensen
13.9k 19 87 113

answered Apr 14 '10 at 23:15

Mangesh Pimpalkar
1

6 While everything here is pretty much true it doesn't really explain the difference between by value by reference or out. At best it half explains the difference between reference and value/immutable types. – Conrad Frix Dec 25 '12 at 4:10

If you want that code to write cat please pass that object along with the ' ref ' key like this: public static void Bar(ref MyClass someObject), Bar(ref myObject); – Daniel Botero Correa Feb 4 at 6:11

138

Let's say Dom shows up at Peter's cubicle about the memo about the TPS reports.

If Dom were a ref argument, he would have a printed copy of the memo.

If Dom were an out argument, he'd make Peter print a new copy of the memo for him to take with him.

edited Jun 29 '15 at 14:48

answered Mar 23 '10 at 19:33

Michael Blackburn
2,609 1 18 18

51 ref Dom would have written the report in pencil so that Peter could modify it – Deebster Aug 5 '11 at 9:25

5 @Deebster you know, that metaphor never did anything to you, why must you torture it so? ;) – Michael Blackburn Aug 8 '11 at 16:09

18 entertaining yet educating, stackoverflow needs more posts like this – Frank Visaggio Jun 26 '12 at 16:54

1 Just in case somebody finds this answer only half-funny, please watch the movie "Office Space". – displayName Jun 13 '17 at 19:33

and Dom and Peters boss would stand behin Dom (as out argument), forcing both to work on printing it out anew until Peter hands Domd the printout – Patrick Artner Dec 21 '17 at 7:05

4

ref and out work just like passing by references and passing by pointers as in C++.

For ref, the argument must declared and initialized.

For out, the argument must declared but may or may not be initialized

```
        double nbr = 6; // if not initialized we get error
        double dd = doit.square(ref nbr);

        double Half_nbr ; // fine as passed by out, but inside the c
initialize it
        doit.math_routines(nbr, out Half_nbr);
```

edited Mar 15 '15 at 10:36

answered Mar 15 '15 at 8:49

RotatingWheel
478 5 18

1 You can declare a variable inline: `out double Half_nbr .` – Sebastian Hofmann Feb 8 '18 at 8:15

477

The `ref` modifier means that:

1. The value is already set and
2. The method can read and modify it.

The `out` modifier means that:

1. The Value isn't set and can't be read by the method *until* it is set.
2. The method *must* set it before returning.

edited Feb 21 '15 at 0:19

Agi Hammerthief
1,279 16 27

answered Dec 23 '08 at 12:40

Anton Kolesov
5,394 1 13 5

27 This answer most clearly and concisely explains the restrictions that the compiler imposes when using the out keyword as opposed to the ref keyword. – Dr. Wily's Apprentice Sep 2 '10 at 15:52

5 From MSDN: A ref parameter must be initialized before use, while an out parameter does not have to be explicitly initialized before being passed and any previous value is ignored. – Shiva Kumar Aug 20 '13 at 8:33

With `out` , can it be read at all within the method, before it's been set by that method, if it has been initialized before the method was called? I

mean, can the called method read what the calling method passed to it as an argument? – Panzercrisis Mar 25 '16 at 13:46

2 Panzercrisis, for "out", the called method can read if it is already set. but it must set it again. – robert jebakumar2 Apr 30 '17 at 9:06

0

From the standpoint of a method which receives a parameter, the difference between `ref` and `out` is that C# requires that methods must write to every `out` parameter before returning, and must not do anything with such a parameter, other than passing it as an `out` parameter or writing to it, until it has been either passed as an `out` parameter to another method or written directly. Note that some other languages do not impose such requirements; a virtual or interface method which is declared in C# with an `out` parameter may be overridden in another language which does not impose any special restrictions on such parameters.

From the standpoint of the caller, C# will in many circumstances assume when calling a method with an `out` parameter will cause the passed variable to be written without having been read first. This assumption may not be correct when calling methods written in other languages. For example:

```
struct MyStruct
{
   ...
   myStruct(IDictionary<int, MyStruct> d)
   {
     d.TryGetValue(23, out this);
   }
}
```

If `myDictionary` identifies an `IDictionary<TKey,TValue>` implementation written in a language other than C#, even though `MyStruct s = new MyStruct(myDictionary);` looks like an assignment, it could potentially leave `s` unmodified.

Note that constructors written in VB.NET, unlike those in C#, make no assumptions about whether called methods will modify any `out` parameters, and clear out all fields unconditionally. The odd behavior alluded to above won't occur with code written entirely in VB or entirely in C#, but can occur when code written in C# calls a method written in VB.NET.

answered Feb 11 '15 at 22:51

supercat
58k 3 117 156

1

```
public static void Main(string[] args)
   {
       //int a=10;
       //change(ref a);
       //Console.WriteLine(a);
       // Console.Read();

       int b;
       change2(out b);
       Console.WriteLine(b);
       Console.Read();
   }
   // static void change(ref int a)
   //{
   //    a = 20;
   //}

    static void change2(out int b)
    {
        b = 20;
    }
```

you can check this code it will describe you its complete differnce when you use "ref" its mean that u already initialize that int/string

but when you use "out" it works in both conditions wheather u initialize that int/string or not but u must initialize that int/string in that

function

edited Nov 13 '14 at 11:01

answered Nov 13 '14 at 9:17

Haris Zia

57 8

4

Out: A return statement can be used for returning only one value from a function. However, using output parameters, you can return two values from a function. Output parameters are like reference parameters, except that they transfer data out of the method rather than into it.

The following example illustrates this:

```
using System;

namespace CalculatorApplication
{
   class NumberManipulator
   {
      public void getValue(out int x )
      {
         int temp = 5;
         x = temp;
      }

      static void Main(string[] args)
      {
         NumberManipulator n = new NumberManipulator();
         /* local variable definition */
         int a = 100;

         Console.WriteLine("Before method call, value of a : {0}", a
```

```
            /* calling a function to get the value */
            n.getValue(out a);

            Console.WriteLine("After method call, value of a : {0}", a)
            Console.ReadLine();

        }
    }
}
```

ref: A reference parameter is a reference to a memory location of a variable. When you pass parameters by reference, unlike value parameters, a new storage location is not created for these parameters. The reference parameters represent the same memory location as the actual parameters that are supplied to the method.

In C#, you declare the reference parameters using the ref keyword. The following example demonstrates this:

```
using System;
namespace CalculatorApplication
{
   class NumberManipulator
   {
      public void swap(ref int x, ref int y)
      {
         int temp;

         temp = x; /* save the value of x */
         x = y;   /* put y into x */
         y = temp; /* put temp into y */
       }

      static void Main(string[] args)
      {
         NumberManipulator n = new NumberManipulator();
         /* local variable definition */
         int a = 100;
         int b = 200;

         Console.WriteLine("Before swap, value of a : {0}", a);
         Console.WriteLine("Before swap, value of b : {0}", b);

         /* calling a function to swap the values */
```

```
            n.swap(ref a, ref b);

            Console.WriteLine("After swap, value of a : {0}", a);
            Console.WriteLine("After swap, value of b : {0}", b);

            Console.ReadLine();

        }
    }
}
```

answered Oct 25 '14 at 16:57

Faisal Naseer
1,833 16 35

27

[ref is in **and** out](#).

You should use `out` in preference wherever it suffices for your requirements.

edited Aug 13 '14 at 11:50

answered Dec 23 '08 at 9:17

Ruben Bartelink
44.4k 17 144 204

not quite, as the accepted answer ref if directional and useless ignoring value-types if not passed back out. – kenny Mar 27 '10 at 13:17

@kenny: Can you clarify a bit please - i.e., which words would you change to maintain the spirit of the answer but remove the inacuracy you percieve? My answer is not a crazy guess from a newbie, but the haste (terseness, typos) in your comment seems to assume it is. The aim is to provide a way of thinking about the difference with the least number of words. – Ruben Bartelink Mar 27 '10 at 13:46

(BTW I'm familiar with value types, reference types, passing by reference, passing by value, COM and C++ should you find it useful to make reference to those concepts in your clarification) – Ruben Bartelink Mar 27 '10 at 14:17

1 Object references are passed by value (except when using the "ref" or "out" keyword). Think of objects as ID numbers. If a class variable holds "Object #1943" and one passes that variable by value to a routine, that routine can make changes to Object #1943, but it can't make the variable point to anything other than "Object #1943". If the variable was passed by reference, the routine could make the variable point hold "Object #5441". – supercat Jan 15 '12 at 18:40

1 @supercat: I do like your explanation of ref vs val (and this followup anaology). I think kenny doesnt actually need any of this explained to him, (relatively) confusing as his comments were. I do wish we could all just remove these goddam comments though as they're just confusing everyone. The root cause of all this nonsense appears to be that kenny misread my answer and has yet to point out a single word that should be added/removed/replaced. None of the three of us have learned anything from the discussion we didnt already know and the other answer has a ludicrous number of upvotes. – Ruben Bartelink Jan 15 '12 at 23:20

8

`ref` and `out` behave similarly except following differences.

- `ref` variable must be initialized before use. `out` variable can be used without assignment
- `out` parameter must be treated as an unassigned value by the function that uses it. So, we can use initialized `out` parameter in the calling code, but the value will be lost when the function executes.

answered Feb 10 '14 at 16:29

gmail user

1,696 4 26 38

Below I have shown an example using both **Ref** and **out**. Now, you

0

all will be cleared about ref and out.

In below mentioned example when i comment **//myRefObj = new myClass { Name = "ref outside called!! " };** line, will get an error saying **"Use of unassigned local variable 'myRefObj'"**, but there is no such error in **out**.

Where to use Ref: when we are calling a procedure with an in parameter and the same parameter will be used to store the output of that proc.

Where to use Out: when we are calling a procedure with no in parameter and teh same param will be used to return the value from that proc. Also note the output

```
public partial class refAndOutUse : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        myClass myRefObj;
        myRefObj = new myClass { Name = "ref outside called!!  <br/>
        myRefFunction(ref myRefObj);
        Response.Write(myRefObj.Name); //ref inside function

        myClass myOutObj;
        myOutFunction(out myOutObj);
        Response.Write(myOutObj.Name); //out inside function
    }

    void myRefFunction(ref myClass refObj)
    {
        refObj.Name = "ref inside function <br/>";
        Response.Write(refObj.Name); //ref inside function
    }
    void myOutFunction(out myClass outObj)
    {
        outObj = new myClass { Name = "out inside function <br/>" };
        Response.Write(outObj.Name); //out inside function
    }
}

public class myClass
{
```

```
    public string Name { get; set; }
}
```

edited Oct 29 '13 at 15:45

FCR
170 1 14

answered Jun 25 '13 at 11:18

Ankur Bhutani
1,980 2 21 22

1

They're pretty much the same - the only difference is that a variable you pass as an out parameter doesn't need to be initialised, and the method using the ref parameter has to set it to something.

```
int x;    Foo(out x); // OK
int y;    Foo(ref y); // Error
```

Ref parameters are for data that might be modified, out parameters are for data that's an additional output for the function (eg int.TryParse) that are already using the return value for something.

answered Jun 20 '13 at 16:06

Talha Khan
29 1

12

Extending the Dog, Cat example. The second method with ref changes the object referenced by the caller. Hence "Cat" !!!

```
    public static void Foo()
    {
```

```
    MyClass myObject = new MyClass();
    myObject.Name = "Dog";
    Bar(myObject);
    Console.WriteLine(myObject.Name); // Writes "Dog".
    Bar(ref myObject);
    Console.WriteLine(myObject.Name); // Writes "Cat".
}

public static void Bar(MyClass someObject)
{
    MyClass myTempObject = new MyClass();
    myTempObject.Name = "Cat";
    someObject = myTempObject;
}

public static void Bar(ref MyClass someObject)
{
    MyClass myTempObject = new MyClass();
    myTempObject.Name = "Cat";
    someObject = myTempObject;
}
```

edited May 13 '13 at 16:01

Richard Bos
410 3 15

answered May 19 '10 at 13:35

BBB
121 1 2

-4

I may not be so good at this, but surely strings (even though they are technically reference types and live on the heap) are passed by value, not reference?

```
string a = "Hello";

string b = "goodbye";

b = a; //attempt to make b point to a, won't work.
```

```
a = "testing";

Console.WriteLine(b); //this will produce "hello", NOT "test
```

This why you need ref if you want changes to exist outside of the scope of the function making them, you aren't passing a reference otherwise.

As far as I am aware you only need ref for structs/value types and string itself, as string is a reference type that pretends it is but is not a value type.

I could be completely wrong here though, I am new.

answered Dec 8 '11 at 18:36

Edwin
13 1

5 Welcome to Stack Overflow, Edwin. Strings are passed by reference, just like any other object, as far as I know. You may be confused because strings are immutable objects, so it's not as obvious that they are passed by reference. Imagine that string had a method called `Capitalize()` that would change the contents of the string to capital letters. If you then replaced your line `a = "testing";` with `a.Capitalize();` , then your output would be "HELLO", not "Hello". One of the advantages of immutable types is that you can pass around references and not worry about other code changing the value. – Don Kirkby Dec 8 '11 at 19:42

2 There are three fundamental types of semantics a type can expose: mutable reference semantics, mutable value semantics, and immutable semantics. Consider variables x and y of a type T, which has field or property m, and assume x is copied to y. If T has reference semantics, changes to x.m will be observed by y.m. If T has value semantics, one can change x.m without affecting y.m. If T has immutable semantics, neither x.m nor y.m will ever change. Immutable semantics can be simulated by either reference or value objects. Strings are immutable reference objects. – supercat Jan 15 '12 at 18:37

51

I am going to try my hand at an explanation:

I think we understand how the value types work right? Value types are (int, long, struct etc.). When you send them in to a function without a ref command it COPIES the **data**. Anything you do to that data in the function only affects the copy, not the original. The ref command sends the ACTUAL data and any changes will affect the data outside the function.

Ok on to the confusing part, reference types:

Lets create a reference type:

```
List<string> someobject = new List<string>()
```

When you new up *someobject*, two parts are created:

1. The block of memory that holds data for *someobject*.
2. A reference (pointer) to that block of data.

Now when you send in *someobject* into a method without ref it COPIES the **reference** pointer, NOT the data. So you now have this:

```
(outside method) reference1 => someobject
(inside method)  reference2 => someobject
```

Two references pointing to the same object. If you modify a property on *someobject* using reference2 it will affect the same data pointed to by reference1.

```
(inside method)  reference2.Add("SomeString");
(outside method) reference1[0] == "SomeString"   //this is true
```

If you null out reference2 or point it to new data it will not affect reference1 nor the data reference1 points to.

```
(inside method) reference2 = new List<string>();
(outside method) reference1 != null; reference1[0] == "SomeString" /,

The references are now pointing like this:
reference2 => new List<string>()
reference1 => someobject
```

Now what happens when you send *someobject* by ref to a method? The **actual reference** to *someobject* gets sent to the method. So you now have only one reference to the data:

```
(outside method) reference1 => someobject;
(inside method)  reference1 => someobject;
```

But what does this mean? It acts exactly the same as sending someobject not by ref except for two main thing:

1) When you null out the reference inside the method it will null the one outside the method.

```
(inside method)  reference1 = null;
(outside method) reference1 == null;  //true
```

2) You can now point the reference to a completely different data location and the reference outside the function will now point to the new data location.

```
(inside method)  reference1 = new List<string>();
(outside method) reference1.Count == 0; //this is true
```

answered Sep 23 '10 at 17:55

James Roland

7,593 4 24 16

You mean after all (in ref case) there's only one reference to data but two alias for it. Right? – Sadiq Feb 14 '15 at 5:22

3 Upvoted for the clear explanation. But I think this doesn't answer the question, as it do not explain the difference between `ref` and `out` parameters. – Joyce Babu Sep 1 '16 at 8:26

1 Amazing. can you explain same as for `out` keyword? – Asif Mushtaq Sep 27 '16 at 19:06

Great answer, been trying to understand how to pass a ref type like in cpp, but it seemed wierd to pass ref type with ref keyword :D But now I know I can actually do that legally – Phoera Jul 19 '18 at 9:38

6

"Baker"

That's because the first one changes your string-reference to point to "Baker". Changing the reference is possible because you passed it via the ref keyword (=> a reference to a reference to a string). The Second call gets a copy of the reference to the string.

string looks some kind of special at first. But string is just a reference class and if you define

```
string s = "Able";
```

then s is a reference to a string class that contains the text "Able"! Another assignment to the same variable via

```
s = "Baker";
```

does not change the original string but just creates a new instance and let s point to that instance!

You can try it with the following little code example:

```
string s = "Able";
string s2 = s;
```

```
s = "Baker";
Console.WriteLine(s2);
```

What do you expect? What you will get is still "Able" because you just set the reference in s to another instance while s2 points to the original instance.

EDIT: string is also immutable which means there is simply no method or property that modifies an existing string instance (you can try to find one in the docs but you won't fins any :-)). All string manipulation methods return a new string instance! (That's why you often get a better performance when using the StringBuilder class)

edited Dec 23 '08 at 13:45

answered Dec 23 '08 at 13:40

mmmmmmmm

11.6k 2 25 52

1 Exactly. So it's not strictly true to say "Since you're passing in a reference type (a class) there is no need use ref". – Paul Mitchell Dec 23 '08 at 13:43

In theory it is right to say so because he wrote "so that it can be modified" which isn't possible on strings. But because of immutable objects "ref" and "out" are very useful also for reference types! (.Net contains a lot of immutable classes!) – mmmmmmmm Dec 23 '08 at 13:49

Yes, you're right. I didn't think of immutable objects like strings because most object are mutable. – Albic Dec 23 '08 at 16:55

1 Well, this is a puzzling answer to see in LQP, to be sure; there's not a thing wrong with it except that it appears to be a long and thorough response to another comment (since the original question mentions Able and Baker in none of its revisions), as though this were a forum. I guess that wasn't really sorted out yet way back when. – Nathan Tuggy May 25 '15 at 1:35

8

Since you're passing in a reference type (a class) there is no need use `ref` because per default only a **reference** to the actual object is passed and therefore you always change the object behind the reference.

Example:

```
public void Foo()
{
    MyClass myObject = new MyClass();
    myObject.Name = "Dog";
    Bar(myObject);
    Console.WriteLine(myObject.Name); // Writes "Cat".
}

public void Bar(MyClass someObject)
{
    someObject.Name = "Cat";
}
```

As long you pass in a class you don't have to use `ref` if you want to change the object inside your method.

answered Dec 23 '08 at 10:57

Albic
2,834 2 16 23

5 This works only if no new object is created and returned. When a new object is created, the reference to the old object would be lost. – etsuba Dec 23 '08 at 15:41

7 This is wrong - try the following: add `someObject = null` to `Bar` end execute. Your code will run fine as only `Bar` 's reference to the instance was nulled. Now change `Bar` to `Bar(ref MyClass someObject)` and execute again - you'll get a `NullReferenceException` because `Foo` 's reference to the instance has been nulled too. – Keith Sep 27 '10 at 7:55

protected by Community ♦ Dec 7 '13 at 12:22

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?