Should 'using' directives be inside or outside the namespace?

Ask Question

1863

405

I have been running StyleCop over some C# code, and it keeps reporting that my `using` directives should be inside the namespace.

Is there a technical reason for putting the `using` directives inside instead of outside the namespace?

c# .net namespaces stylecop code-organization

edited Apr 13 '18 at 11:44
Philippe
3,214 3 31 52

asked Sep 24 '08 at 3:49
benPearce
19.8k 14 53 92

3 Sometimes it makes difference where you put usings: stackoverflow.com/questions/292535/linq-to-sql-designer-bug – gius Dec 12 '08 at 18:15

78 Just for reference, there are implications beyond just the question of multiple classes per file, so if you're new to this question, please keep reading. – Charlie Jan 22 '10 at 17:23

3 @user-12506 - this does not work very well in a medium to large

development team where some level of code consistency is required. And as noted previously, if you don't understand the different layouts you may find edge cases that don't work as you expect. – benPearce Feb 3 '14 at 22:28

19 Terminology: Those are not `using` *statements*; they are `using` ***directives***. A `using` statement, on the other hand, is a language structure that occurs along with other statements inside a method body etc. As an example, `using (var e = s.GetEnumerator()) { /* ... */ }` is a statement that is loosely the same as `var e = s.GetEnumerator(); try { /* ... */ } finally { if (e != null) { e.Dispose(); } }`. – Jeppe Stig Nielsen Apr 11 '17 at 21:30

If this was not mentioned already by anyone, actually Microsoft too recommends putting `using` statements inside the `namespace` declarations, in their internal coding guidlines – user1451111 Jul 27 '18 at 2:15

10 Answers

1948 ✓

There is actually a (subtle) difference between the two. Imagine you have the following code in File1.cs:

```
// File1.cs
using System;
namespace Outer.Inner
{
    class Foo
    {
        static void Bar()
        {
            double d = Math.PI;
        }
    }
}
```

Now imagine that someone adds another file (File2.cs) to the project that looks like this:

Home

PUBLIC

Stack Overflow

Tags

Users

```
// File2.cs
namespace Outer
{
    class Math
    {
    }
}
```

The compiler searches `Outer` before looking at those `using` directives outside the namespace, so it finds `Outer.Math` instead of `System.Math`. Unfortunately (or perhaps fortunately?), `Outer.Math` has no `PI` member, so File1 is now broken.

This changes if you put the `using` inside your namespace declaration, as follows:

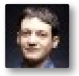
```
// File1b.cs
namespace Outer.Inner
{
    using System;
    class Foo
    {
        static void Bar()
        {
            double d = Math.PI;
        }
    }
}
```

Now the compiler searches `System` before searching `Outer`, finds `System.Math`, and all is well.

Some would argue that `Math` might be a bad name for a user-defined class, since there's already one in `System`; the point here is just that there *is* a difference, and it affects the maintainability of your code.

It's also interesting to note what happens if `Foo` is in namespace `Outer`, rather than `Outer.Inner`. In that case, adding `Outer.Math` in File2 breaks File1 regardless of where the `using` goes. This implies that the compiler searches the innermost enclosing namespace before it looks at any `using` directive.

edited Apr 13 '18 at 11:46


Philippe
3,214 3 31 52

answered Sep 30 '08 at 2:33

Charlie
34.4k 4 36 66

25 This is imho a much better reason to put using statements locally than Mark's multiple-namespaces-in-one-file argument. Especially sine the compile can and will complain about the naming clash (see the StyleCop documentation for this rule (e.g. as posted by Jared)). – David Schmitt Dec 30 '09 at 11:27

114 The accepted answer is good, but to me it seems like a good reason to put the using clauses *outside* the namespace. If I'm in namespace Outer.Inner I would expect it to use the Math class from Outer.Inner and not System.Math. – Frank Wallis Dec 9 '11 at 16:29

7 I concur with this as well. The accepted answer is correct in that it technically describes the difference. However, one or the other class will need an explicit callout. I would very much ratehr have "Math" resolve to my own local class, and "System.Math" refer to the external class - even if System.Math was being used as "Math" before Outer.Math existed. Yes, it's more work to fix however many pre-existing references, but that could also be a hint that maybe Outer.Math should have a different name! – mbmcavoy Apr 4 '12 at 15:47

13 Great answer, but it seems to me that I'd only want to put non-framework using statements locally, and keep the framework using statements global. Anyone have further explanation why I should completely changed my preference? Also where did this come from, the templates in VS2008 put using outside the namespace? – Thymine May 4 '12 at 20:47

28 I think this is more of a bad naming convention rather than changing the place of your using. There shouldn't be a class called Math in your solution – jDeveloper Sep 6 '12 at 17:03

1

Another subtlety that I don't believe has been covered by the other answers is for when you have a class and namespace with the same name.

When you have the import inside the namespace then it will find the class. If the import is outside the namespace then the import will be ignored and the class and namespace have to be fully qualified.

```
//file1.cs
namespace Foo
{
    class Foo
    {
    }
}

//file2.cs
namespace ConsoleApp3
{
    using Foo;
    class Program
    {
        static void Main(string[] args)
        {
            //This will allow you to use the class
            Foo test = new Foo();
        }
    }
}

//file2.cs
using Foo; //Unused and redundant
namespace Bar
{
    class Bar
    {
        Bar()
        {
            Foo.Foo test = new Foo.Foo();
            Foo test = new Foo(); //will give you an error that a na
used like a class.
        }
    }
}
```

edited Sep 3 '18 at 8:13

answered Aug 24 '18 at 15:28

Ben Gardner
16 6

0

The technical reasons are discussed in the answers and I think that it comes to the personal preferences in the end since the difference is not that *big* and there are tradeoffs for both of them. Visual Studio's default template for creating `.cs` files use `using` directives outside of namespaces e.g.

One can adjust stylecop to check `using` directives outside of namespaces through adding `stylecop.json` file in the root of the project file with the following:

```
{
  "$schema":
"https://raw.githubusercontent.com/DotNetAnalyzers/StyleCopAnalyzers,

    "orderingRules": {
      "usingDirectivesPlacement": "outsideNamespace"
    }
  }
}
```

You can create this config file in solution level and add it to your projects as 'Existing Link File' to share the config across all of your projects too.

answered Jun 3 '18 at 12:38
sotn
1,809 16 21

3

As Jeppe Stig Nielsen said, this thread already has great answers, but I thought this rather obvious subtlety was worth mentioning too.

`using` directives specified inside namespaces can make for shorter code since they don't need to be fully qualified as when they're specified on the outside.

The following example works because the types `Foo` and `Bar` are both in the same global namespace, `Outer`.

Presume the code file *Foo.cs*:

```
namespace Outer.Inner
{
    class Foo { }
}
```

And *Bar.cs*:

```
namespace Outer
{
    using Outer.Inner;

    class Bar
    {
        public Foo foo;
    }
}
```

That may omit the outer namespace in the `using` directive, for short:

```
namespace Outer
{
    using Inner;

    class Bar
    {
        public Foo foo;
    }
}
```

edited May 23 '17 at 12:26
Community ♦
1 1

answered Sep 17 '16 at 10:32
Biscuits
1,277 1 9 19

5 It is true that you "may omit the outer namespace," but it doesn't mean you should. To me, this is another argument as to why using directives (other than aliases as in @Neo's answer) should go outside the namespace, to force fully-qualified namespace names. – Keith Robertson Nov 3 '16 at 15:39

187

Putting it inside the namespaces makes the declarations local to that namespace for the file (in case you have multiple namespaces in the file) but if you only have one namespace per file then it doesn't make much of a difference whether they go outside or inside the namespace.

```
using ThisNamespace.IsImported.InAllNamespaces.Here;

namespace Namespace1
```

```
{
    using ThisNamespace.IsImported.InNamespace1.AndNamespace2;

    namespace Namespace2
    {
        using ThisNamespace.IsImported.InJustNamespace2;
    }
}

namespace Namespace3
{
    using ThisNamespace.IsImported.InJustNamespace3;
}
```

edited Apr 17 '16 at 14:49

answered Sep 24 '08 at 3:52
Mark Cidade
85.3k 29 205 226

namespaces provide a logical separation, not a physical (file) one. – Jowen Sep 26 '13 at 11:08

9 It's not quite true that there is no difference; `using` directives within `namespace` blocks can refer to relative namespaces based on the enclosing `namespace` block. – O. R. Mapper Feb 9 '14 at 10:07

66 yeah I know. we established that in this question's accepted answer five years ago. – Mark Cidade Feb 10 '14 at 21:58

31

There is an issue with placing using statements inside the namespace when you wish to use aliases. The alias doesn't benefit from the earlier `using` statements and has to be fully qualified.

Consider:

```
namespace MyNamespace
{
    using System;
    using MyAlias = System.DateTime;

    class MyClass
    {
    }
}
```

versus:

```
using System;

namespace MyNamespace
{
    using MyAlias = DateTime;

    class MyClass
    {
    }
}
```

This can be particularly pronounced if you have a long-winded alias such as the following (which is how I found the problem):

```
using MyAlias = Tuple<Expression<Func<DateTime, object>>, Expression
object>>>;
```

With `using` statements inside the namespace, it suddenly becomes:

```
using MyAlias =
System.Tuple<System.Linq.Expressions.Expression<System.Func<System.Da
System.Linq.Expressions.Expression<System.Func<System.TimeSpan, obje
```

Not pretty.

edited Mar 24 '16 at 1:57

answered Oct 10 '12 at 18:47

Neo
1,509 3 24 45

1 Your `class` needs a name (identifier). You cannot have a `using` directive inside a class as you indicate. It must be on a namespace level, for example outside the outermost `namespace`, or just inside the innermost `namespace` (but not inside a class/interface/etc.). – Jeppe Stig Nielsen Mar 23 '16 at 11:51

@JeppeStigNielsen Thanks. I misplaced the `using` directives mistakenly. I've edited it to how I intended it to be. Thanks for pointing out. The reasoning is still the same, though. – Neo Mar 23 '16 at 20:54

-8

It is a better practice if those **default** using i.e. "*references*" used in your source solution should be outside the namespaces and those that are **"new added reference"** is a good practice is you should put it inside the namespace. This is to distinguish what references are being added.

edited Oct 14 '14 at 21:37

answered Oct 14 '14 at 21:30

Israel Ocbina
164 3 6

6 No, actually that is a bad idea. You should not base the location between locally scoped and globally scoped of using directives on the fact that they are newly added or not. Instead, it is good practice to alphabetize them, except for BCL references, which should go on top. – Abel Nov 3 '14 at 14:01

389

This thread already has some great answers, but I feel I can bring a little more detail with this additional answer.

First, remember that a namespace declaration with periods, like:

```
namespace MyCorp.TheProduct.SomeModule.Utilities
{
    ...
}
```

is entirely equivalent to:

```
namespace MyCorp
{
    namespace TheProduct
    {
        namespace SomeModule
        {
            namespace Utilities
            {
                ...
            }
        }
    }
}
```

If you wanted to, you could put `using` directives on all of these levels. (Of course, we want to have `using` s in only one place, but it would be legal according to the language.)

The rule for resolving which type is implied, can be loosely stated like this: **First search the inner-most "scope" for a match, if nothing is found there go out one level to the next scope and search there, and so on**, until a match is found. If at some level more than one match is found, if one of the types are from the current assembly, pick that one and issue a compiler warning. Otherwise, give up (compile-time error).

Now, let's be explicit about what this means in a concrete example with the two major conventions.

(1) With usings outside:

```
using System;
using System.Collections.Generic;
using System.Linq;
//using MyCorp.TheProduct;  <-- uncommenting this would change nothi
using MyCorp.TheProduct.OtherModule;
using MyCorp.TheProduct.OtherModule.Integration;
using ThirdParty;

namespace MyCorp.TheProduct.SomeModule.Utilities
{
    class C
    {
        Ambiguous a;
    }
}
```

In the above case, to find out what type `Ambiguous` is, the search goes in this order:

1. Nested types inside `C` (including inherited nested types)
2. Types in the current namespace `MyCorp.TheProduct.SomeModule.Utilities`
3. Types in namespace `MyCorp.TheProduct.SomeModule`
4. Types in `MyCorp.TheProduct`
5. Types in `MyCorp`
6. Types in the *null* namespace (the global namespace)
7. Types in `System`, `System.Collections.Generic`, `System.Linq`, `MyCorp.TheProduct.OtherModule`, `MyCorp.TheProduct.OtherModule.Integration`, and `ThirdParty`

The other convention:

(2) With usings inside:

```
namespace MyCorp.TheProduct.SomeModule.Utilities
{
    using System;
    using System.Collections.Generic;
    using System.Linq;
    using MyCorp.TheProduct;                          // MyCorp can
using is NOT redundant
    using MyCorp.TheProduct.OtherModule;              // MyCorp.The
out
    using MyCorp.TheProduct.OtherModule.Integration;   // MyCorp.The
out
    using ThirdParty;

    class C
    {
        Ambiguous a;
    }
}
```

Now, search for the type `Ambiguous` goes in this order:

1. Nested types inside `C` (including inherited nested types)
2. Types in the current namespace `MyCorp.TheProduct.SomeModule.Utilities`
3. Types in `System` , `System.Collections.Generic` , `System.Linq` , `MyCorp.TheProduct` , `MyCorp.TheProduct.OtherModule` , `MyCorp.TheProduct.OtherModule.Integration` , and `ThirdParty`
4. Types in namespace `MyCorp.TheProduct.SomeModule`
5. Types in `MyCorp`
6. Types in the *null* namespace (the global namespace)

(Note that `MyCorp.TheProduct` was a part of "3." and was therefore not needed between "4." and "5.".)

Concluding remarks

No matter if you put the usings inside or outside the namespace declaration, there's always the possibility that someone later adds a new type with identical name to one of the namespaces which have higher priority.

Also, if a nested namespace has the same name as a type, it can cause problems.

It is always dangerous to move the usings from one location to another because the search hierarchy changes, and another type may be found. Therefore, choose one convention and stick to it, so that you won't have to ever move usings.

Visual Studio's templates, by default, put the usings *outside* of the namespace (for example if you make VS generate a new class in a new file).

One (tiny) advantage of having usings *outside* is that you can then utilize the using directives for a global attribute, for example `[assembly: ComVisible(false)]` instead of `[assembly: System.Runtime.InteropServices.ComVisible(false)]` .

edited Nov 8 '13 at 9:57
peSHIr
5,253 1 28 44

answered Apr 18 '13 at 21:00
Jeppe Stig Nielsen
44k 6 74 134

26 This is the best explanation, because it highlights the fact that the position of the 'using' statements is a deliberate decision from the developer. In no case should someone carelessly change the location of the 'using' statements without understanding the implications. Therefore the StyleCop rule is just dumb. – ZunTzu Jan 8 '16 at 9:32

According to Hanselman - Using Directive and Assembly Loading...

59

and other such articles there is technically no difference.

My preference is to put them outside of namespaces.

edited Oct 22 '12 at 14:46

answered Sep 24 '08 at 3:53

Quintin Robinson
69.1k 14 108 128

2 @Chris M: uh... the link posted in the answer indicates there's *no* benefit to in vs. out, actually showing an example that falsifies the claim made in the link you posted... – johnny Aug 19 '11 at 18:07

2 Aye I didn't fully read the thread but bought in when the MVPs said it was right. A guy disproves it, explains it and shows his code further down... "The IL that the C# compiler generates is the same in either case. In fact the C# compiler generates precisely nothing corresponding to each using directive. Using directives are purely a C#ism, and they have no meaning to .NET itself. (Not true for using statements but those are something quite different.)" groups.google.com/group/wpf-disciples/msg/781738deb0a15c46 – Chris McKee Aug 24 '11 at 21:52

78 Please include a summary of the link. **When** the link is broken (because it *will* happen, given enough time), suddenly an answer with 32 upvotes is only worth `My style is to put them outside the namespaces.` - barely an answer at all. – ANeves Oct 22 '12 at 7:45

7 The claim here is simply wrong ... there is a technical difference and your own citation says so ... in fact, that's what it's all about. Please delete this mistaken answer ... there are far better, and accurate, ones. – Jim Balter Apr 17 '16 at 3:44

46

According the to StyleCop Documentation:

SA1200: UsingDirectivesMustBePlacedWithinNamespace

Cause A C# using directive is placed outside of a namespace element.

Rule Description A violation of this rule occurs when a using directive or a using-alias directive is placed outside of a namespace element, unless the file does not contain any namespace elements.

For example, the following code would result in two violations of this rule.

```
using System;
using Guid = System.Guid;

namespace Microsoft.Sample
{
    public class Program
    {
    }
}
```

The following code, however, would not result in any violations of this rule:

```
namespace Microsoft.Sample
{
    using System;
    using Guid = System.Guid;

    public class Program
    {
    }
}
```

This code will compile cleanly, without any compiler errors. However, it is unclear which version of the Guid type is being allocated. If the using directive is moved inside of the namespace, as shown below, a compiler error will occur:

```
namespace Microsoft.Sample
{
    using Guid = System.Guid;
    public class Guid
    {
        public Guid(string s)
        {
        }
    }

    public class Program
    {
        public static void Main(string[] args)
        {
            Guid g = new Guid("hello");
        }
    }
}
```

The code fails on the following compiler error, found on the line containing `Guid g = new Guid("hello");`

CS0576: Namespace 'Microsoft.Sample' contains a definition conflicting with alias 'Guid'

The code creates an alias to the System.Guid type called Guid, and also creates its own type called Guid with a matching constructor interface. Later, the code creates an instance of the type Guid. To create this instance, the compiler must choose between the two different definitions of Guid. When the using-alias directive is placed outside of the namespace element, the compiler will choose the local definition of Guid defined within the local namespace, and completely ignore the using-alias directive defined outside of the namespace. This, unfortunately, is not obvious when reading the code.

When the using-alias directive is positioned within the namespace, however, the compiler has to choose between two different, conflicting Guid types both defined within the same namespace. Both of these types provide a matching constructor. The compiler is unable to make a decision, so it flags the compiler error.

Placing the using-alias directive outside of the namespace is a bad practice because it can lead to confusion in situations such as this, where it is not obvious which version of the type is actually being used. This can potentially lead to a bug which might be difficult to diagnose.

Placing using-alias directives within the namespace element eliminates this as a source of bugs.

2. Multiple Namespaces

Placing multiple namespace elements within a single file is generally a bad idea, but if and when this is done, it is a good idea to place all using directives within each of the namespace elements, rather than globally at the top of the file. This will scope the namespaces tightly, and will also help to avoid the kind of behavior described above.

It is important to note that when code has been written with using directives placed outside of the namespace, care should be taken when moving these directives within the namespace, to ensure that this is not changing the semantics of the code. As explained above, placing using-alias directives within the namespace element allows the compiler to choose between conflicting types in ways that will not happen when the directives are placed outside of the namespace.

How to Fix Violations To fix a violation of this rule, move all using directives and using-alias directives within the namespace element.

answered Sep 14 '09 at 15:17

JaredCacurak
781 10 18

@Jared - as I noted in my answer, my prefered workaround / solution is to only ever have one class per file. I think that this is a fairly common convention. – benPearce Sep 14 '09 at 21:01

23 Indeed, it's also a StyleCop rule! SA1402: A C# document may only contain a single class at the root level unless all of the classes are partial

and are of the same type. Showcasing one rule by breaking another just drips with wrong sauce. – Task Mar 5 '10 at 18:09

6 Upvoted for being the first answer to actually cover it from the StyleCop perspective. Personally I like the visual feel of `using` s outside the namespace. Inner `using` s looks so ugly to me. :) – nawfal Jun 6 '15 at 10:45

2 Finally a good answer to the question. And benPearce's comment is irrelevant ... this has nothing to do with the number of classes in the file. – Jim Balter Apr 17 '16 at 3:51