What is global::?

Ask Question

88

In C# I see `global::` used quite often in auto-generated code. It is not something I have ever used myself so I don't know what the purpose is. Can someone explain this?

c#

7

asked Feb 22 '13 at 10:50

Sachin Kainth
20k 66 166 258

1 stackoverflow.com/questions/3552763/… – Massimiliano Peluso Feb 22 '13 at 10:51

2 From msdn: msdn.microsoft.com/en-us/library/c3ay4x3d.aspx – Sina Iravanian Feb 22 '13 at 10:52

Possible duplicate of Why use the global keyword in C#? – Daniel B Jul 17 '18 at 19:04

4 Answers

global refers to the global namespace, it can be used to solve problems whereby you may redefine types. For example:

78

```
class foo
{
    class System
    {

    }

}
```

If you were to use System where it would be locally scoped in the foo class, you could use:

```
global::System.Console.WriteLine("foobar");
```

to access the global namespace.

Example

```
using System;

class Foo
{
    public void baz()
    {
        Console.WriteLine("Foo 1");
    }
}

namespace Demo
{
    class Foo
    {
        public void baz()
        {
            Console.WriteLine("Foo 2");
        }
    }

    class Program
    {
        protected static global::Foo bar = new global::Foo();

        static void Main(string[] args)
```

```
        {
            bar.baz(); // would write Foo 1 to console as it refers
            Foo qux = new Foo();
            qux.baz(); // would write Foo 2 to the console as it ref
namespace
        }
    }
}
```

edited Jun 8 '17 at 8:39

Edgar Rokjān
15.2k 4 30 53

answered Feb 22 '13 at 10:55

chrisw
2,367 1 19 30

What would happen if I has an auto-generated class with a class prefixed with the global namespace and the class was called Foo I had a class I had created which also was called Foo also with no namespace? – Sachin Kainth Feb 22 '13 at 11:07

If the auto generated class was prefixed with the global namespace and was called Foo, global would refer to the class in the global namespace. There can only be a single definition with the same name in any namespace. If you were to create an instance of the class from within another namespace whereby you defined another meaning for Foo, it would take the most local scoped. See the edit – chrisw Feb 22 '13 at 11:19

18

It's a sometime-necessary prefix indicating the root namespace.

It's often added to generated code to avoid name clashes with user code.

For example, imagine you had a class called `System`, but then you wanted to use `System.String`. You could use `global::System.String`

to differentiate.

I believe the `::` comes from C++ where it's used as a namespace separator.

In practice I've never used it, other than in generating code. Note that you can also get around some conflicts via using aliases. For example `using String = System.String;`

answered Feb 22 '13 at 10:52

Drew Noakes
189k 118 529 621

12

The `global` contextual keyword, when it comes before the :: operator, refers to the global namespace, which is the default namespace for any C# program and is otherwise unnamed.

The `global::` specifier tells the compiler to start looking for the namespace or class starting from the root. You'll see it in system-generated code so that the code always works. That way if you have a namespace right under your current namespace that is the same as the top level namespace the code is trying to access, there won't be a conflict.

For example, say you have namespace A and namespace B and namespace B.A if I write code in namespace B.A that needs to reference a class in namespace A, without global:: I have no way of getting to it. If I reference A.classname, the compiler will look for classname in B.A. With global:: I can tell it to look for classname in global::A.classname and it will find classname in the proper location.

answered Feb 22 '13 at 10:59

coder
1,682 3 17 31

---

13 If you take content from online resources, please be kind enough to

provide the proper attribution: MSDN and What is the global keyword in C# – Bart Feb 26 '13 at 10:47

2 k..@Bart thanks for your comment I will keep in mind when i am answering nexttime..But why is that downvote? – coder Feb 26 '13 at 11:29

Thats not mine. – Bart Feb 26 '13 at 11:34

1 why is that downvote to my answer? – coder Feb 26 '13 at 11:35

1 @Bart k sorry for asking.. – coder Feb 26 '13 at 11:36

0

The `global::` namespace and its identifier is not what most people think. It is not a universal identifier of everything created in an application that lies outside one of your application's defined namespaces and which is attached to some global root.

If you create a class or type outside your top level namespaces you would assume its automatically a part of the GLOBAL namespace and accessible by the `global::` identifier in all files in your application or assembly. In fact those names are more often in that file's compiled LOCAL scope only, yet are accessible via the `global::` identifier.

If you create a top level class or namespace in an aspx.cs file it is accessible via `global::` from the global namespace in that file. But if you type `global::` in another file, that class and namespace doesn't exist in the global namespace. If you create that same class or namespace in a class.cs file however, those items are available to all other files via `global::` and in the global namespace as well as that files local scope. Why?

It turns out `global::` is really a reference to top level LOCAL names under the file's scope as well as GLOBAL names shared by the assembly (like what might be compiled in your App_Code class files in a typical ASP.NET project).

I found this very confusing and not consistent, since `global::` implies access to top-level namespaces and types created in the application that are tied to the global namespace. Some like "System" are tied to the global namespace by default in all files, but custom ones may or may not be depending on the scope of that file. That is why the global identifier has a secondary role of resolving references to your local root scope names as well.

You can test this by creating top level namespaces and classes in parts of your application then using `global::` to see which ones it can access in the global namespace from different parts of your application and which ones it cannot. The one's it cannot access are clearly assigned to a "local global scope" in that file only, which `global::` helps you access in naming conflicts.

edited Sep 20 '18 at 21:27

answered Sep 20 '18 at 21:06

Stokely
485 4 5