# Contents

# Namespaces (C# Programming Guide)

1/23/2019 • 2 minutes to read • Edit Online

Namespaces are heavily used in C# programming in two ways. First, the .NET Framework uses namespaces to organize its many classes, as follows:

```
System.Console.WriteLine("Hello World!");
```

`System` is a namespace and `Console` is a class in that namespace. The `using` keyword can be used so that the complete name is not required, as in the following example:

```
using System;
```

```
Console.WriteLine("Hello");
Console.WriteLine("World!");
```

For more information, see the using Directive.

Second, declaring your own namespaces can help you control the scope of class and method names in larger programming projects. Use the namespace keyword to declare a namespace, as in the following example:

```
namespace SampleNamespace
{
    class SampleClass
    {
        public void SampleMethod()
        {
            System.Console.WriteLine(
              "SampleMethod inside SampleNamespace");
        }
    }
}
```

The name of the namespace must be a valid C# identifier name.

## Namespaces Overview

Namespaces have the following properties:

- They organize large code projects.
- They are delimited by using the `.` operator.
- The `using` directive obviates the requirement to specify the name of the namespace for every class.
- The `global` namespace is the "root" namespace: `global::System` will always refer to the .NET System namespace.

## C# Language Specification

For more information, see the C# Language Specification. The language specification is the definitive source for C# syntax and usage.

## See also

- Using Namespaces
- How to: Use the Global Namespace Alias
- How to: Use the My Namespace
- C# Programming Guide
- Identifier names
- Namespace Keywords
- using Directive
- :: Operator
- . Operator

# Using Namespaces (C# Programming Guide)

1/23/2019 • 3 minutes to read • Edit Online

Namespaces are heavily used within C# programs in two ways. Firstly, the .NET Framework classes use namespaces to organize its many classes. Secondly, declaring your own namespaces can help control the scope of class and method names in larger programming projects.

## Accessing Namespaces

Most C# applications begin with a section of `using` directives. This section lists the namespaces that the application will be using frequently, and saves the programmer from specifying a fully qualified name every time that a method that is contained within is used.

For example, by including the line:

```
using System;
```

At the start of a program, the programmer can use the code:

```
Console.WriteLine("Hello, World!");
```

Instead of:

```
System.Console.WriteLine("Hello, World!");
```

## Namespace Aliases

The using Directive can also be used to create an alias for a namespace. For example, if you are using a previously written namespace that contains nested namespaces, you might want to declare an alias to provide a shorthand way of referencing one in particular, as in the following example:

```
using Co = Company.Proj.Nested;  // define an alias to represent a namespace
```

## Using Namespaces to control scope

The `namespace` keyword is used to declare a scope. The ability to create scopes within your project helps organize code and lets you create globally-unique types. In the following example, a class titled `SampleClass` is defined in two namespaces, one nested inside the other. The . Operator is used to differentiate which method gets called.

```
namespace SampleNamespace
{
    class SampleClass
    {
        public void SampleMethod()
        {
            System.Console.WriteLine(
              "SampleMethod inside SampleNamespace");
        }
    }

    // Create a nested namespace, and define another class.
    namespace NestedNamespace
    {
        class SampleClass
        {
            public void SampleMethod()
            {
                System.Console.WriteLine(
                  "SampleMethod inside NestedNamespace");
            }
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            // Displays "SampleMethod inside SampleNamespace."
            SampleClass outer = new SampleClass();
            outer.SampleMethod();

            // Displays "SampleMethod inside SampleNamespace."
            SampleNamespace.SampleClass outer2 = new SampleNamespace.SampleClass();
            outer2.SampleMethod();

            // Displays "SampleMethod inside NestedNamespace."
            NestedNamespace.SampleClass inner = new NestedNamespace.SampleClass();
            inner.SampleMethod();
        }
    }
}
```

## Fully Qualified Names

Namespaces and types have unique titles described by fully qualified names that indicate a logical hierarchy. For example, the statement `A.B` implies that `A` is the name of the namespace or type, and `B` is nested inside it.

In the following example, there are nested classes and namespaces. The fully qualified name is indicated as a comment following each entity.

```
namespace N1     // N1
{
    class C1      // N1.C1
    {
        class C2   // N1.C1.C2
        {
        }
    }
    namespace N2  // N1.N2
    {
        class C2   // N1.N2.C2
        {
        }
    }
}
```

In the previous code segment:

- The namespace `N1` is a member of the global namespace. Its fully qualified name is `N1`.
- The namespace `N2` is a member of `N1`. Its fully qualified name is `N1.N2`.
- The class `C1` is a member of `N1`. Its fully qualified name is `N1.C1`.
- The class name `C2` is used two times in this code. However, the fully qualified names are unique. The first instance of `C2` is declared inside `C1`; therefore, its fully qualified name is: `N1.C1.C2`. The second instance of `C2` is declared inside a namespace `N2`; therefore, its fully qualified name is `N1.N2.C2`.

Using the previous code segment, you can add a new class member, `C3`, to the namespace `N1.N2` as follows:

```
namespace N1.N2
{
    class C3   // N1.N2.C3
    {
    }
}
```

In general, use `::` to reference a namespace alias or `global::` to reference the global namespace and `.` to qualify types or members.

It is an error to use `::` with an alias that references a type instead of a namespace. For example:

```
using Alias = System.Console;
```

```
class TestClass
{
    static void Main()
    {
        // Error
        //Alias::WriteLine("Hi");

        // OK
        Alias.WriteLine("Hi");
    }
}
```

Remember that the word `global` is not a predefined alias; therefore, `global.X` does not have any special meaning. It acquires a special meaning only when it is used with `::`.

Compiler warning CS0440 is generated if you define an alias named global because `global::` always references the global namespace and not an alias. For example, the following line generates the warning:

```
using global = System.Collections;   // Warning
```

Using `::` with aliases is a good idea and protects against the unexpected introduction of additional types. For example, consider this example:

```
using Alias = System;
```

```
namespace Library
{
    public class C : Alias.Exception { }
}
```

This works, but if a type named `Alias` were to subsequently be introduced, `Alias.` would bind to that type instead. Using `Alias::Exception` insures that `Alias` is treated as a namespace alias and not mistaken for a type.

See the topic How to: Use the Global Namespace Alias for more information regarding the `global` alias.

## See also

- C# Programming Guide
- Namespaces
- Namespace Keywords
- . Operator
- :: Operator
- extern

# How to: Use the Global Namespace Alias (C# Programming Guide)

1/23/2019 • 2 minutes to read • Edit Online

The ability to access a member in the global namespace is useful when the member might be hidden by another entity of the same name.

For example, in the following code, `Console` resolves to `TestApp.Console` instead of to the `Console` type in the System namespace.

```
using System;
```

```
class TestApp
{
    // Define a new class called 'System' to cause problems.
    public class System { }

    // Define a constant called 'Console' to cause more problems.
    const int Console = 7;
    const int number = 66;

    static void Main()
    {
        // The following line causes an error. It accesses TestApp.Console,
        // which is a constant.
        //Console.WriteLine(number);
    }
}
```

Using `System.Console` still results in an error because the `System` namespace is hidden by the class `TestApp.System`:

```
// The following line causes an error. It accesses TestApp.System,
// which does not have a Console.WriteLine method.
System.Console.WriteLine(number);
```

However, you can work around this error by using `global::System.Console`, like this:

```
// OK
global::System.Console.WriteLine(number);
```

When the left identifier is `global`, the search for the right identifier starts at the global namespace. For example, the following declaration is referencing `TestApp` as a member of the global space.

```
class TestClass : global::TestApp
```

Obviously, creating your own namespaces called `System` is not recommended, and it is unlikely you will encounter any code in which this has happened. However, in larger projects, it is a very real possibility that namespace duplication may occur in one form or another. In these situations, the global namespace qualifier is your guarantee that you can specify the root namespace.

## Example

In this example, the namespace `System` is used to include the class `TestClass` therefore, `global::System.Console` must be used to reference the `System.Console` class, which is hidden by the `System` namespace. Also, the alias `colAlias` is used to refer to the namespace `System.Collections`; therefore, the instance of a System.Collections.Hashtable was created using this alias instead of the namespace.

```
using colAlias = System.Collections;
namespace System
{
    class TestClass
    {
        static void Main()
        {
            // Searching the alias:
            colAlias::Hashtable test = new colAlias::Hashtable();

            // Add items to the table.
            test.Add("A", "1");
            test.Add("B", "2");
            test.Add("C", "3");

            foreach (string name in test.Keys)
            {
                // Searching the global namespace:
                global::System.Console.WriteLine(name + " " + test[name]);
            }
        }
    }
}
```

**A 1 B 2 C 3**

## See also

- C# Programming Guide
- Namespaces
- . Operator
- :: Operator
- extern

# How to: Use the My Namespace (C# Programming Guide)

1/23/2019 • 2 minutes to read • Edit Online

The Microsoft.VisualBasic.MyServices namespace ( `My` in Visual Basic) provides easy and intuitive access to a number of .NET Framework classes, enabling you to write code that interacts with the computer, application, settings, resources, and so on. Although originally designed for use with Visual Basic, the `MyServices` namespace can be used in C# applications.

For more information about using the `MyServices` namespace from Visual Basic, see Development with My.

## Adding a Reference

Before you can use the `MyServices` classes in your solution, you must add a reference to the Visual Basic library.

**To add a reference to the Visual Basic library**

1. In **Solution Explorer**, right-click the **References** node, and select **Add Reference**.
2. When the **References** dialog box appears, scroll down the list, and select Microsoft.VisualBasic.dll.

   You might also want to include the following line in the `using` section at the start of your program.

```
using Microsoft.VisualBasic.Devices;
```

## Example

This example calls various static methods contained in the `MyServices` namespace. For this code to compile, a reference to Microsoft.VisualBasic.DLL must be added to the project.

```
using System;
using Microsoft.VisualBasic.Devices;

class TestMyServices
{
    static void Main()
    {
        // Play a sound with the Audio class:
        Audio myAudio = new Audio();
        Console.WriteLine("Playing sound...");
        myAudio.Play(@"c:\WINDOWS\Media\chimes.wav");

        // Display time information with the Clock class:
        Clock myClock = new Clock();
        Console.Write("Current day of the week: ");
        Console.WriteLine(myClock.LocalTime.DayOfWeek);
        Console.Write("Current date and time: ");
        Console.WriteLine(myClock.LocalTime);

        // Display machine information with the Computer class:
        Computer myComputer = new Computer();
        Console.WriteLine("Computer name: " + myComputer.Name);

        if (myComputer.Network.IsAvailable)
        {
            Console.WriteLine("Computer is connected to network.");
        }
        else
        {
            Console.WriteLine("Computer is not connected to network.");
        }
    }
}
```

Not all the classes in the `MyServices` namespace can be called from a C# application: for example, the FileSystemProxy class is not compatible. In this particular case, the static methods that are part of FileSystem, which are also contained in VisualBasic.dll, can be used instead. For example, here is how to use one such method to duplicate a directory:

```
// Duplicate a directory
Microsoft.VisualBasic.FileIO.FileSystem.CopyDirectory(
    @"C:\original_directory",
    @"C:\copy_of_original_directory");
```

## See also

- C# Programming Guide
- Namespaces
- Using Namespaces