

C# - NAMESPACES

http://www.tutorialspoint.com/csharp/csharp_namespaces.htm

Copyright © tutorialspoint.com

A **namespace** is designed for providing a way to keep one set of names separate from another. The class names declared in one namespace does not conflict with the same class names declared in another.

Defining a Namespace

A namespace definition begins with the keyword **namespace** followed by the namespace name as follows:

```
namespace namespace_name
{
    // code declarations
}
```

To call the namespace-enabled version of either function or variable, prepend the namespace name as follows:

```
namespace_name.item_name;
```

The following program demonstrates use of namespaces:

```
using System;
namespace first_space
{
    class namespace_cl
    {
        public void func()
        {
            Console.WriteLine("Inside first_space");
        }
    }
}

namespace second_space
{
    class namespace_cl
    {
        public void func()
        {
            Console.WriteLine("Inside second_space");
        }
    }
}

class TestClass
{
    static void Main(string[] args)
    {
        first_space.namespace_cl fc = new first_space.namespace_cl();
        second_space.namespace_cl sc = new second_space.namespace_cl();
        fc.func();
        sc.func();
        Console.ReadKey();
    }
}
```

When the above code is compiled and executed, it produces the following result:

```
Inside first_space
Inside second_space
```

The *using* Keyword

The **using** keyword states that the program is using the names in the given namespace. For example, we are using the **System** namespace in our programs. The class Console is defined there. We just write:

```
Console.WriteLine ("Hello there");
```

We could have written the fully qualified name as:

```
System.Console.WriteLine("Hello there");
```

You can also avoid prepending of namespaces with the **using** namespace directive. This directive tells the compiler that the subsequent code is making use of names in the specified namespace. The namespace is thus implied for the following code:

Let us rewrite our preceding example, with using directive:

```
using System;
using first_space;
using second_space;

namespace first_space
{
    class abc
    {
        public void func()
        {
            Console.WriteLine("Inside first_space");
        }
    }
}

namespace second_space
{
    class efg
    {
        public void func()
        {
            Console.WriteLine("Inside second_space");
        }
    }
}

class TestClass
{
    static void Main(string[] args)
    {
        abc fc = new abc();
        efg sc = new efg();
        fc.func();
        sc.func();
        Console.ReadKey();
    }
}
```

When the above code is compiled and executed, it produces the following result:

```
Inside first_space
Inside second_space
```

Nested Namespaces

You can define one namespace inside another namespace as follows:

```
namespace namespace_name1
{
    // code declarations
    namespace namespace_name2
    {
        // code declarations
    }
}
```

You can access members of nested namespace by using the dot . operator as follows:

```
using System;
using first_space;
using first_space.second_space;

namespace first_space
{
    class abc
    {
        public void func()
        {
            Console.WriteLine("Inside first_space");
        }
    }
    namespace second_space
    {
        class efg
        {
            public void func()
            {
                Console.WriteLine("Inside second_space");
            }
        }
    }
}

class TestClass
{
    static void Main(string[] args)
    {
        abc fc = new abc();
        efg sc = new efg();
        fc.func();
        sc.func();
        Console.ReadKey();
    }
}
```

When the above code is compiled and executed, it produces the following result:

```
Inside first_space
Inside second_space
Loading [MathJax]/jax/output/HTML-CSS/jax.js
```