This chapter presents a quick overview of the C# language. It assumes a certain level of programming knowledge and therefore doesn’t present very much detail. If the explanation here doesn’t make sense, look for a more detailed explanation of the particular topic later in the book.

The second part of the chapter discusses how to obtain the C# compiler and the advantages of using Visual Studio to develop C# applications.

Hello, Universe

As a supporter of SETI,¹ I thought that it would be appropriate to do a “Hello, Universe” program rather than the canonical “Hello, World” program.

```csharp
using System;
class Hello
{
    public static void Main(string[] args)
    {
        Console.WriteLine("Hello, Universe");

        // iterate over command-line arguments,
        // and print them out
        for (int arg = 0; arg < args.Length; arg++)
        {
            Console.WriteLine("Arg {0}: {1}", arg, args[arg]);
        }
    }
}
```

As discussed earlier, the .NET Runtime has a unified namespace for all program information (or metadata). The using System clause is a way of referencing the classes that are in the System namespace so they can be used without having to put System in front of the type name. The System namespace contains many useful classes, one of which is the Console class, which is used (not surprisingly) to communicate with the console (or DOS box, or command line, for those who have never seen a console).

Because there are no global functions in C#, the example declares a class called Hello that contains the static Main() function, which serves as the starting point for execution. Main() can be declared with no parameters or with a string array. Since it’s the starting function, it must be a static function, which means it isn’t associated with an instance of an object.

¹Search for Extraterrestrial Intelligence. See [http://www.teamseti.org](http://www.teamseti.org) for more information.
The first line of the function calls the `WriteLine()` function of the `Console` class, which will write “Hello, Universe” to the console. The `for` loop iterates over the parameters that are passed in and then writes out a line for each parameter on the command line.

## Namespace and Using Statements

Namespaces in the .NET Runtime are used to organize classes and other types into a single hierarchical structure. The proper use of namespaces will make classes easy to use and prevent collisions with classes written by other authors.

Namespaces can also be thought of as a way to specify long and useful names for classes and other types without having to always type a full name.

Namespaces are defined using the `namespace` statement. For multiple levels of organization, namespaces can be nested:

```csharp
namespace Outer
{
    namespace Inner
    {
        class MyClass
        {
            public static void Function() {}
        }
    }
}
```

That’s a fair amount of typing and indenting, so it can be simplified by using the following instead:

```csharp
namespace Outer.Inner
{
    class MyClass
    {
        public static void Function() {}
    }
}
```

A source file can define more than one namespace, but in the majority of cases, all the code within one file lives in a single namespace.

The fully qualified name of a class—the name of the namespace followed by the name of the class—can become quite long. The following is an example of such a class:

```csharp
System.Xml.Serialization.Advanced.SchemaImporterExtension
```

It would be very tedious to have to write that full class name every time we wanted to use it, so we can add a `using` statement:

```csharp
using System.Xml.Serialization.Advanced;
```

This statement says, “treat all of the types defined inside this namespace as if they don’t have a namespace in front of them,” which allows us to use

```csharp
SchemaImporterExtension
```