In this chapter, I’ll introduce you to the key components of C# and the .NET Framework and explain the relationship between them. I’ll also introduce the major technology building blocks that form the functionality that you’ll look at throughout this book.

I’m not going to get bogged down in detail; instead, I will going to focus on the knowledge you need to put everything in context before you start learning the C# language and becoming an effective C# programmer. This chapter is about giving you a good overview that will help you form a picture of C# and .NET as you move through the book.

.NET can seem impenetrable—a wall of terms and technologies and a pit of languages and syntaxes. By the time you reach the final chapter of this book, you’ll have forgotten this moment and you’ll be up and running with C# and .NET. The process in between will be surprisingly easy. I use lots of examples, I don’t bombard you with the tedious detail, and I tell you what you need to know to understand how to use the feature at hand.

Your interest will be piqued as you move from chapter to chapter. There are a couple of .NET features that I am particularly fond of, and I can remember wanting to dive into the details as soon as I saw them. If you are like me, you’ll have the same reaction, although the features that interest you may be different. To that end, this chapter includes sources for further reading that can give you the detail on individual topics.

C# and the .NET Framework at a Glance
Let’s introduce the main characters in this story – C# and the .NET Framework. Having a clear view of how the various terms and technologies associated with .NET fit together will help you when you design your C# programs and will make the process of learning about .NET easier.

Introducing C#
C# is a programming language. Like any language, C# has a syntax and a vocabulary. Programming languages, like C# or Java, differ from natural languages, such as English or French. Learning a new programming language is easier than learning a natural language. There are only a handful of words in a programming language vocabulary; C# has around 100 keywords. That may seem daunting, but you’ll use some keywords more than others and you probably won’t use some of them at all.

Programming languages have very rigid syntaxes—the keywords have to be used in a certain way. English is fluid and flexible because it is used for so many different reasons. Programming languages are
terse and precise because they are used for one reason: to express your instructions to the computer as clearly and unambiguously as possible. Helpfully, there is a universal, authoritative specification for C#. Grammarians can vigorously debate the history and meaning of words in English, but in C# the history of all keywords is that they were defined by Microsoft and the meaning is contained in the C# language specification, which you can find at http://msdn.microsoft.com/en-us/library/ms228593.aspx. I am not suggesting that you run off and read the language specification right now; it is a dry, precise, overly-technical document in the way that formal specifications tend to be. But if you are ever in doubt about the exact meaning and use of a C# term, the specification provides the authoritative reference.

Introducing the .NET Framework

The .NET Framework runs .NET programs. For client applications, this means that the user must install the framework before running your program for the first time (although many developers include the .NET Framework in their program installers to make life easier for the user). For servers, the .NET Framework must be installed before running .NET-based services, such as web applications. There are two parts to the .NET Framework, described in the following sections.

Introducing the Common Language Runtime

The Common Language Runtime (CLR) is the part that actually executes .NET programs – i.e. performs the instructions that you gave using C#. The CLR (commonly referred to as the runtime) is a virtual machine, an idea you will be familiar with if you have used Java. In essence, the CLR provides the services that provide and enforce the .NET feature set, freeing the programmer from working directly with the operating system. The CLR lets the Microsoft .NET designers add features to make life easier for programmers (such as automatic memory management, described in the “Understanding Automatic Memory Management” section) and remove or restrict problematic features (such as unsafe code, covered in the “Understanding Unmanaged/Native Code Support” section).

The C# compiler translates your C# code into Common Intermediate Language (CIL) instructions, which is the language of the CLR. When you run your compiled program, the CLR loads the CIL and gets to work.

Introducing the Class Library

The second part of the .NET Framework is the .NET Framework Class Library. A class is a mix of data and program logic (I explain classes in Chapter 6). The class library is an extensive collection of functionality and features that you can use to speed up your development process and access system features. For example, you don’t have to write your own classes for working with a relational database. There are classes in the class library that you can use to handle the connections, read the data, represent the SQL data types, and so on. You could use C# to implement all this functionality yourself, but you’ll find that the classes in the class library are comprehensive and well-thought out.

The tasks that you can perform with the library classes range from the basic (performing basic numeric calculations—see Chapter 5), to the advanced (creating graphical user interfaces—see Chapters 32-35) and from the run-of-the-mill (handling program errors—see Chapter 14) to the cutting-edge (parallel programming—see Chapter 24).