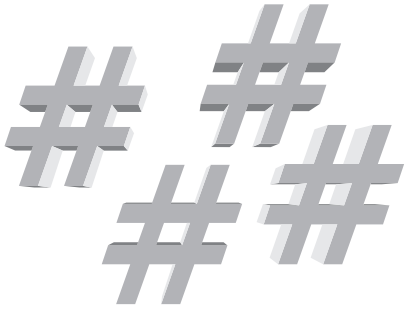
Chapter

3

Introduction to Programming in C#



Though this be madness, yet there is method in't

Hamlet Act 2, Scene 2, Shakespeare

'Plenty of practice', he went on repeating, all the time that Alice was getting him on his feet again, 'plenty of practice.'

The White Knight, *Through the Looking Glass and What Alice Found There*, Lewis Carroll

Aims

The aims of the chapter are:

- to look at a simple example, initially running from a DOS box or console session and then using Developer Studio;
- to look at simple text I/O;
- to look at simple numeric I/O.

3.1 Introduction

This chapter looks at some simple examples in C#. They have been chosen because you should already be familiar with the concepts involved from your experience with other programming languages.

If you already have a programming background in C, C++ or Java the syntax of the language will appear very familiar.

If you have a background in the Algol family of languages (Algol 60, Algol 68, Pascal, Modula 2 and Oberon) or the Fortran family (Fortran 66, Fortran 77, Fortran 90 and Fortran 95), the syntax will look a little strange. Don't worry as the underlying knowledge you have from one of these languages can be mapped fairly quickly onto C#.

Most people will be familiar with the following model of programming:

- write the program using an editor;
- compile the source using a compiler;
- link the output from the compiler to produce an executable;
- run the executable.

As most of us know, this is an iterative process. We rarely get it right first time. We first look at a C# example written using this approach.

3.2 Hello World Using the Console

The following is the classic hello world program in C#.

```
using System;
class HelloWorld
{
   static int Main()
   {
      Console.WriteLine("Hello World");
      return 0;
   }
}
```

This can be compiled and run at the MS-DOS command prompt. Click on Start->Programs->Microsoft Visual Studio .Net->Visual Studio .Net Tools->Visual Studio .Net command prompt

Type in the program using Notepad and save it with the name helloworld.cs. Then type:

```
csc helloworld.cs
```

This will compile the program, if there are no errors, and generate a file called helloworld.exe. To run the program, type:

```
helloworld
```

C# is case-sensitive so ensure that you spell things correctly. We will look at each line of the program in turn.

3.2.1 using System;

This is a directive that makes available the classes that make up the .NET Framework class library. One of the classes in this namespace is the Console class. This class provides communication within a DOS box or command line session.

3.2.2 class HelloWorld

C# is an object-oriented language. We create a class called HelloWorld that contains one method, Main.

3.2.3 { and }

Braces are used to organize programs in C#. The outer set of braces surround the class HelloWorld. The inner set surround the Main method. We use indentation to highlight the structure of our C# programs.

3.2.4 static int Main()

This is the starting point for our program. It is a method within the HelloWorld class with the following characteristics: it is static (it is not associated with an instance of an object) and it returns an integer value.

3.2.5 Console.WriteLine("Hello World");

Console.WriteLine is a class within System that displays text to the screen or console.

3.2.6 return 0;

It is a tradition within the C family of languages to return a value of zero when things work.

3.2.7 ;

The semicolon is a statement terminator.

3.3 Hello World Using Developer Studio

This example takes you through programming in C# using Developer Studio. The same example is used, as the emphasis is on getting familiar with Developer Studio.

There are screenshots taking you through all of the steps. These are from a Windows XP Professional system. There will be slight variations with other versions of Windows.

3.3.1 Starting Developer Studio

From Windows, click on Start->Microsoft Visual Studio .Net->Microsoft Visual Studio .Net

Figure 3.1 *Starting Developer Studio.*

This should bring up a screen similar to the one below.

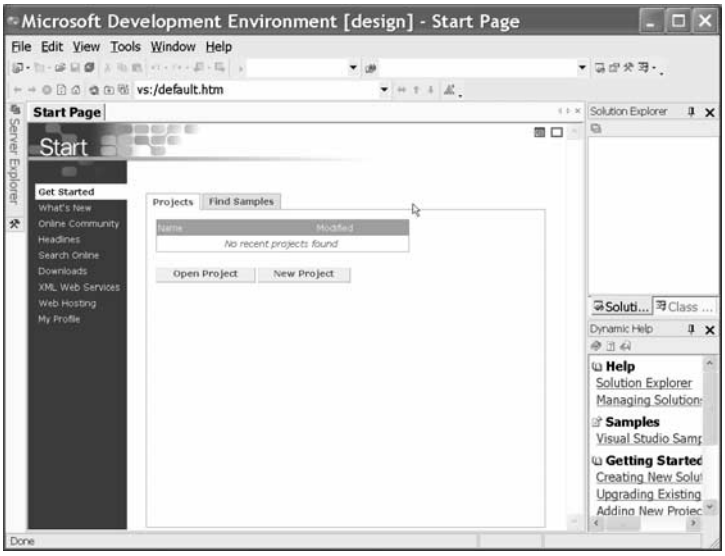


Figure 3.2 *The Developer Studio opening screen.*

3.3.2 Creating a New Project

Click on File->New->Project.

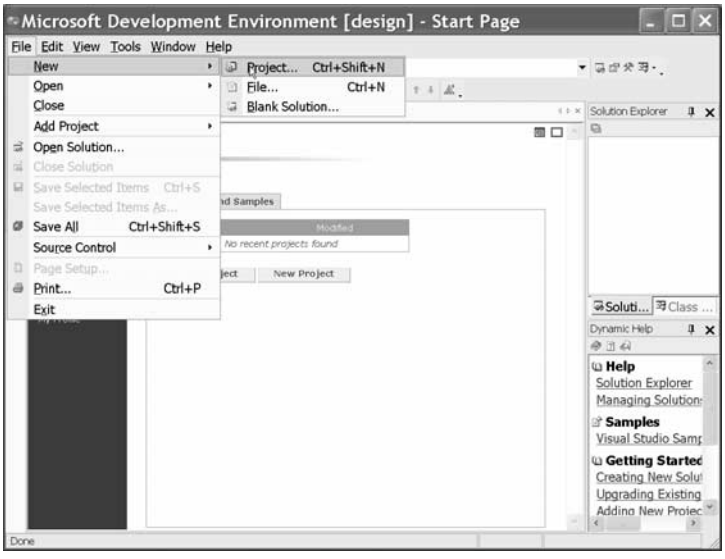


Figure 3.3 *Creating a new project.*

This should bring up a screen similar to the one below.

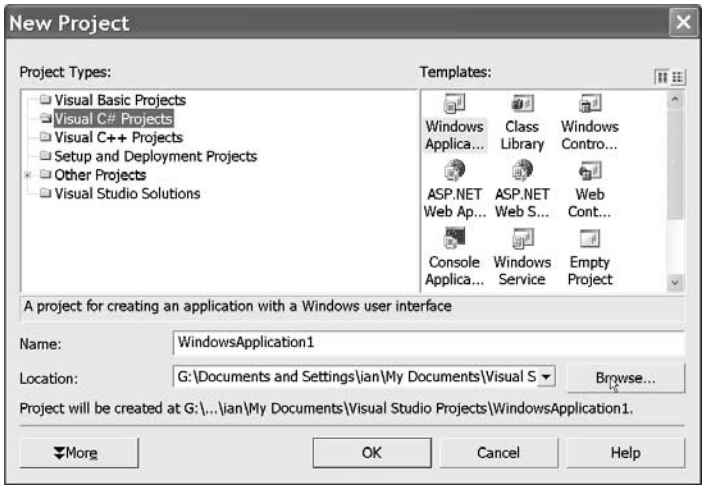


Figure 3.4 *Developer Studio new project screen.*

The key features are the type of project, the template, the project name and the project location.

3.3.3 Creating an Empty Project

Select the Empty Project template.

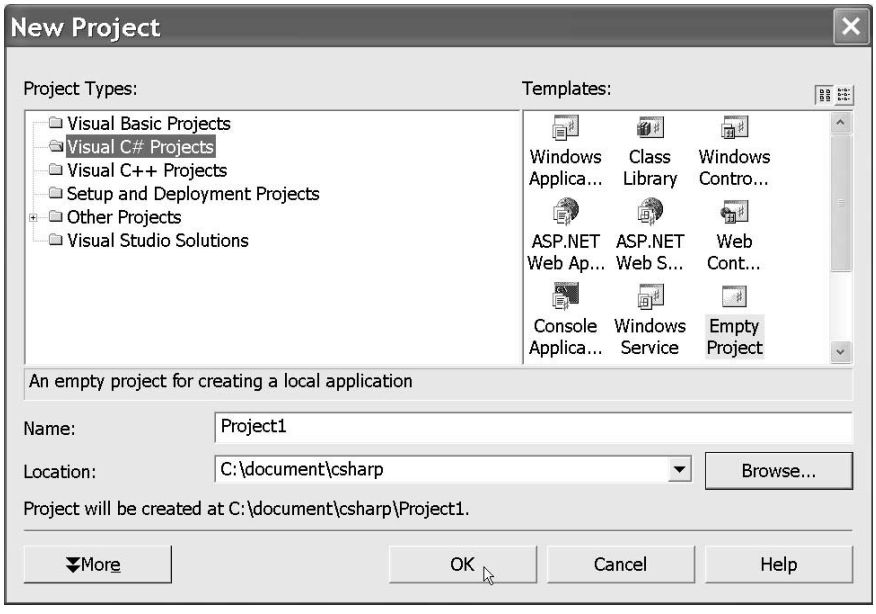


Figure 3.5 *Creating an empty project.*

This should bring up a screen similar to the following.

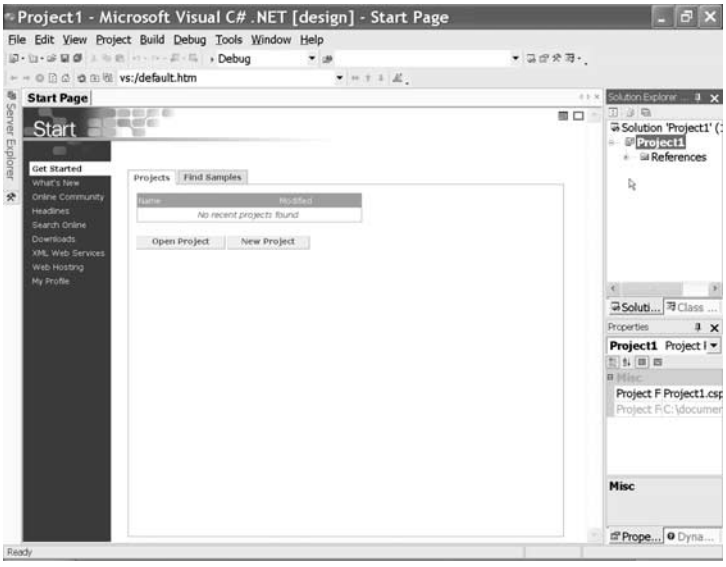


Figure 3.6 *Project start page.*

3.3.4 Adding an Existing Item

Click on Project->Add Existing Item:

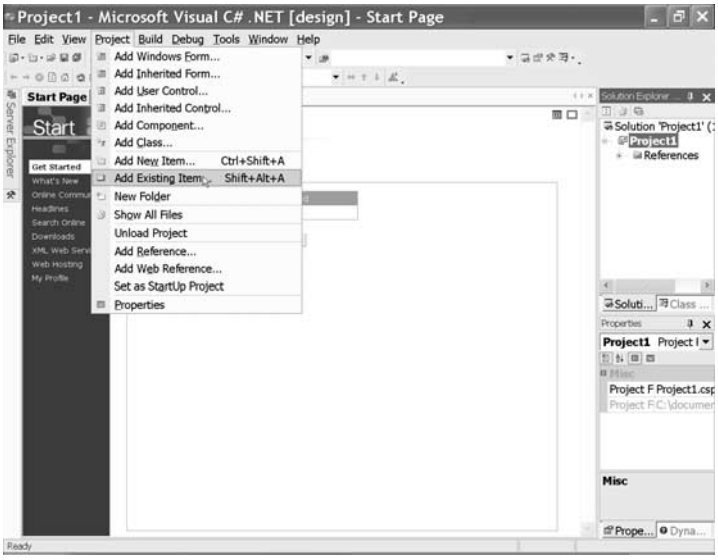


Figure 3.7 *Selecting the add existing item option.*

This should bring up a screen similar to the following.

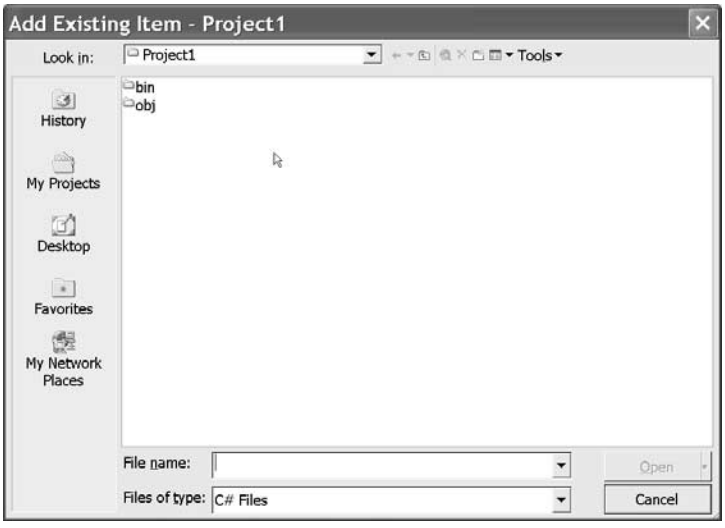


Figure 3.8 *Selecting a file.*

Copy your existing "Hello world" C# program into the project directory. You should end up with a screen similar to the one below:

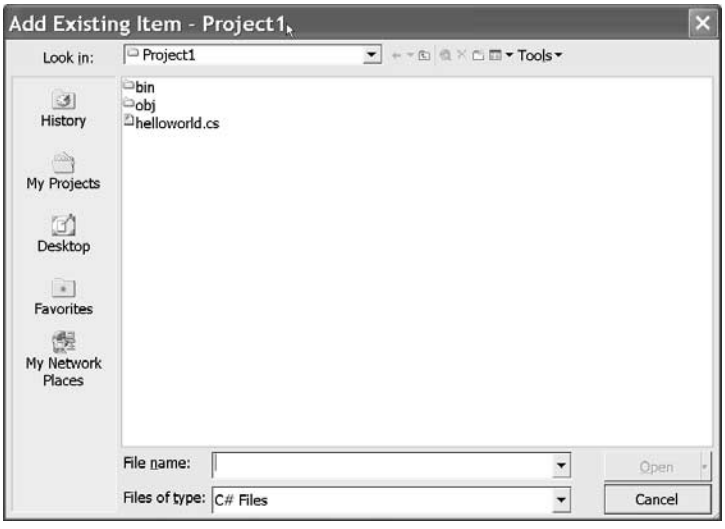


Figure 3.9 *Including the "Hello world" program.*

3.3.5 Building the Project

Now click on Build->Build project.

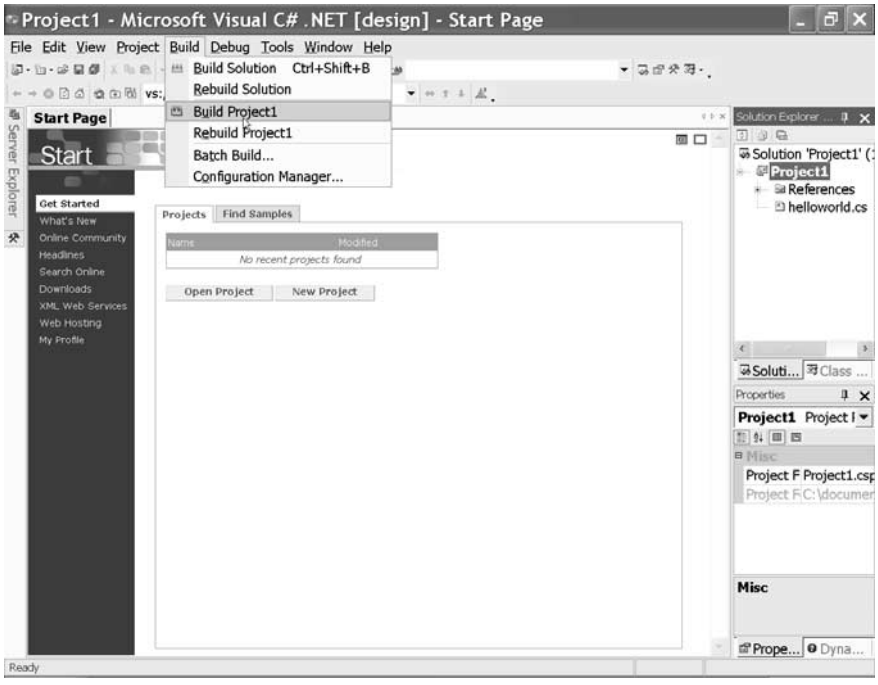


Figure 3.10 *Building the project.*

You should end up with a screen similar to the following, with an output window at the bottom.

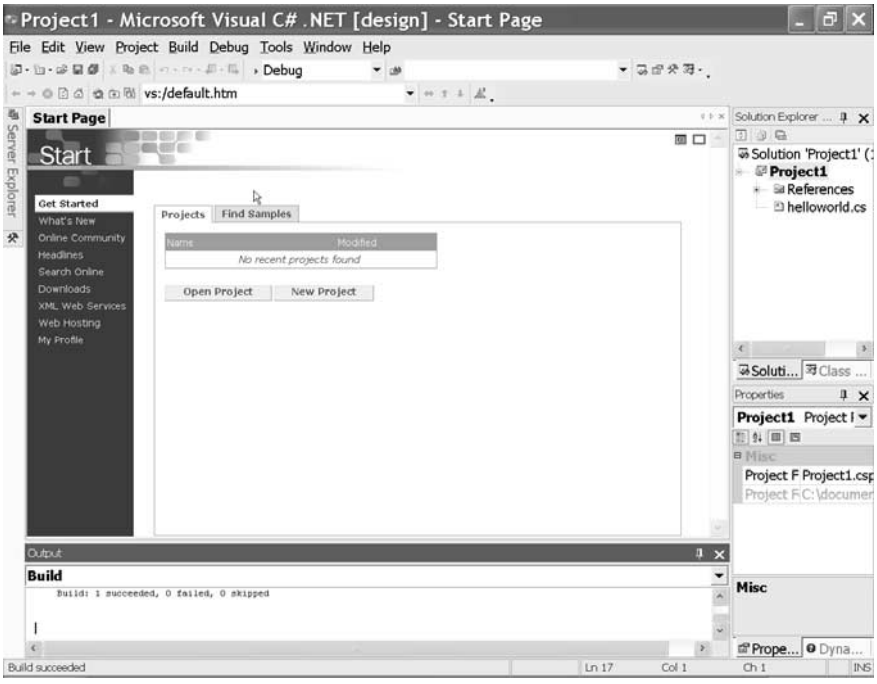


Figure 3.11 *A successful build.*

This output window contains the compilation and build messages. In this case it tells us that the build was successful.

3.3.6 Running the Project

Now click on Debug->Start:

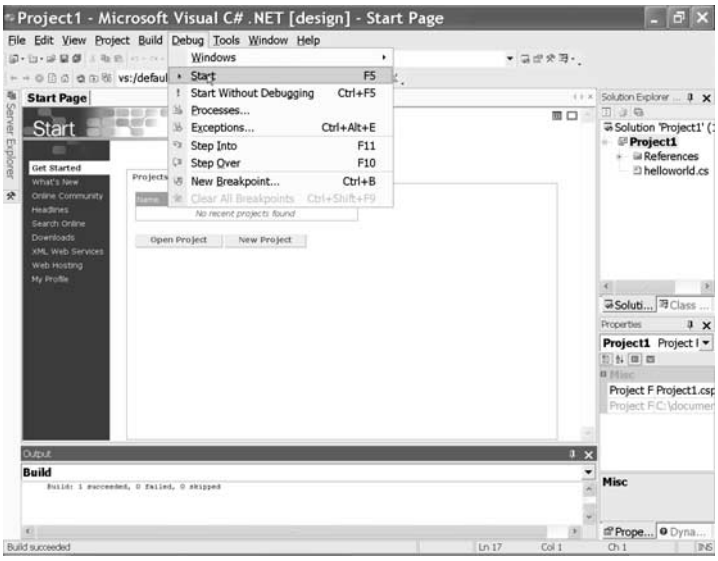


Figure 3.12 *Running the project.*

This should bring up the following DOS window (console session):

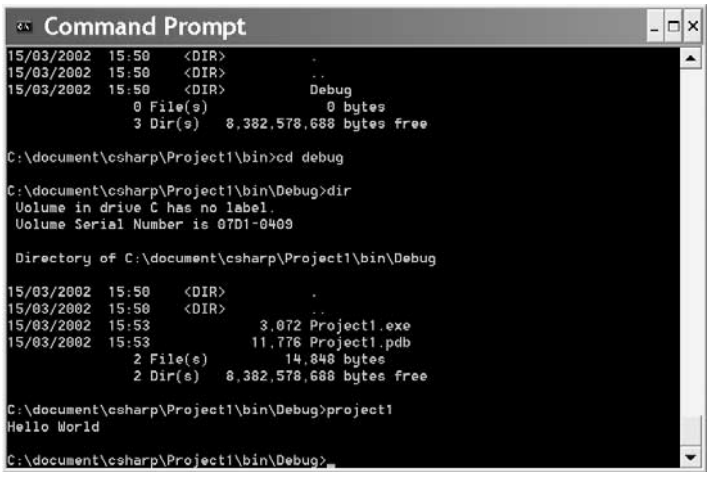


Figure 3.13 *The DOS window.*

Pressing the return key should bring us back to Developer Studio.

3.3.7 Editing the Project

If the following screen does not appear automatically, click on the C# source file in the Solution Explorer window. The key feature is the C# source highlighting.

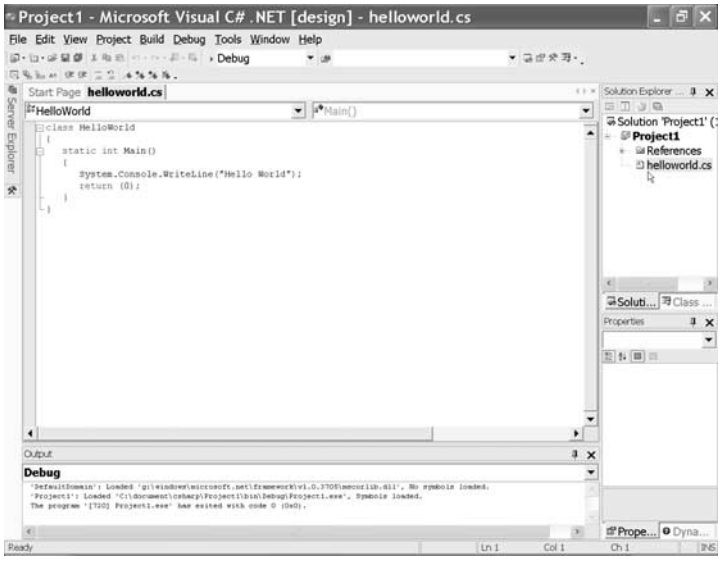


Figure 3.14 *Editing the project.*

3.4 Console Application

In the next example, we let the Developer Studio environment create a skeleton C# program (a console application). Create a new project, but this time select the Console Application template. This is the skeleton C# source file created:

```
namespace ch02200
{
     using System;
     /// <summary>
     ///     Summary description for Class1.
```

```
    /// </summary>
    public class Class1
    {
        public Class1()
        {
            //
            // TODO: Add Constructor Logic here
            //
        }
        public static int Main(string[] args)
        {
            //
            // TODO: Add code to start application here
            //
            return 0;
        }
    }
}
```

Where it says

```
// TODO: Add code to start application here
```

we need to add the following line:

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

We can then build and run the program. A screen opens, the text appears and then it closes. Now open a DOS prompt and go to the project directory. In my case this is:

```
c:\document\csharp\helloworld
```

Type:

```
dir /s /p
```

and look in the bin/debug directory. You can now run the program from the DOS prompt by typing:

```
helloworld
```

We will now look at two more examples from the DOS prompt, rather than within the Developer Studio environment.

3.5 Simple Text I/O

This example reads and writes a line of text.

```
using System;
class c0302
{

   static int Main()
   {
     string line;
     Console.WriteLine(" Type in a line ");
     line=Console.ReadLine();
     Console.WriteLine(line);
     return 0;
   }
}
```

Compile and run the program. It will read in a complete line of text and then echo it back to the screen. `line` is a variable of type string, which is one of the built-in C# data types.

```
line=Console.ReadLine();
```

In an object-oriented programming language, we use a method (ReadLine) to access the data we want and assign it to a variable (line). We will look at strings in more depth in a later chapter.

3.6 Simple Numeric I/O

This example reads in three numbers and prints out their sum.

```
using System;
class c0303
{
   static int Main()
   {
     float [] x = new float [3];
```

```
    float sum=(float)0;
    string line;
    Console.Write(" Type in three numbers");
    Console.WriteLine(", one per line");
    for (int i=0;i<x.Length;i++)
    {
       line=System.Console.ReadLine();
       x[i]=(float)double.Parse(line);
       sum+=x[i];
    }
    Console.WriteLine(" sum is {0:F} ",sum);
    return 0;
  }
}
```

Let us look at each line in turn.

```
float [] x = new float [3];
```

This declares the variable x to be an array of type float (a single-precision data type in C#). As we are working with an object-oriented language we need to create the array object explicitly using new. The array can hold three floats.

```
float sum=(float)0;
```

This declares sum to be a variable of type float and gives it an initial value of zero. By default, numbers in C# are of type double. We need to explicitly cast from one data type to another as C# is a strongly-typed language.

```
string line;
```

This declares line to be a string variable, which we will use to interact with the DOS window.

```
Console.Write(" Type in three numbers");
Console.WriteLine(", one per line");
```

This prints a simple text message to indicate what the user should type.

```
for (int i=0;i<x.Length;i++)
```

This is a simple for-loop in C#. The integer variable i only exists within the scope of the for-loop. Arrays in C# start at 0, i.e. for an array of size three the index goes from 0 to 2. As arrays are objects in C# they have a size associated with them and we access their size using the Length property. This prevents one of the most common programming errors in C and C++ of going outside the array.

```
{
```

We use the braces to enable the for loop to execute multiple statements.

```
line=System.Console.ReadLine();
```

This reads one number at a time, as text, into the line variable.

```
x[i]=(float)double.Parse(line);
```

This statement parses `line`, extracts the number and converts from double precision to single precision.

```
sum+=x[i];
```

This increments sum by the value of x[i] (it is equivalent to sum=sum + x[i]). We will look at expressions in C# in much more detail in a later chapter.

```
}
```

This brace ends the loop.

```
Console.WriteLine(" sum is {0:F} ",sum);
```

This writes out the value of sum and formats it as a floating point number. We will look at formatting in more depth in a later chapter.

```
return 0;
```

This ends the program.

3.7 Online Documentation

Whilst you can access the online help and documentation from within the Developer Studio environment I've found it more useful to do this using the separate access mechanism provided by Microsoft. The following screenshot shows how to do this:

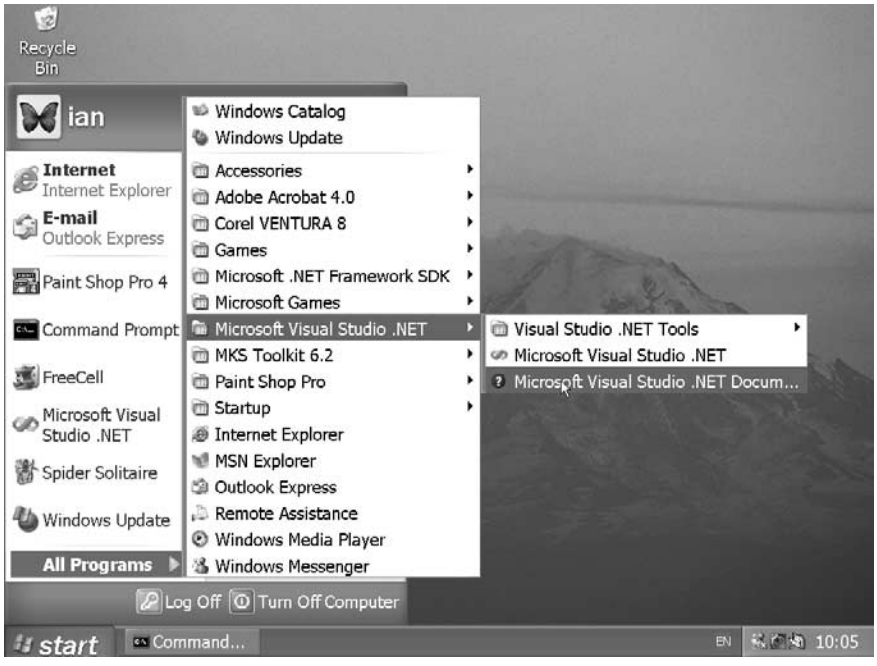


Figure 3.15 *Accessing the online documentation.*

The following screen shot shows the opening screen.

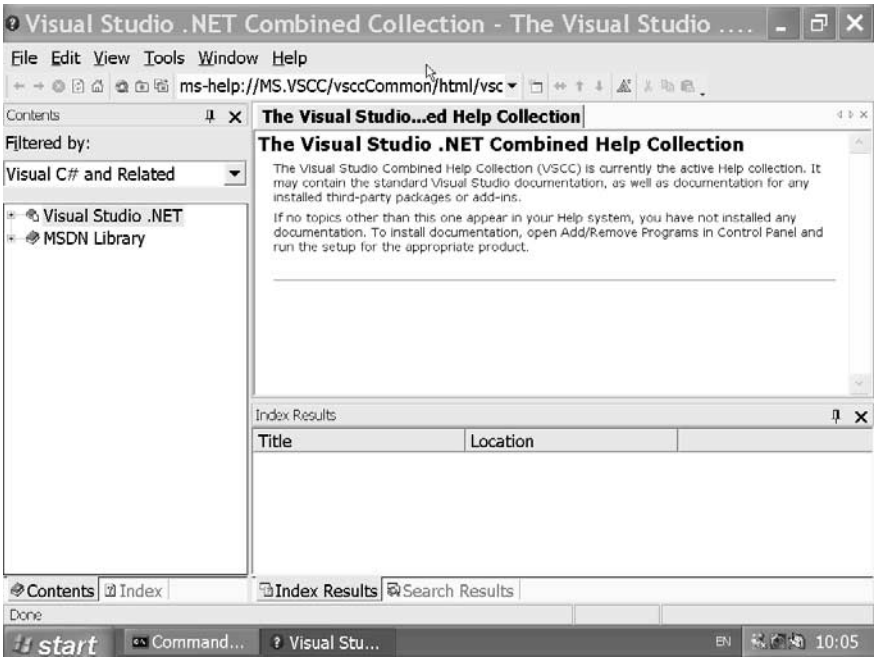


Figure 3.16 *The online documentation opening screen.*

The key points are the mode in which the screen comes up (Contents or Index mode) and the details of how the information is filtered.

The following screenshot shows what is available under the Help menu. The key features are the first two options, Contents mode and Index mode.

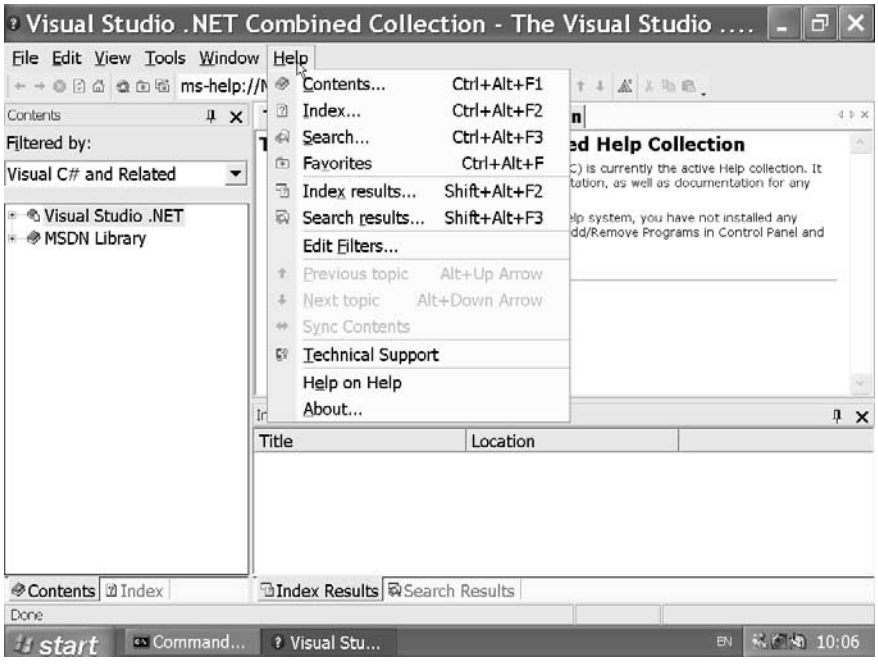


Figure 3.17 *The Help menu.*

3.7.1 Contents Mode

In this case, we are in Contents mode, filtered by Visual C# and Related.

Click on Visual C# and related material to see the other filtering options:

Figure 3.18 *The filtering options.*

Leave Visual C# and related material as the filter. The next thing to do is expand the details. Click on the + sign to the left of the first entry, Visual Studio .Net. This should bring up a screen similar to the following.

Figure 3.19 *The detailed contents list.*

Next click on Visual Basic and Visual C#.

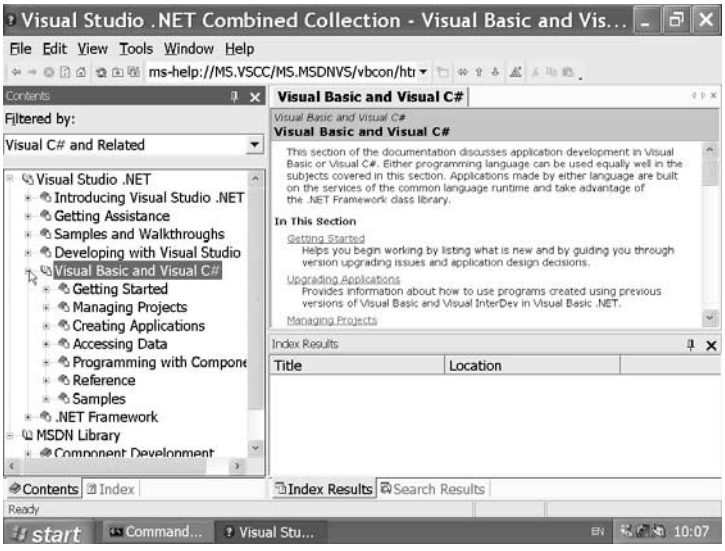


Figure 3.20 *Visual Basic and Visual C# contents.*

This should bring up a screen similar to the following.

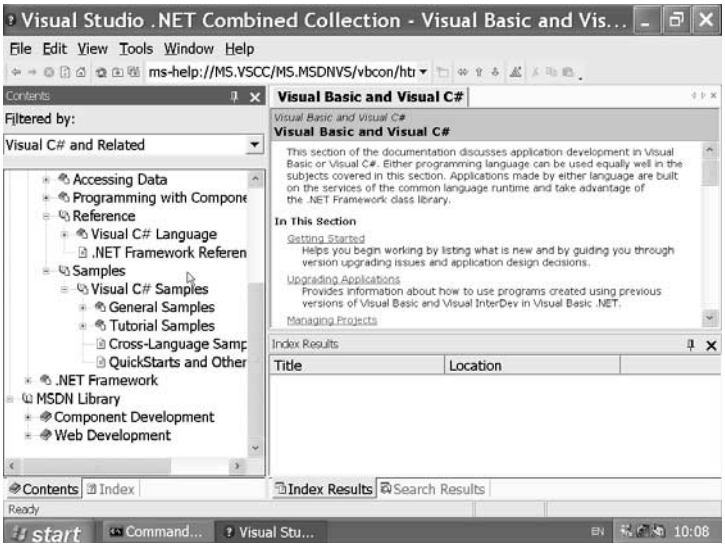


Figure 3.21 *The Visual Basic and Visual C# help introduction.*

Experiment with expanding and contracting the details to see what is available. Microsoft provides a lot of material and it will pay dividends to get familiar with what is in the online help. It would not have been possible to write the examples in this book without reference to this material.

3.7.2 Compiler Error Messages

If you get a compiler error message you can get quite a lot of additional information about what might be the problem. Put the help system into Index mode and type in the compiler error message number.

The following screen shot illustrates the additional information provided, over the rather terse one line compiler error message. There is also a sample program that will generate this error message.

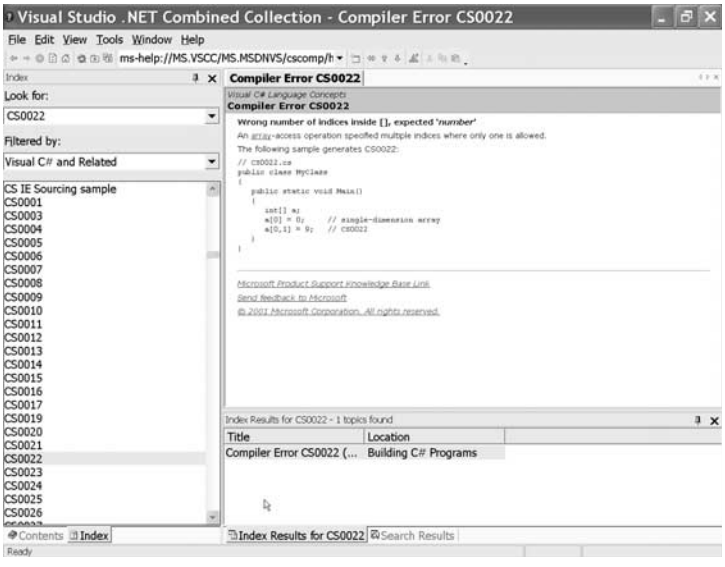


Figure 3.22 *Compiler error CS0022.*

3.8 Key Concepts of C#

C# has a similar syntax to the C, C++ and Java family of languages.

C# is object-oriented.

C# programs are organized using classes.

C# program execution starts with the Main method.

C# is a strongly-typed language and casting is required to convert between types.

3.9 Summary

The examples in this chapter were chosen to highlight some of the key concepts of C#. Compile and run them to gain familiarity with C# and the development environment.