1

Programming with Visual C++ 2005

COPYRIGHTED MATERIAL

Windows programming isn't difficult. In fact, Microsoft Visual C++ 2005 makes it remarkably easy, as you'll see throughout the course of this book. There's just one obstacle in your path: Before you get to the specifics of Windows programming, you have to be thoroughly familiar with the capabilities of the C++ programming language, particularly the object-oriented aspects of the language. Object-oriented techniques are central to the effectiveness of all the tools that are provided by Visual C++ 2005 for Windows programming, so it's essential that you gain a good understanding of them. That's exactly what this book provides.

This chapter gives you an overview of the essential concepts involved in programming applications in C++. You'll take a rapid tour of the Integrated Development Environment (IDE) that comes with Visual C++ 2005. The IDE is straightforward and generally intuitive in its operation, so you'll be able to pick up most of it as you go along. The best approach to getting familiar with it is to work through the process of creating, compiling, and executing a simple program. By the end of this chapter, you will have learned:

- What the principal components of Visual C++ 2005 are
- What the .NET Framework consists of and the advantages it offers
- What solutions and projects are and how you create them
- About console programs
- How to create and edit a program
- How to compile, link, and execute C++ console programs
- How to create and execute basic Windows programs

So power up your PC, start Windows, load the mighty Visual C++ 2005, and begin your journey.

The .NET Framework

The .NET Framework is a central concept in Visual C++ 2005 as well as in all the other .NET development products from Microsoft. The .NET Framework consists of two elements: the **Common Language Runtime** (CLR) in which your application executes, and a set of libraries called the .NET Framework class libraries. The .NET Framework class libraries provide the functional support your code will need when executing with the CLR, regardless of the programming language used, so .NET programs written in C++, C#, or any of the other languages that support the .NET Framework all use the same .NET libraries.

There are two fundamentally different kinds of C++ applications you can develop with Visual C++ 2005. You can write applications that natively execute on your computer. These applications will be referred to as **native C++ programs**. You write native C++ programs in the version of C++ that is defined by the ISO/ANSI language standard. You can also write applications to run under the control of the CLR in an extended version of C++ called **C++/CLI**. These programs will be referred to as **CLR programs**, or **C++/CLI programs**.

The .NET Framework is not strictly part of Visual C++ 2005 but rather a component of the Windows operating system that makes it easier to build software applications and Web services. The .NET Framework offers substantial advantages in code reliability and security, as well as the ability to integrate your C++ code with code written in over 20 other programming languages that target the .NET Framework. A slight disadvantage of targeting the .NET Framework is that there is a small performance penalty, but you won't notice this in the majority of circumstances.

The Common Language Runtime (CLR)

The CLR is a standardized environment for the execution of programs written in a wide range of high-level languages including Visual Basic, C#, and of course C++. The specification of the CLR is now embodied in the European Computer Manufacturers (ECMA) standard for the **Common Language Infrastructure** (CLI), ECMA-335, and also in the equivalent ISO standard, ISO/IEC 23271, so the CLR is an implementation of this standard. You can see why C++ for the CLR is referred to as C++/CLI — it's C++ for the Common Language Infrastructure, so you are likely to see C++/CLI compilers on other operating systems that implement the CLI.

> *Note that information on all ECMA standards is available from* `http://www.ecma-international.org` *and ECMA-335 is currently available as a free download.*

The CLI is essentially a specification for a **virtual machine** environment that enables applications written in diverse high-level programming languages to be executed in different system environments without changing or recompiling the original source code. The CLI specifies a standard **intermediate** language for the virtual machine to which the high-level language source code is compiled. With the .NET Framework, this intermediate language is referred to as **M**icro**s**oft **I**ntermediate **L**anguage **(MSIL**). Code in the intermediate language is ultimately mapped to machine code by a just-in-time (JIT) compiler when you execute a program. Of course, code in the CLI intermediate language can be executed within any other environment that has a CLI implementation.

The CLI also defines a common set of data types called the **Common Type System** (**CTS**) that should be used for programs written in any programming language targeting a CLI implementation. The CTS specifies how data types are used within the CLR and includes a set of predefined types. You may also define your own data types, and these must be defined in a particular way to be consistent with the CLR, as you'll see. Having a standardized type system for representing data allows components written in different programming languages to handle data in a uniform way and makes it possible to integrate components written in different languages to be integrated into a single application.

Data security and program reliability is greatly enhanced by the CLR, in part because dynamic memory allocation and release for data is fully automatic but also because the MSIL code for a program is comprehensively checked and validated before the program executes. The CLR is just one implementation of the CLI specification that executes under Microsoft Windows on a PC; there will undoubtedly be other implementations of the CLI for other operating system environments and hardware platforms. You'll sometimes find that the terms CLI and CLR used interchangeably, although it should be evident that they are not the same things. The CLI is a standard specification; the CLR is Microsoft's implementation of the CLI.

Writing C++ Applications

You have tremendous flexibility in the types of applications and program components that you can develop with Visual C++ 2005. As noted earlier in this chapter, you have two basic options for Windows applications: you can write code that executes with the CLR, and you can also write code that compiles directly to machine code and thus executes natively. For window-based applications targeting the CLR, you use Windows Forms as the base for the GUI provided by the .NET Framework libraries. Using Windows Forms enables rapid GUI development because you assemble the GUI graphically from standard components and have the code generated completely automatically. You then just need to customize the code that has been generated to provide the functionality you require.

For natively executing code, you have several ways to go. One possibility is to use the Microsoft Foundation Classes (MFC) for programming the graphical user interface for your Windows application. The MFC encapsulates the Windows operating system Application Programming Interface (API) for GUI creation and control and greatly eases the process of program development. The Windows API originated long before the C++ language arrived on the scene so it has none of the object-oriented characteristics that would be expected if it were written today; however, you are not obliged to use the MFC. If you want the ultimate in performance, you can write your C++ code to access the Windows API directly.

C++ code that executes with the CLR is described as **managed C++** because data and code is managed by the CLR. In CLR programs, the release of memory that you have allocated dynamically for storing data is taken care of automatically, thus eliminating a common source of error in native C++ applications. C++ code that executes outside of the CLR is sometimes described by Microsoft as **unmanaged C++** because the CLR is not involved in its execution. With unmanaged C++ you must take care of all aspects allocating and releasing memory during execution of your program yourself, and you also forego the enhanced security provided by the CLR. You'll also see unmanaged C++ referred to as **native C++** because it compiles directly to native machine code.

Figure 1-1 shows the basic options you have for developing C++ applications.

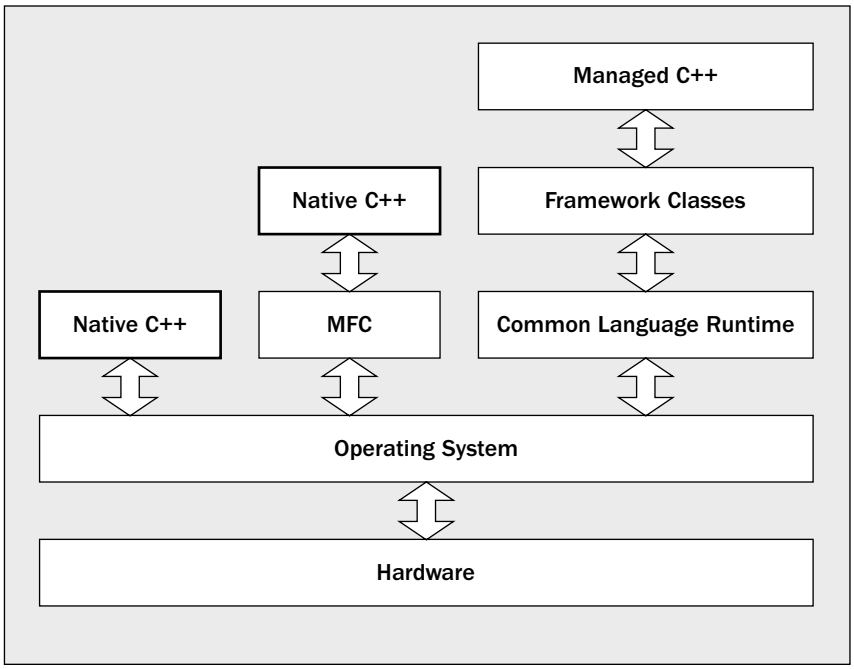

Figure 1-1

Figure 1-1 is not the whole story. An application can consist partly of managed C++ and partly of native C++, so you are not obliged to stick to one environment or the other. Of course, you do lose out somewhat by mixing the code, so you would choose to follow this approach only when necessary, such as when you want to convert an existing native C++ application to run with the CLR. You obviously won't get the benefits inherent in managed C++ in the native C++ code, and there can also be appreciable overhead involved in communications between the managed and unmanaged code components. The ability to mix managed and unmanaged code can be invaluable, however, when you need to develop or extend existing unmanaged code but also want to obtain the advantages of using the CLR. Of course, for new applications you should decide whether you want to create it as a managed C++ application at the outset.

Learning Windows Programming

There are always two basic aspects to interactive applications executing under Windows: you need code to create the Graphical User Interface (the GUI) with which the user interacts, and you need code to process these interactions to provide the functionality of the application. Visual C++ 2005 provides you with a great deal of assistance in both aspects of Windows application development. As you'll see later in this chapter, you can create a working Windows program with a GUI without writing any code yourself at all. All the basic code to create the GUI can be generated automatically by Visual C++ 2005; however, it's essential to understand how this automatically generated code works because you need to extend and modify it to make it do what you want, and to do that you need a comprehensive understanding of C++.

For this reason, you'll first learn C++ — both the native C++ and C++/CLI versions of the language — without getting involved in Windows programming considerations. After you're comfortable with C++, you'll learn how you develop fully-fledged Windows applications using native C++ and C++/CLI. This means that while you are learning C++, you'll be working with programs that just involved command line input and output. By sticking to this rather limited input and output capability, you'll be able to concentrate of the specifics of how the C++ language works and avoid the inevitable complications involved in GUI building and control. After you become comfortable with C++, you'll find that it's an easy and natural progression to applying C++ to the development of Windows application programs.

Learning C++

Visual C++ 2005 fully supports two versions of C++ defined by two separate standards:

- The ISO/ANSI C++ standard is for implementing native applications — unmanaged C++. This version of C++ is supported on the majority of computer platforms.
- The C++/CLI standard is designed specifically for writing programs that target the CLR and is an extension to the ISO/ANSI C++.

Chapters 2 through 10 of this book teach you the C++ language. Because C++/CLI is an extension of ISO/ANSI C++, the first part of each chapter introduces elements of the ISO/ANSI C++ language; the second part explains the additional features that C++/CLI introduces.

Writing programs in C++/CLI allows you to take full advantage of the capabilities of the .NET Framework, something that is not possible with programs written in ISO/ANSI C++. Although C++/CLI is an extension of ANSI/ISO C++, to be able to execute your program fully with the CLR means that it must conform to the requirements of the CLR. This implies that there are some features of ANSI/ISO C++ that you cannot use in your CLR programs. One example of this that you might deduce from what I have said up to now is that the dynamic memory allocation and release facilities offered by ISO/ANSI C++ are not compatible with the CLR; you must use the CLR mechanism for memory management and this implies that you must use C++/CLI classes, not native C++ classes.

The C++ Standards

The ISO/ANSI standard is defined by the document ISO/IEC 14882 that is published by the American National Standards Institute (ANSI). ISO/ANSI standard C++ is the well-established version of C++ that has been around since 1998 and is supported by compilers on the majority of computer hardware platforms and operating systems. Programs that you write in ISO/ANSI C++ can be ported from one system environment to another reasonably easily, although the library functions that a program uses — particularly those related to building a graphical user interface — are a major determinant of how easy or difficult it will be. ISO/ANSI standard C++ has been the first choice of many professional program developers because it is so widely supported, and because is one of the most powerful programming languages available today.

The ISO/ANSI standard for C++ can be purchased from `http://www.iso.org`.

C++/CLI is a version of C++ that extends the ISO/ANSI standard for C++ to better support the Common Language Infrastructure (CLI) that is defined by the standard ECMA-355. The first draft of this standard appeared in 2003 and was developed from an initial technical specification that was produced by Microsoft to support the execution of C++ programs with the .NET Framework. Thus both the CLI and C++/CLI were originated by Microsoft in support of the .NET Framework. Of course, standardizing

the CLI and C++/CLI greatly increases the likelihood of implementations in environments other than Windows. It's important to appreciate that although C++/CLI is an extension of ISO/ANSI C++, there are features of ISO/ANSI C++ that you must not use when you want your program to execute fully under the control of the CLR. You'll learn what these are as you progress through the book.

The CLR offers substantial advantages over the native environment. By targeting your C++ programs at the CLR, your programs will be more secure and not prone to the potential errors you can make when using the full power of ISO/ANSI C++. The CLR also removes the incompatibilities introduced by various high-level languages by standardizing the target environment to which they are compiled and thus permit modules written in C++ to be combined with modules written in other languages such as C# or Visual Basic.

Console Applications

As well as developing Windows applications, Visual C++ 2005 also allows you to write, compile, and test C++ programs that have none of the baggage required for Windows programs — that is, applications that are essentially character-based, command-line programs. These programs are called **console applications** in Visual C++ 2005 because you communicate with them through the keyboard and the screen in character mode.

Writing console applications might seem as though you are being sidetracked from the main objective of Windows programming, but when it comes to learning C++ (which you do need to do before embarking on Windows-specific programming), it's the best way to proceed. There's a lot of code in even a simple Windows program, and it's very important not to be distracted by the complexities of Windows when learning the ins and outs of C++. Therefore, in the early chapters of the book where you are concerned with how C++ works, you'll spend time walking with a few lightweight console applications before you get to run with the heavyweight sacks of code in the world of Windows.

While you're learning C++, you'll be able to concentrate on the language features without worrying about the environment in which you're operating. With the console applications that you'll write, you have only a text interface, but this will be quite sufficient for understanding all of C++ because there's no graphical capability within the definition of the language. Naturally, I will provide extensive coverage of graphical user interface programming when you come to write programs specifically for Windows using Microsoft Foundation Classes (MFC) in native C++ applications and Windows Forms with the CLR.

There are two distinct kinds of console applications and you'll be using both. **Win32 console applications** compile to native code, and you'll be using these to try out the capabilities of ISO/ANSI C++. **CLR console applications** target the CLR so you'll be using these when you are working with the features of C++/CLI.

Windows Programming Concepts

Our approach to Windows programming is to use all the tools that Visual C++ 2005 provides. The project creation facilities that are provided with Visual C++ 2005 can generate skeleton code for a wide variety of application programs automatically, including basic Windows programs. Creating a project is the starting point for all applications and components that you develop with Visual C++ 2005, and to get a

flavor of how this works, you'll look at the mechanics of creating some examples, including an outline Windows program, later in this chapter.

A Windows program has a different structure from that of the typical console program you execute from the command line, and it's more complicated. In a console program, you can get input from the keyboard and write output back to the command line directly, whereas a Windows program can access the input and output facilities of the computer only by way of functions supplied by the Windows operating system; no direct access to the hardware resources is permitted. Because several programs can be active at one time under Windows, Windows has to determine which application a given raw input such as a mouse click or the pressing of a key on the keyboard is destined for and signal the program concerned accordingly. Thus the Windows operating system has primary control of all communications with the user.

Also, the nature of the interface between a user and a Windows application is such that a wide range of different inputs is usually possible at any given time. A user may select any of a number of menu options, click a toolbar button, or click the mouse somewhere in the application window. A well-designed Windows application has to be prepared to deal with any of the possible types of input at any time because there is no way of knowing in advance which type of input is going to occur. These user actions are received by the operating system in the first instance and are all regarded by Windows as **events**. An event that originates with the user interface for your application will typically result in a particular piece of your program code being executed. How program execution proceeds is therefore determined by the sequence of user actions. Programs that operate in this way are referred to as **event-driven programs** and are different from traditional procedural programs that have a single order of execution. Input to a procedural program is controlled by the program code and can occur only when the program permits it; therefore, a Windows program consists primarily of pieces of code that respond to events caused by the action of the user, or by Windows itself. This sort of program structure is illustrated in Figure 1-2.

Each square block in Figure 1-2 represents a piece of code written specifically to deal with a particular event. The program may appear to be somewhat fragmented because of the number of disjointed blocks of code, but the primary factor welding the program into a whole is the Windows operating system itself. You can think of your program as customizing Windows to provide a particular set of capabilities.

Of course, the modules servicing various external events, such as selecting a menu or clicking the mouse, all typically have access to a common set of application-specific data in a particular program. This application data contains information that relates to what the program is about — for example, blocks of text in an editor or player scoring records in a program aimed at tracking how your baseball team is doing — as well as information about some of the events that have occurred during execution of the program. This shared collection of data allows various parts of the program that look independent to communicate and operate in a coordinated and integrated fashion. I will go into this in much more detail later in the book.

Even an elementary Windows program involves several lines of code, and with Windows programs that are generated by the application wizards that come with Visual C++ 2005, "several" turns out to be "many." To simplify process of understanding how C++ works, you need a context that is as uncomplicated as possible. Fortunately, Visual C++ 2005 comes with an environment that is ready-made for the purpose.

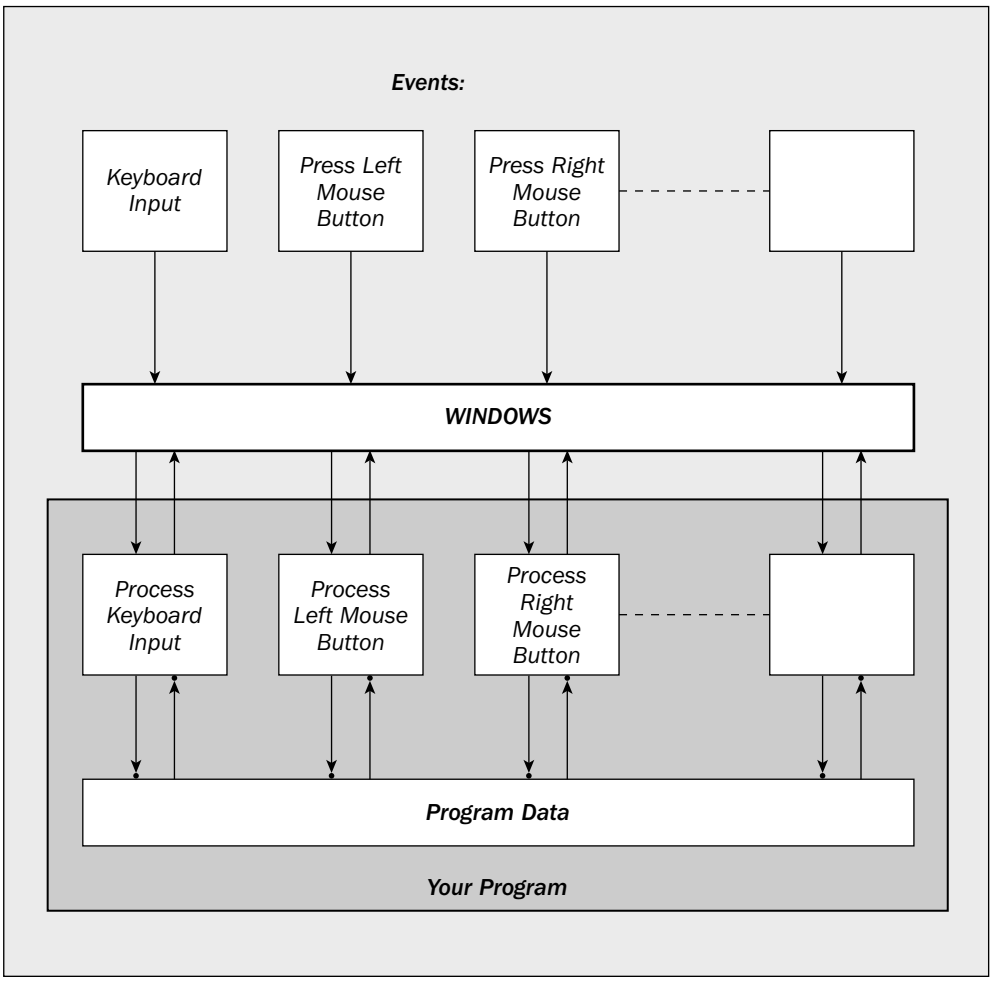

Figure 1-2

What Is the Integrated Development Environment?

The Integrated Development Environment (IDE) that comes with Visual C++ 2005 is a completely self-contained environment for creating, compiling, linking, and testing your C++ programs. It also happens to be a great environment in which to learn C++ (particularly when combined with a great book).

Visual C++ 2005 incorporates a range of fully integrated tools designed to make the whole process of writing C++ programs easy. You will see something of these in this chapter, but rather than grind through a boring litany of features and options in the abstract, first take a look at the basics to get a view of how the IDE works and then pick up the rest in context as you go along.

Components of the System

The fundamental parts of Visual C++ 2005, provided as part of the IDE, are the editor, the compiler, the linker, and the libraries. These are the basic tools that are essential to writing and executing a C++ program. Their functions are as follows.

The Editor

The editor provides an interactive environment for you to create and edit C++ source code. As well as the usual facilities, such as cut and paste, which you are certainly already familiar with, the editor also provides color cues to differentiate between various language elements. The editor automatically recognizes fundamental words in the C++ language and assigns a color to them according to what they are. This not only helps to make your code more readable but also provides a clear indicator of when you make errors in keying such words.

The Compiler

The compiler converts your source code into **object code,** and detects and reports errors in the compilation process. The compiler can detect a wide range of errors that are due to invalid or unrecognized program code, as well as structural errors, where, for example, part of a program can never be executed. The object code output from the compiler is stored in files called **object files**. There are two types of object code that the compiler produces. These object codes usually have names with the extension `.obj`.

The Linker

The linker combines the various modules generated by the compiler from source code files, adds required code modules from program libraries supplied as part of C++, and welds everything into an executable whole. The linker can also detect and report errors — for example, if part of your program is missing or a non-existent library component is referenced.

The Libraries

A **library** is simply a collection of pre-written routines that supports and extends the C++ language by providing standard professionally produced code units that you can incorporate into your programs to carry out common operations. The operations that are implemented by routines in the various libraries provided by Visual C++ 2005 greatly enhance productivity by saving you the effort of writing and testing the code for such operations yourself. I have already mentioned the .NET Framework library, and there are a number of others — too many to enumerate here — but I'll mention the most important ones.

The **Standard C++ Library** defines a basic set of routines common to all ISO/ANSI C++ compilers. It contains a wide range of routines including numerical functions such as calculating square roots and evaluating trigonometrical functions, character and string processing routines such as classifying characters and comparing character strings, and many others. You'll get to know quite a number of these as you develop your knowledge of ISO/ANSI C++. There are also libraries that support the C++/CLI extensions to ISO/ANSI C++.

Native window-based applications are supported by a library called the **Microsoft Foundation Classes** (MFC). The MFC greatly reduces the effort needed to build the graphical user interface for an application. You'll see a lot more of the MFC when you finish exploring the nuances of the C++ language. Another library contains a set of facilities called **Windows Forms** that are roughly the equivalent of the MFC for window-based applications that executed with the .NET Framework. You'll be seeing how you make use of Windows Forms to develop applications, too.

Using the IDE

All program development and execution in this book is performed from within the IDE. When you start Visual C++ 2005, notice an application window similar to that shown in Figure 1-3.

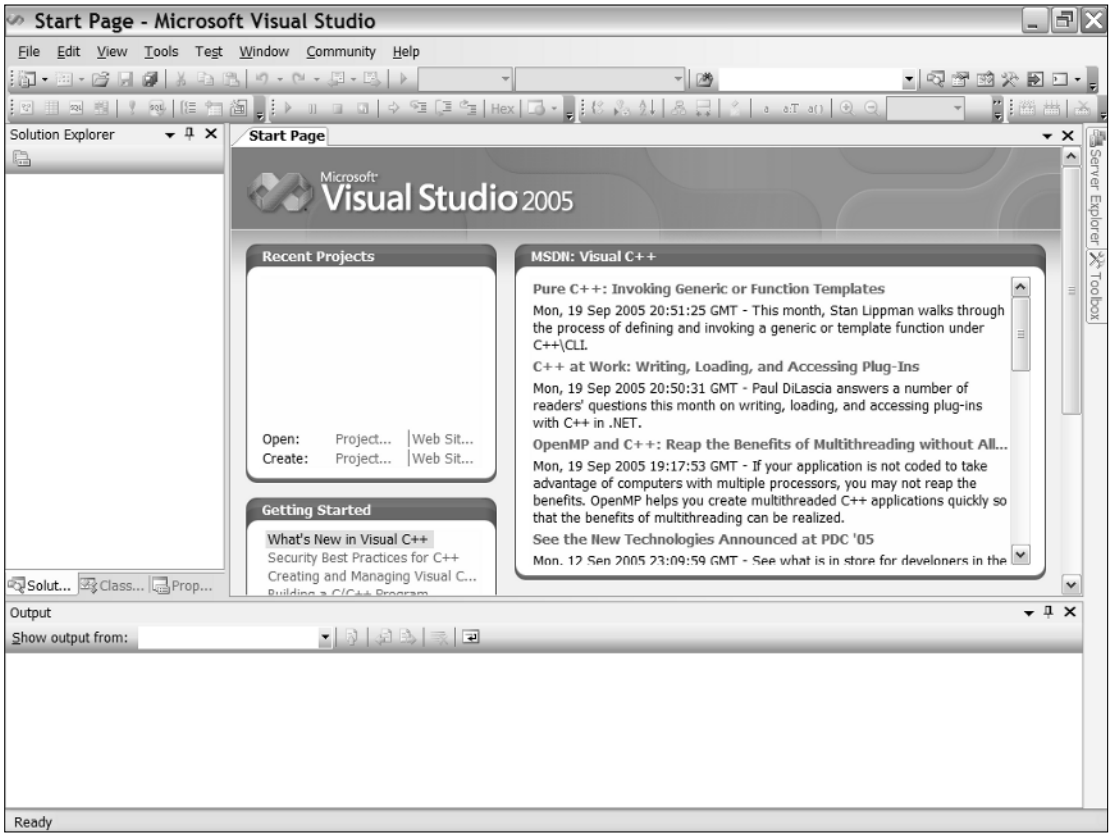

Figure 1-3

The window to the left in Figure 1-3 is the **Solution Explorer window**, the top-right window presently showing the Start page is the **Editor window**, and the window at the bottom is the **Output window**. The Solution Explorer window enables you to navigate through your program files and display their contents in the Editor window and to add new files to your program. The Solution Explorer window has up to three additional tabs (only two are shown in Figure 1-3) that display **Class View**, **Resource View**, and **Property Manager** for your application, and you can select which tabs are to be displayed from the View menu. The Editor window is where you enter and modify source code and other components of your application. The Output window displays messages that result from compiling and linking your program.

Toolbar Options

You can choose which toolbars are displayed in your Visual C++ window by right-clicking in the toolbar area. A pop-up menu with a list of toolbars (Figure 1-4) appears, and the toolbars that are currently displayed have check marks alongside.

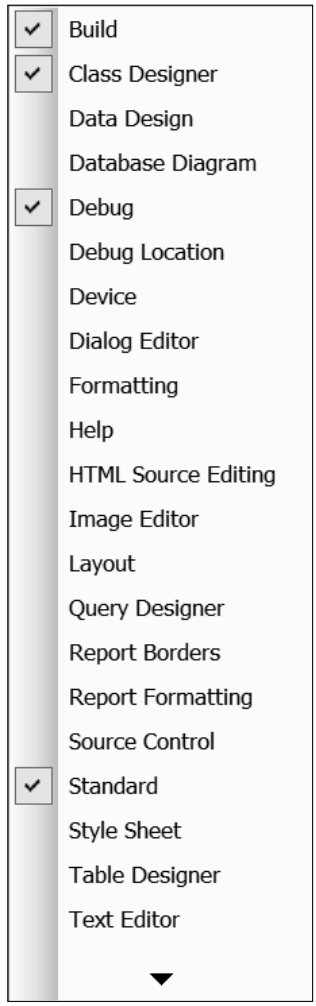

Figure 1-4

This is where you decide which toolbars are visible at any one time. You can make your set of toolbars the same as those shown in Figure 1-3 by making sure the `Build`, `Class Designer`, `Debug`, `Standard`, and `View Designer` menu items are checked. Clicking in the gray area to the left of a toolbar checks it if it unchecked and results in it being displayed; clicking a check mark hides the corresponding toolbar.

You don't need to clutter up the application window with all the toolbars you think you might need at some time. Some toolbars appear automatically when required, so you'll probably find that the default toolbar selections are perfectly adequate most of the time. As you develop your applications, from time to time you might think it would be more convenient to have access to toolbars that aren't displayed. You can change the set of toolbars that are visible whenever it suits you by right-clicking in the toolbar area and choosing from the context menu.

Similar to many other Windows applications, the toolbars that make up Visual C++ 2005 come complete with tooltips. Just let the mouse pointer linger over a toolbar button for a second or two and a white label displays the function of that button.

Dockable Toolbars

A **dockable** toolbar is one that you can drag around with the mouse to position at a convenient place in the window. When it is placed in any of the four borders of the application, it is said to be *docked* and looks similar to the toolbars you see at the top of the application window. The toolbar on the upper line of toolbar buttons that contains the disk icons and the text box to the right of a pair of binoculars is the Standard toolbar. You can drag this away from the toolbar by placing the cursor on it and dragging it with the mouse while you hold down the left mouse button. It then appears as a separate window you can position anywhere.

If you drag any dockable toolbar away from its docked position, it looks like the `Standard` toolbar you see in Figure 1-5, enclosed in a little window — with a different caption. In this state, it is called a **floating toolbar**. All the toolbars that you see in Figure 1-3 are dockable and can be floating, so you can experiment with dragging any of them around. You can position them in docked positions where they revert to their normal toolbar appearance. You can dock a dockable toolbar at any side of the main window.

Figure 1-5

You'll become familiar with many of the toolbar icons that Visual C++ 2005 uses from other Windows applications, but you may not appreciate exactly what these icons do in the context of Visual C++, so I'll describe them as we use them.

Because you'll use a new project for every program you develop, looking at what exactly a project is and understanding how the mechanism for defining a project works is a good place to start finding out about Visual C++ 2005.

Documentation

There will be plenty of occasions when you'll want to find out more information about Visual C++ 2005. The **Microsoft Development Network** (**MSDN**) **Library** provides comprehensive reference material on all the capabilities on Visual C++ 2005 and more besides. When you install Visual C++ 2005 onto your machine, there is an option to install part or all of the MSDN documentation. If you have the disk space available I strongly recommend that you install the MSDN Library.

Press the F1 function to browse the MSDN Library. The Help menu also provides various routes into the documentation. As well as offering reference documentation, the MSDN Library is a useful tool when dealing with errors in your code, as you'll see later in this chapter.

Projects and Solutions

A **project** is a container for all the things that make up a program of some kind — it might be a console program, a window-based program, or some other kind of program, and it usually consists of one or more source files containing your code plus possibly other files containing auxiliary data. All the files for a project are stored in **the project folder** and detailed information about the project is stored in an XML file with the extension `.vcproj` that is also in the project folder. The project folder also contains other folders that are used to store the output from compiling and linking your project.

The idea of a **solution** is expressed by its name, in that it is a mechanism for bringing together all the programs and other resources that represent a solution to a particular data processing problem. For example, a distributed order entry system for a business operation might be composed of several different programs that could each be developed as a project within a single solution; therefore, a **solution** is a folder in which all the information relating to one or more projects is stored, so one or more project folders are subfolders of the solution folder. Information about the projects in a solution is stored in two files with the extensions `.sln` and `.suo`. When you create a project, a new solution is created automatically unless you elect to add the project to an existing solution.

When you create a project along with a solution, you can add further projects to the same solution. You can add any kind of project to an existing solution, but you would usually add only a project that was related in some way to the existing project or projects in the solution. Generally, unless you have a good reason to do otherwise, each of your projects should have its own solution. Each example you create with this book will be a single project within its own solution.

Defining a Project

The first step in writing a Visual C++ 2005 program is to create a project for it using the `File > New > Project` menu option from the main menu or you can press `Ctrl+Shift+N`. As well as containing files that define all the code and any other data that goes to make up your program, the project XML file in the project folder also records the Visual C++ 2005 options you're using. Although you don't need to concern yourself with the project file — it is entirely maintained by the IDE — you can browse it if you want to see what the contents are, but take care not to modify it accidentally.

That's enough introductory stuff for the moment. It's time to get your hands dirty.

Try It Out — Creating a Project for a Win32 Console Application

You'll now take a look at creating a project for a console application. First select `File > New > Project` to bring up the New Project dialog box, shown in Figure 1-6.

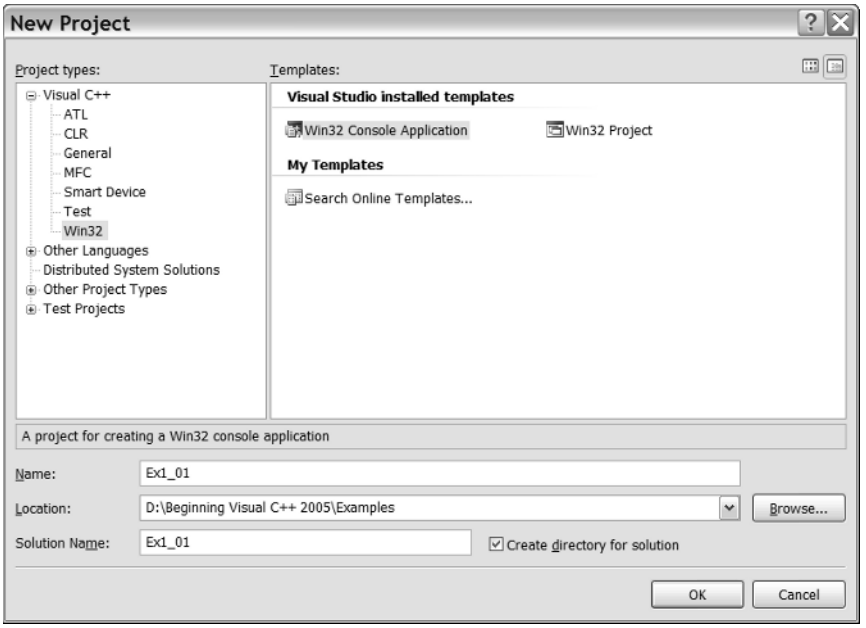

Figure 1-6

The left pane in the New Project dialog box displays the types of projects you can create; in this case, click `Win32`. This also identifies an application wizard that creates the initial contents for the project. The right pane displays a list of templates available for the project type you have selected in the left pane. The template you select is used by the application wizard when creating the files that make up the project. In the next dialog box, you have an opportunity to customize the files that are created when you click the `OK` button in this dialog box. For most of the type/template options, a basic set of program source modules are created automatically.

You can now enter a suitable name for your project by typing into the Name: edit box — for example, you could call this one Ex1_01, or you can choose your own project name. Visual C++ 2005 supports long file names, so you have a lot of flexibility. The name of the solution folder appears in the bottom edit box and, by default, the solution folder has the came name as the project. You can change this if you want. The dialog box also allows you to modify the location for the solution that contains your project — this appears in the `Location:` edit box. If you simply enter a name for your project, the solution folder is automatically set to a folder with that name, with the path shown in the Location: edit box. By default the solution folder is created for you if it doesn't already exist. If you want to specify a different path for the solution folder, just enter it in the Location: edit box. Alternatively, you can use the `Browse` button to select another path for your solution. Clicking the `OK` button displays the Win32 Application Wizard dialog box shown in Figure 1-7.

This dialog box explains the settings currently in effect. If you click the `Finish` button, the wizard creates all the project files based on this. In this case you can click `Applications Settings` on the left to display the `Application Settings` page of the wizard shown in Figure 1-8.

Figure 1-7

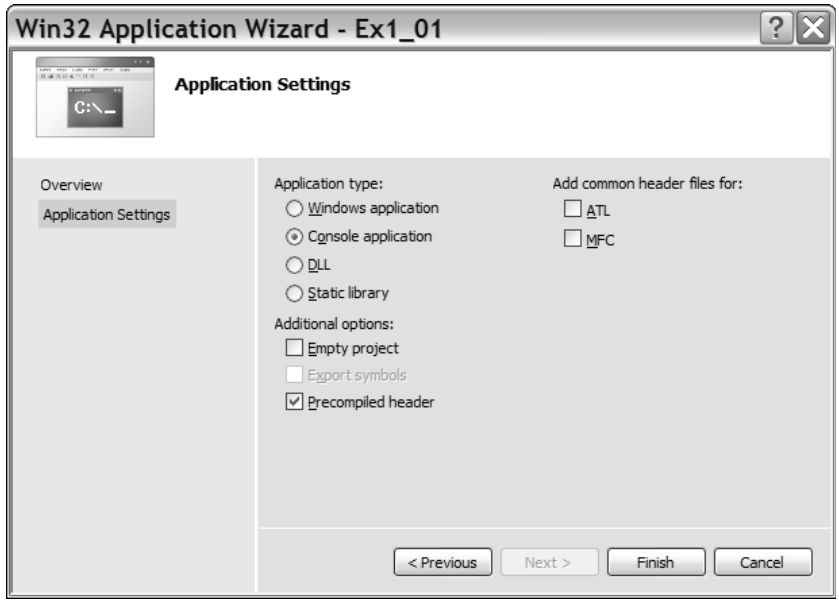

Figure 1-8

The `Application Settings` page allows you to choose options that you want to apply to the project. For most of the projects you'll be creating when you are learning the C++ language, you select the `Empty project` checkbox, but here you can leave things as they are and click the `Finish` button. The application wizard then creates the project with all the default files.

The project folder will have the name that you supplied as the project name and will hold all the files making up the project definition. If you didn't change it, the solution folder has the same name as the project folder and contains the project folder plus the files defining the contents of the solution. If you use Windows Explorer to inspect the contents of the solution folder, you'll see that it contains three files:

- A file with the extension `.sln` that records information about the projects in the solution.
- A file with the extension `.suo` in which user options that apply to the solution will be recorded.
- A file with the extension `.ncb` that records data about **Intellisense** for the solution. Intellisense is the facility that provides auto-completion and prompting for code in the Editor window as you enter it.

If you use Windows Explorer to look in the project folder, notice there are six files initially, including a file with the name `ReadMe.txt` that contains a summary of the contents of the files that have been created for the project. The one file that `ReadMe.txt` may not mention is a file with a compound name of the form `Ex1_01.vcproj.ComputerName.UserName.user` used to store options you set for the project.

The project you have created will automatically open in Visual C++ 2005 with the left pane as in Figure 1-9. I have increased the width of this pane so that you can see the complete names on the tabs.

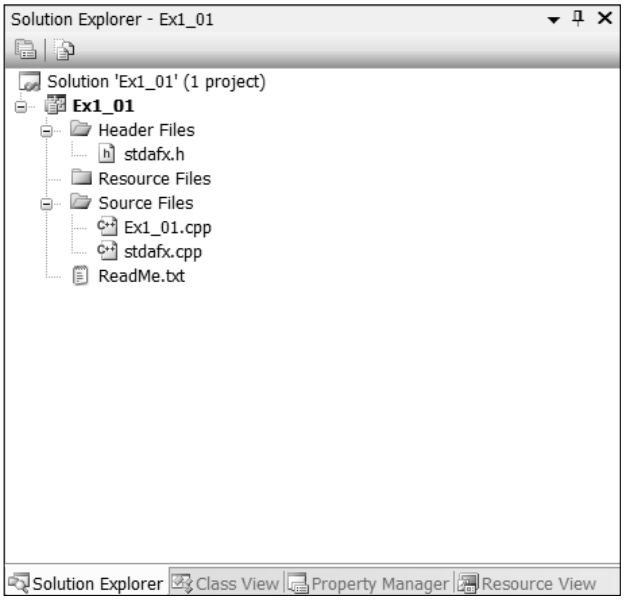

Figure 1-9

The **Solution Explorer** tab presents a view of all the projects in the current solution and the files they contains — here there is just one project of course. You can display the contents of any file as an additional tab in the Editor pane just by double-clicking in name in the Solution Explorer tab. In the edit pane you can switch instantly between any of the files that have been displayed just by clicking on the appropriate tab.

The **Class View** tab displays the classes defined in your project and also shows the contents of each class. You don't have any classes in this application, so the view is empty. When we discuss classes, you will see that you can use the `Class View` tab to move around the code relating to the definition and implementation of all your application classes quickly and easily.

The **Property Manager** tab shows the properties that have been set for the Debug and Release versions of your project. I'll explain these versions a little later in this chapter. You can change any of the properties shown by right-clicking a property and selecting Properties from the context menu; this displays a dialog box where you can set the project property. You can also press Alt+F7 to display the properties dialog box at any time; I'll also discuss this in more detail when we go into the Debug and Release versions of a program.

The **Resource View** shows the dialog boxes, icons, menus toolbars, and other resources that are used by the program. Because this is a console program, no resources are used; however, when you start writing Windows applications, you'll see a lot of things here. Through this tab you can edit or add to the resources available to the project.

Like most elements of the Visual C++ 2005 IDE, the Solution Explorer and other tabs provide context-sensitive pop-up menus when you right-click items displayed in the tab and in some cases in the empty space in the tab, too. If you find that the Solution Explorer pane gets in your way when writing code, you can hide it by clicking the Autohide icon. To redisplay it, click the name tab on the left of the IDE window.

Modifying the Source Code

The Application wizard generates a complete Win32 console program that you can compile and execute. Unfortunately, the program doesn't do anything as it stands, so to make it a little more interesting you need to change it. If it is not already visible in the Editor pane, double-click `Ex1_01.cpp` in the Solution Explorer pane. This file is the main source file for the program that the Application wizard generated and it looks like that shown in Figure 1-10.

Figure 1-10

If the line numbers are not displayed on your system, select Tools > Options from the main menu to display the Options dialog box. If you extend the C/C++ option in the right pane and select General from the extended tree, you can select Line Numbers in the right pane of the dialog box. I'll first give you a rough guide to what this code in Figure 1-10 does, and you'll see more on all of these later.

The first two lines are just comments. Anything following "//"in a line is ignored by the compiler. When you want to add descriptive comments in a line, precede your text by "//".

Line 4 is an `#include` directive that adds the contents of the file `stdafx.h` to this file in place of this `#include` directive. This is the standard way of adding the contents of `.h` source files to a `.cpp` source file a in a C++ program.

Line 7 is the first line of the executable code in this file and the beginning of the function `_tmain()`. A function is simply a named unit of executable code in a C++ program; every C++ program consists of at least one — and usually many more — functions.

Lines 8 and 10 contain left and right braces, respectively, that enclose all the executable code in the function `_tmain()`. The executable code is, therefore, just the single line 10 and all this does is end the program.

Now you can add the following two lines of code in the Editor window:

```
// Ex1_01.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#include <iostream>

int _tmain(int argc, _TCHAR* argv[])
{
  std::cout << "Hello world!\n";
  return 0;
}
```

The unshaded lines are the ones generated for you. The new lines you should add are shown shaded. To introduce each new line, place the cursor at the end on the text on the preceding line and press `Enter` to create an empty line in which you can type the new code. Make sure it is exactly as shown in the preceding example; otherwise, the program may not compile.

The first new line is an `#include` directive that adds the contents of one of the standard libraries for ISO/ANSI C++ to the source file. The `<iostream>` library defines facilities for basic I/O operations, and the one you are using in the second line that you added writes output to the command line. `std::cout` is the name of the standard output stream and you write the string `"Hello world!\n"` to `std::cout` in the second addition statement. Whatever appears between the pair of double quote characters is written to the command line.

Building the Solution

To build the solution, press `F7` or select the `Build > Build Solution` menu item. Alternatively, you can click the toolbar button corresponding to this menu item. The toolbar buttons for the `Build` menu may not display, but you can easily fix this by right-clicking in the toolbar area and selecting the `Build`

toolbar from those in the list. The program should then compile successfully. If there are errors, ensure you didn't make an error while entering the new code, so check the two new lines very carefully.

Files Created by Building a Console Application

After the example has been built without error, take a look in the project folder by using Windows Explorer to see a new subfolder called `Debug`. This folder contains the output of the build you just performed on the project. Notice that this folder contains several files.

Other than the `.exe` file, which is your program in executable form, you don't need to know much about what's in these files. In case you're curious, however, let's do a quick run-through of what the more interesting ones are for.

File Extension	Description
.exe	This is the executable file for the program. You get this file only if both the compile and link steps are successful.
.obj	The compiler produces these object files containing machine code from your program source files. These are used by the linker, along with files from the libraries, to produce your `.exe` file.
.ilk	This file is used by the linker when you rebuild your project. It enables the linker to incrementally link the object files produced from the modified source code into the existing `.exe` file. This avoids the need to re-link everything each time you change your program.
.pch	This is a **pre-compiled header** file. With pre-compiled headers, large tracts of code that are not subject to modification (particularly code supplied by C++ libraries) can be processed once and stored in the `.pch` file. Using the `.pch` file substantially reduces the time needed to rebuild your program.
.pdb	This file contains debugging information that is used when you execute the program in debug mode. In this mode, you can dynamically inspect information that is generated during program execution.
.idb	Contains information used when you rebuild the solution.

Debug and Release Versions of Your Program

You can set a range of options for a project through the `Project > Ex1_01 Properties` menu item. These options determine how your source code is processed during the compile and link stages. The set of options that produces a particular executable version of your program is called a **configuration**. When you create a new project workspace, Visual C++ 2005 automatically creates configurations for producing two versions of your application. One version, called the `Debug` version, includes information that helps you debug the program. With the `Debug` version of your program you can step through the code when things go wrong, checking on the data values in the program. The other, called the `Release` version, has no debug information included and has the code optimization options for the compiler turned on to provide you with the most efficient executable module. These two configurations are sufficient for your needs throughout this book, but when you need to add other configurations for an application, you can do so through the `Build > Configuration Manager` menu. Note that this menu item won't appear if you haven't got a project loaded. This is obviously not a problem, but might be confusing if you're just browsing through the menus to see what's there.

You can choose which configuration of your program to work with by selecting the configuration from the `Active solution configuration` drop-down list in the `Configuration Manager` dialog box, as shown in Figure 1-11.

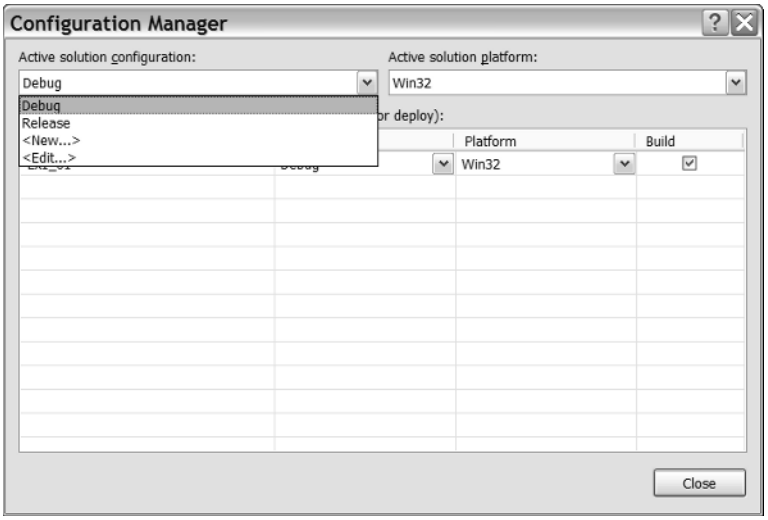

Figure 1-11

Select the configuration you want to work with from the list and then click the `Close` button. While you're developing an application, you'll work with the debug configuration. After your application has been tested using the debug configuration and appears to be working correctly, you typically rebuild the program as a release version; this produces optimized code without the debug and trace capability, so the program runs faster and occupies less memory.

Executing the Program

After you have successfully compiled the solution, you can execute your program by pressing Ctrl+F5. You should see the window shown in Figure 1-12.

Figure 1-12

As you see, you get the text that was between the double quotes written to the command line. The `"\n"` that appeared at the end of the text string is a special sequence called an **escape sequence** that denotes a newline character. Escape sequences are used to represent characters in a text string that you cannot enter directly from the keyboard.

Try It Out Creating an Empty Console Project

The previous project contained a certain amount of excess baggage that you don't need when working with simple C++ language examples. The precompiled headers option chosen by default resulted in the `stdafx.h` file being created in the project. This is a mechanism for making the compilation process more efficient when there are a lot of files in a program but this won't be necessary for many of our examples. In these instances you start with an empty project to which you can add your own source files. You can see how this works by creating a new project in a new solution for a `Win32` console program with the name `Ex1_02`. After you have entered the project name and clicked the `OK` button, click on `Applications Settings` on the right side of the dialog box that follows. You can then select `Empty project` from the additional options, as Figure 1-13 shows.

Figure 1-13

When you click the `Finish` button, the project is created as before, but this time without any source files.

Next you add a new source file to the project. Right-click the Solution Explorer pane and then select `Add > New Item` from the context menu. A dialog box displays; click `Code` in the right pane, and `C++ File(.cpp)` in the left pane. Enter the file name as `Ex1_02`, as shown in Figure 1-14.

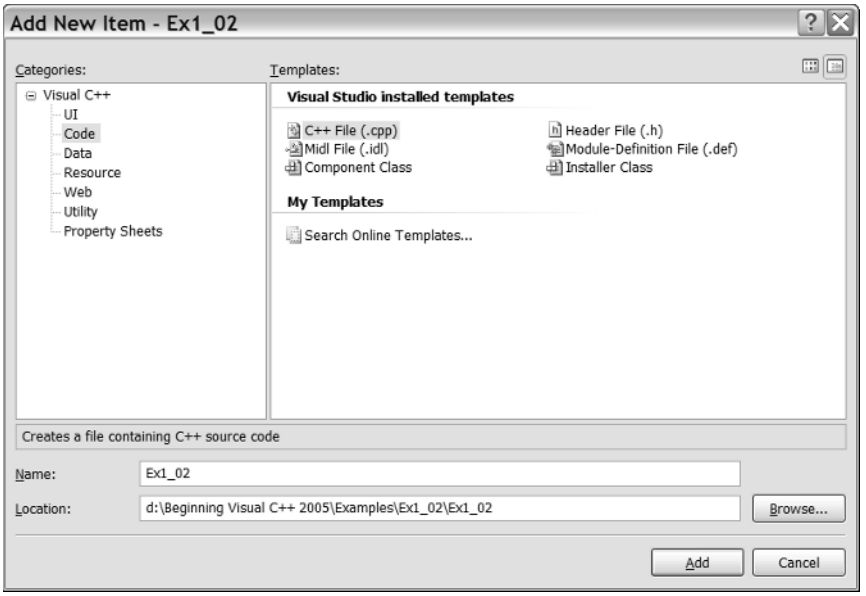

Figure 1-14

When you click the `Add` button, the new file is added to the project and is displayed in the Editor window. Of course, the file is empty so nothing will be displayed; enter the following code in the Editor window:

```
// Ex1_02.cpp A simple console program
#include <iostream>                    // Basic input and output library

int main()
{
  std::cout << "This is a simple program that outputs some text." << std::endl;
  std::cout << "You can output more lines of text" << std::endl;
  std::cout << "just by repeating the output statement like this." << std::endl;
  return 0;                            // Return to the operating system
}
```

Note the automatic indenting that occurs as you type the code. C++ uses indenting to make programs more readable, and the editor automatically indents each line of code that you enter, based on what was in the previous line. You can also see the syntax color highlighting in action as you type. Some elements of the program are shown in different colors as the editor automatically assigns colors to language elements depending on what they are.

The preceding code is the complete program. You probably noticed a couple of differences compared to the code generated by the Application wizard in the previous example. There's no `#include` directive for the `stdafx.h` file. You don't have this file as part of the project here because you are not using the precompiled headers facility. The name of the function here is `main`; before it was `_tmain`. In fact all ISO/ANSI C++ programs start execution in a function called `main()`. Microsoft also provides for this function to be called `wmain` when Unicode characters are used and the name `_tmain` is defined to be

either `main` or `wmain`, depending on whether or not the program is going to use Unicode characters. For the previous example, the name `_tmain` is defined behind the scenes to be `main`. You use the name `main` in all the ISO/ANSI C++ examples.

The output statements are a little different. The first statement in `main()` is:

```
std::cout << "This is a simple program that outputs some text." << std::endl;
```

You have two occurrences of the << operator, and each one sends whatever follows to `std::cout`, which is the standard output stream. First the string between double quotes is sent to the stream and then `std::endl` where `std::endl` is defined in the standard library as a newline character. Earlier you used the escape sequence `\n` for a newline character within a string between double quotes. You could have written the preceding statement as:

```
std::cout << "This is a simple program that outputs some text.\n";
```

I should explain why the line is shaded, where the previous line of code is not. Where I repeat a line of code for explanation purposes I show it unshaded. The preceding line of code is new and does not appear earlier so I have shown it shaded.

You can now build this project in the same way as the previous example. Note that any open source files in the Editor pane are saved automatically if you have not already saved them. When you have compiled the program successfully, press `Ctrl+F5` to execute it. The window shown in Figure 1-15 displays.

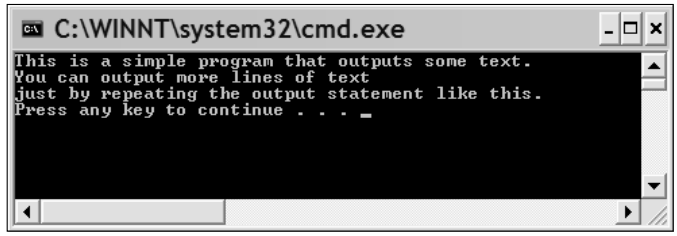

Figure 1-15

Dealing with Errors

Of course, if you didn't type the program correctly, you get errors reported. To show how this works, you could deliberately introduce an error into the program. If you already have errors of your own, you can use those to perform this exercise. Go back to the Editor pane and delete the semicolon at the end of the second-to-last line between the braces (line 8); then rebuild the source file. The Output pane at the bottom of the application window includes the error message:

```
C2143: syntax error : missing ';' before 'return'
```

Every error message during compilation has an error number that you can look up in the documentation. Here, the problem is obvious, ; however, in more obscure cases, the documentation may help you figure out what is causing the error. To get the documentation on an error, click the line in the `Output` pane that contains the error number and then press F1. A new window displays containing further information about the error. You can try it with this simple error, if you like.

When you have corrected the error, you can then rebuild the project. The build operation works efficiently because the project definition keeps track of the status of the files making up the project. During a normal build, Visual C++ 2005 recompiles only the files that have changed since the program was last compiled or built. This means that if your project has several source files and you've edited only one of the files since the project was last built, only that file is recompiled before linking to create a new `.exe` file.

You'll also use CLR console programs, so the next section shows you what a CLR console project looks.

Try It Out Creating a CLR Console Project

Press `Ctrl+Shift+N` to display the New Project dialog box; then select the project type as CLR and the template as `CLR Console Application`, as shown in Figure 1-16.

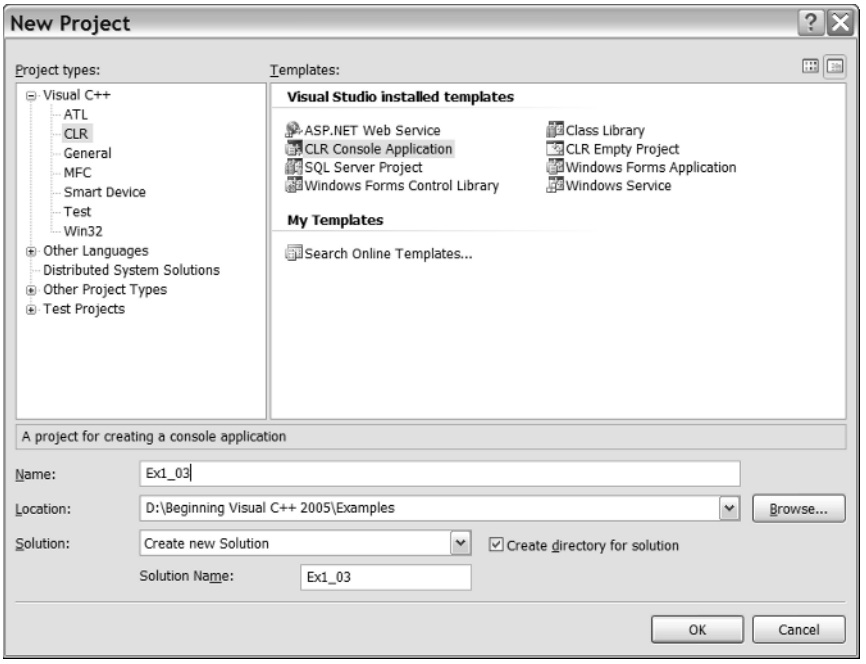

Figure 1-16

Enter the name as `Ex1_03`. When you click the `OK` button, the files for the project are created. There are no options for a CLR console project, so you always start with the same set of files in a project with this template. If you want an empty project — something you won't need with this book — there's a separate template for this.

If you look at the `Solution Explorer` pane shown in Figure 1-17, you see there are some extra files compare to a Win32 console project.

There are a couple of files in the virtual Resource Files folder. The `.ico` file stores an icon for the application that is displayed when the program is minimized; the `.rc` file records the resources for the application — just the icon in this case.

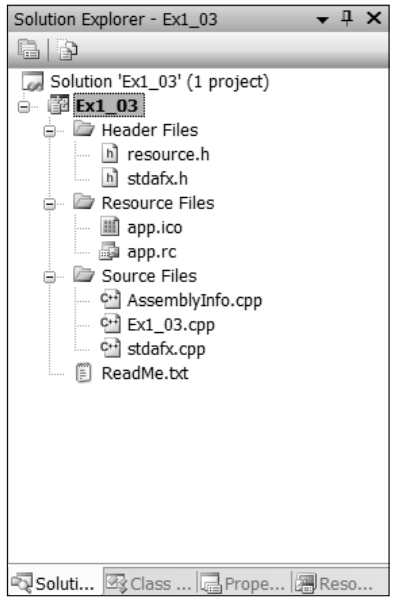

Figure 1-17

There is also a file with the name `AssemblyInfo.cpp`. Every CLR program consists of one or more **assemblies** where an assembly is a collection of code and resources that form a functional unit. An assembly also contains extensive data for the CLR; there are specifications of the data types that are being used, versioning information about the code, and information that determines if the contents of the assembly can be accessed from another assembly. In short, an assembly is a fundamental building block in all CLR programs.

If the source code in the `Ex1_03.cpp` file is not displayed in the Editor window, double-click the file name in the Solution Explorer pane. It should look like Figure 1-18.

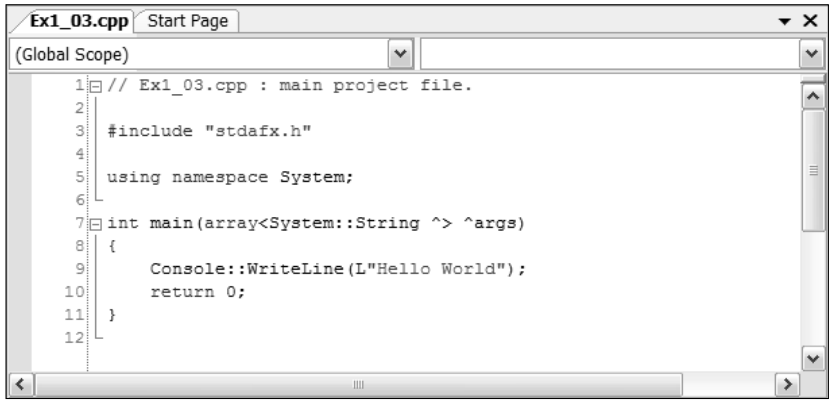

Figure 1-18

It has the same `#include` directive as the default native C++ console program because CLR programs use precompiled headers for efficiency. The next line is new:

```
using namespace System;
```

The .NET library facilities are all defined within a **namespace**, and all the standard sort of stuff you are likely to use is in a namespace with the name `System`. This statement indicates the program code that follows uses the `System` namespace, but what exactly is a namespace?

A namespace is a very simple concept. Within your program code and within the code that forms the .NET libraries, names have to be given to lots of things — data types, variables, and blocks of code called functions all have to have names. The problem is that if you happen to invent a name that is already used in the library, there's potential for confusion. A namespace provides a way of getting around this problem. All the names in the library code that is defined within the `System` namespace are implicitly prefixed with the namespace name. So, a name such as `String` in the library is really `System::String`. This means that if you have inadvertently used the name `String` for something in your code, you can use `System::String` to refer `String` from the .NET library.

The two colons[md]`::` — are an operator called the scope resolution operator. Here the scope resolution operator separates the namespace name `System` from the type name `String`. You have seen this in the native C++ examples earlier in this chapter with `std::cout` and `std::endl`. This is the same story — `std` is the namespace name for native C++ libraries, and `cout` and `endl` are the names that have been defined within the `std` namespace to represent the standard output stream and the newline character, respectively.

In fact, the `using namespace` statement in the example allows you to use any name from the `System` namespace without having to use the namespace name as a prefix. If you did end up with a name conflict between a name you have defined and a name in the library, you could resolve the problem by removing the `using namespace` statement and explicitly qualify the name from the library with the namespace name. You'll learn more about namespaces in Chapter 2.

You can compile and execute the program by pressing `Ctrl+F5`. The output is as shown in Figure 1-19.

Figure 1-19

The output is the same as from the first example. This output is produced by the line:

```
Console::WriteLine(L"Hello World");
```

This uses a .NET library function to write the information between the double quotes to the command line, so this is the CLR equivalent of the native C++ statement that you added to Ex1_01:

```
std::cout << "Hello world!\n";
```

It is more immediately apparent what the CLR statement does than the native C++ statement.

Setting Options in Visual C++ 2005

There are two sets of options you can set. You can set options that apply to the tools provided by Visual C++ 2005, which apply in every project context. Also, you can set options that are specific to a project and determine how the project code is to be processed when it is compiled and linked. Options are set through the Options dialog box that's displayed when you select `Tools > Options` from the main menu. The Options dialog box is shown in Figure 1-20.

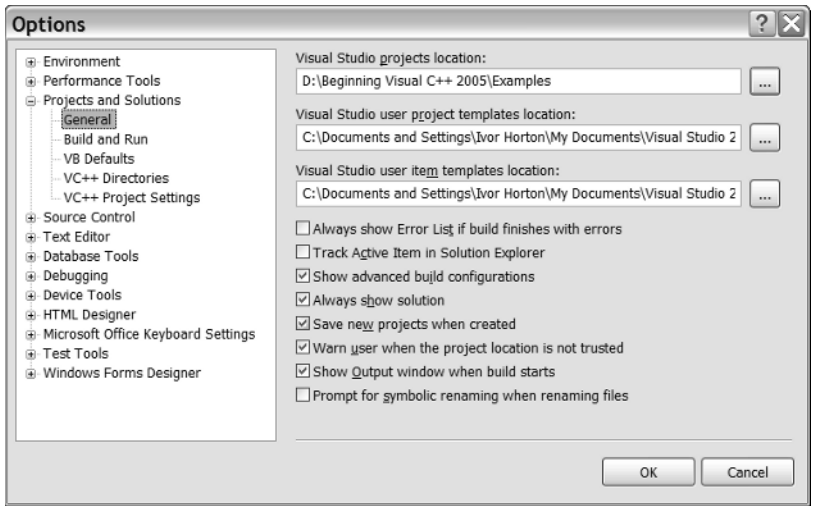

Figure 1-20

Clicking the plus sign (+) for any of the items in the left pane displays a list of subtopics. Figure 1-20 shows the options for the `General` subtopic under `Projects and Solutions`. The right pane displays the options you can set for the topic you have select in the left pane. You should concern yourself with only a few of these at this time, but you'll find it useful to spend a little time browsing the range of options available to you. Clicking the Help button (with the ?) at the top right of the dialog box displays an explanation of the current options.

You probably want to choose a path to use as a default when you create a new project, and you can do this through the first option shown in Figure 1-20. Just set the path to the location where you want your projects and solutions stored.

You can set options that apply to every C++ project by selecting the `Projects and Solutions > VC++ Project Settings` topic in the left pane. You can also set options specific to the current project through the `Project > Properties` menu item in the main menu. This menu item label is tailored to reflect the name of the current project.

Creating and Executing Windows Applications

Just to show how easy it's going to be, now create two working Windows applications. You'll create a native C++ application using MFC and then you'll create a Windows Forms application that runs with the CLR. I'll defer discussion of the programs that you generate until I've covered the necessary ground for you to understand it in detail. You will see, though, that the processes are straightforward.

Creating an MFC Application

To start with, if an existing project is active — as indicated by the project name appearing in the title bar of the Visual C++ 2005 main window — you can select `Close Solution` from the `File` menu. Alternatively, you can create a new project and have the current solution closed automatically.

To create the Windows program select New > Project from the File menu or press `Ctrl+Shift+N;` then choose the project type as MFC and select MFC Application as the project template. You can then enter the project name as `Ex1_04` as shown in Figure 1-21.

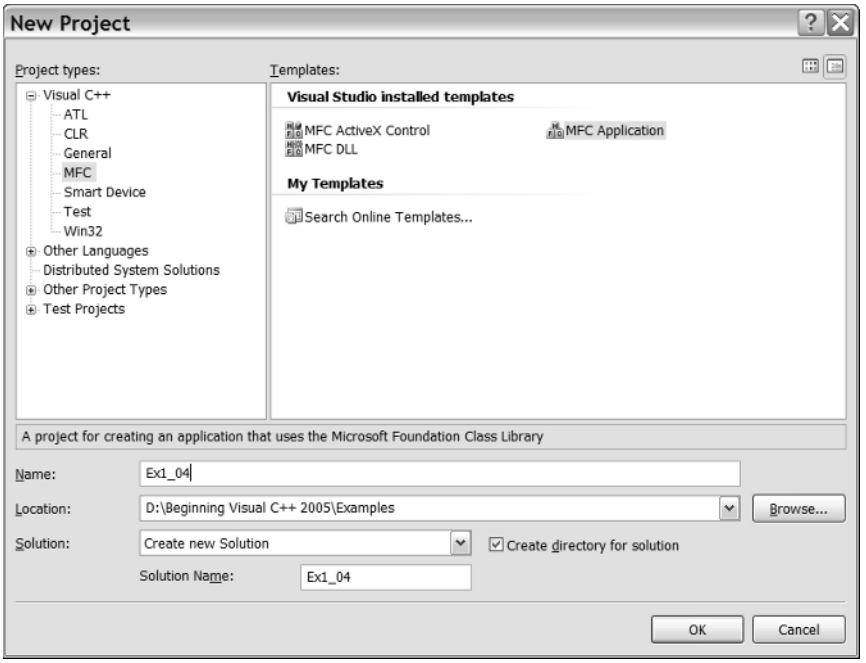

Figure 1-21

When you click the `OK` button, the `MFC Application Wizard` dialog box is displayed. The dialog box has a range of options that let you choose which features you'd like to have included in your application. These are identified by the items in the list on the right of the dialog box, as Figure 1-22 shows. You'll get to use many of these in examples later on.

You can ignore all these options in this instance and just accept the default settings, so click the Finish button to create the project with the default settings. The Solution Explorer pane in the IDE window looks like Figure 1-23.

Figure 1-22

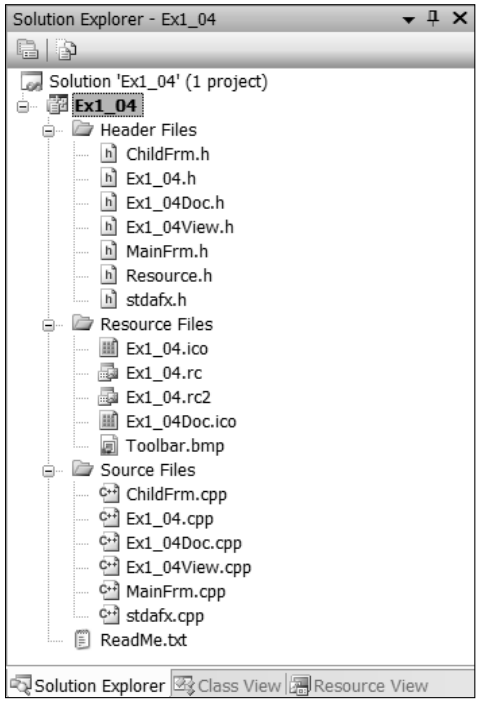

Figure 1-23

Note that I have hidden the Property Manager tab by right-clicking it and selecting Hide, so it doesn't appear in Figure 1-23. The list shows a large number of files that have been created. You need plenty of space on your hard drive when writing Windows programs! The files with the extension `.cpp` contain executable C++ source code, and the `.h` files contain C++ code consisting of definitions that are used by the executable code. The `.ico` files contain icons. The files are grouped into the subfolders you can see for ease of access. These aren't real folders, though, and they won't appear in the project folder on your disk.

If you now take a look at the `Ex1_04` solution folder using Windows Explorer or whatever else you may have handy for looking at the files on your hard disk, notice that you have generated a total of 24 files. Three of these are in the solution folder, a further 17 are in the project folder and four more are in a subfolder, `res`, to the project folder. The files in the `res` subfolder contain the resources used by the program — such as the menus and icons used in the program. You get all this as a result of just entering the name you want to assign to the project. You can see why, with so many files and file names being created automatically, a separate directory for each project becomes more than just a good idea.

One of the files in the `Ex1_04` project directory is `ReadMe.txt`, and this provides an explanation of the purpose of each of the files that the MFC Application wizard has generated. You can take a look at it if you want, using Notepad, WordPad, or even the Visual C++ 2005 editor. To view it in the Editor window, double-click it in the Solution Explorer pane.

Building and Executing the MFC Application

Before you can execute the program, you have to build the project — meaning, compile the source code and link the program modules. You do this in exactly the same way that you did with the console application example. To save time, press `Ctrl+F5` to get the project built and then executed in a single operation.

After the project has been built, the Output window indicates that there are no errors and the executable starts running. The window for the program you've generated is shown in Figure 1-24.

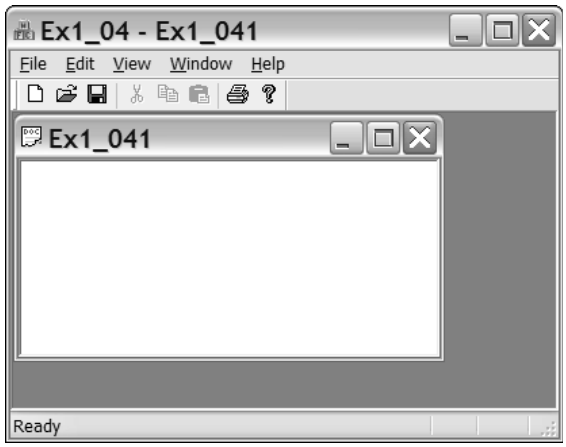

Figure 1-24

As you see, the window is complete with menus and a toolbar. Although there is no specific functionality in the program — that's what you need to add to make it *your* program — all the menus work. You can try them out. You can even create further windows by selecting `New` from the `File` menu.

I think you'll agree that creating a Windows program with the MFC Application wizard hasn't stressed too many brain cells. You'll need to get a few more ticking away when you come to developing the basic program you have here into a program that does something more interesting, but it won't be that hard. Certainly, for many people, writing a serious Windows program the old-fashioned way, without the aid of Visual C++ 2005, required at least a couple of months on a fish diet before making the attempt. That's why so many programmers used to eat sushi. That's all gone now with Visual C++ 2005. You never know, however, what's around the corner in programming technology. If you like sushi, it's best to continue with it to be on the safe side.

Creating a Windows Forms Application

This is a job for another application wizard. So create yet another new project, but this time select the type as `CLR` in the left pane of the New Project dialog box and the template as `Windows Forms Application`. You can then enter the project name as `Ex1_05` as shown in Figure 1-25.

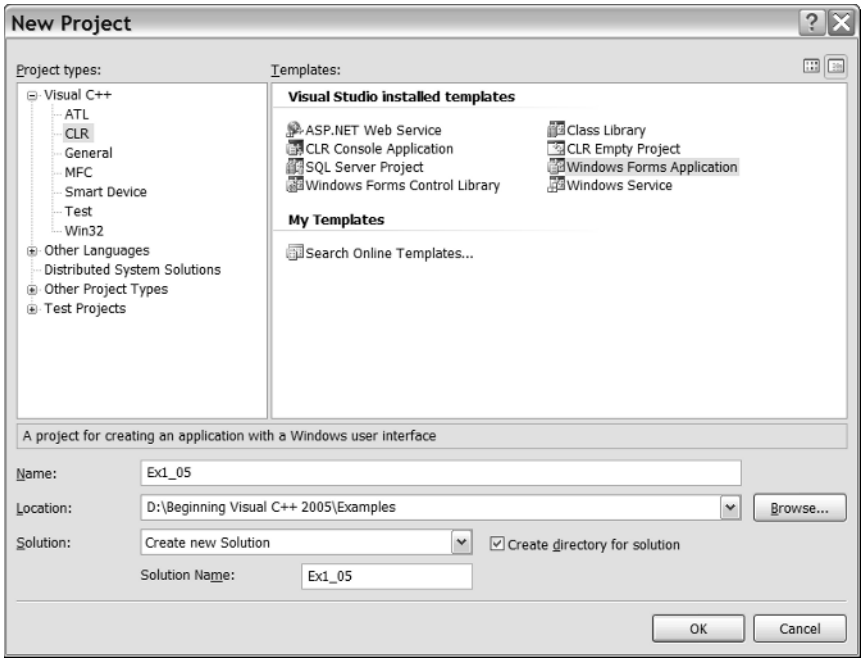

Figure 1-25

There are no options to choose from in this case, so click the `OK` button to create the project.

The Solution Explorer pane in Figure 1-26 shows the files that have been generated for this project.

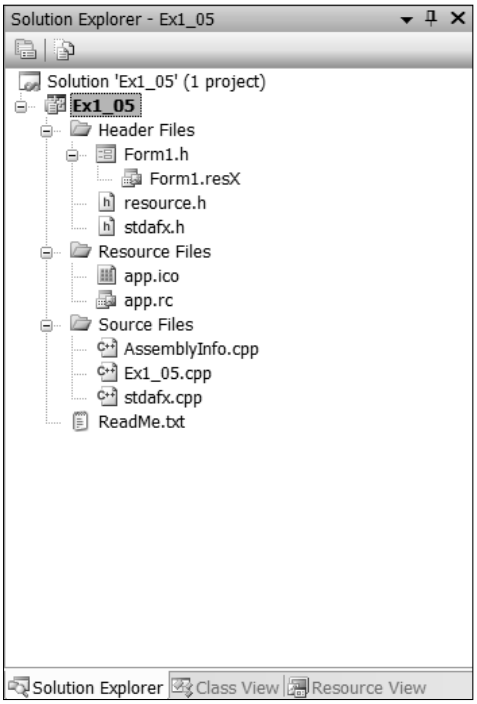

Figure 1-26

There are considerably fewer files in this project — if you look in the directories, you'll see there are a total of 15 including the solution files. One reason for this is the initial GUI is much simpler than the native C++ application using MFC. The Windows Forms application has no menus or toolbars, and there is only one window. Of course, you can add all these things quite easily, but the wizard for a Windows Forms application does not assume you want them from the start.

The `Editor` window looks rather different as Figure 1-27 shows.

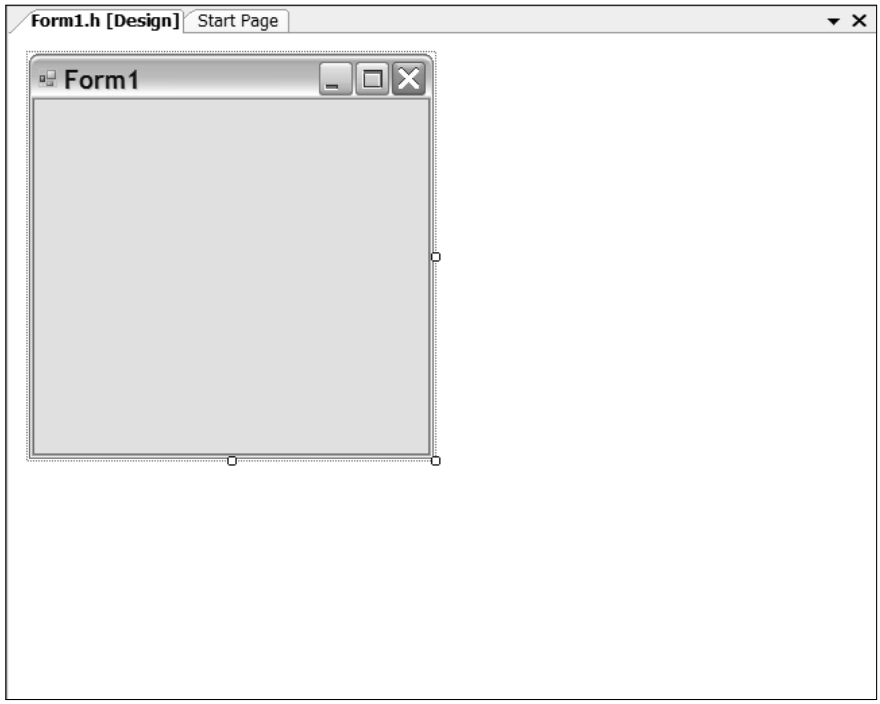

Figure 1-27

The `Editor` window shows an image of the application window rather than code. The reason for this is that developing the GUI for a Windows Forms is oriented towards a graphical design approach rather than a coding approach. You add GUI components to the application window by dragging or placing them there graphically, and Visual C++ 2005 automatically generates the code to display them. If you press `Ctrl+Alt+X` or select `View > Toolbox`, you'll see an additional window displayed showing a list of GUI components as in Figure 1-28.

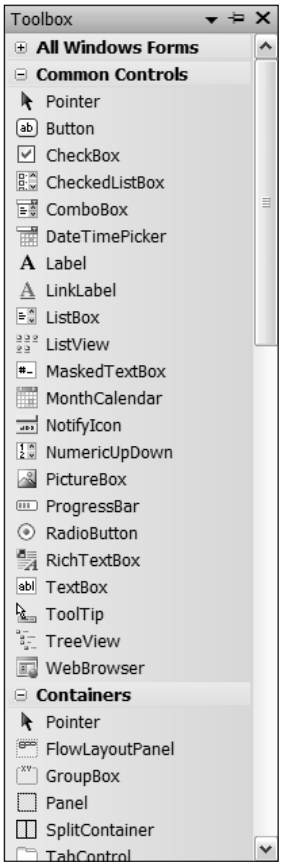

Figure 1-28

The `Toolbox` window presents a list of standard components that you can add to a Windows Forms application. You can try adding some buttons to the window for Ex1_05. Click `Button` in the `Toolbox` window list and then click in the client area of the Ex1_05 application window that is displayed in the Editor window where you want the button to be placed. You can adjust the size of the button by dragging its borders, and you can reposition the button by dragging it around. You can also change the caption just by typing — try entering `Start` on the keyboard and then press `Enter`. The caption changes and along the way another window displays, showing the properties for the button. I won't go into these now, but essentially these are the specifications that affect the appearance of the button, and you can change these to suit your application. Try adding another button with the caption Stop, for example. The Editor window will look like Figure 1-29.

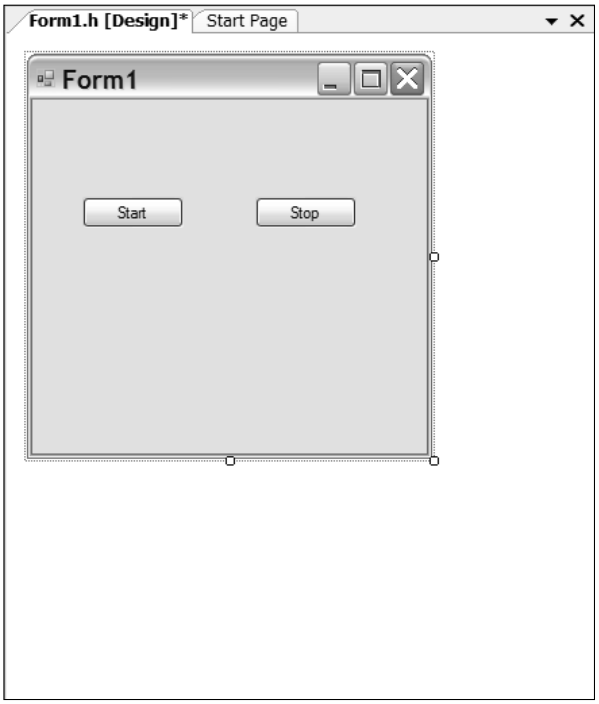

Figure 1-29

You can graphically edit any of the GUI components at any time, and the code adjusts automatically. Try adding a few other components in the same way and then compile and execute the example by pressing `Ctrl+F5`. The application window displays in all its glory. Couldn't be easier, could it?

Summary

In this chapter, you have run through the basic mechanics of using Visual C++ 2005 to create applications of various kinds. You created and executed native and CLR console programs, and with the help of the application wizards, you created an MFC-based Windows program and a Windows Forms program that executes with the CLR.

The points from this chapter that you should keep in mind are:

- The Common Language Runtime (CLR) is the Microsoft implementation of the Common Language Infrastructure (CLI) standard.
- The .NET Framework comprises the CLR plus the .NET libraries that support applications targeting the CLR.
- Native C++ applications are written the ISO/ANSI C++ language.
- Programs written in the C++/CLI language execute with the CLR.

- A solution is a container for one or more projects that form a solution to an information-processing problem of some kind.
- A project is a container for the code and resource elements that make up a functional unit in a program.
- An assembly is a fundamental unit in a CLR program. All CLR programs are made up of one or more assemblies.

Starting with the next chapter, you'll use console applications extensively throughout the first half of the book. All the examples illustrating how C++ language elements are used are executed using either Win32 or CLR console applications. You will return to the Application Wizard for MFC-based programs as soon as you have finished delving into the secrets of C++.