1 Programming with Visual C++

WHAT YOU WILL LEARN IN THIS CHAPTER:

- What the principal components of Visual C++ are
- 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

WROX.COM CODE DOWNLOADS FOR THIS CHAPTER

You can find the wrox.com code downloads for this chapter on the Download Code tab at `www.wrox.com/go/beginningvisualc`. The code is in the Chapter 1 download and individually named according to the names throughout the chapter.

LEARNING WITH VISUAL C++

Windows programming isn't difficult. Microsoft Visual C++ 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 capabilities. Object-oriented techniques are central to the effectiveness of all the tools provided by Visual C++ 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++. 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 way to get familiar with it is to work through the process of creating, compiling, and executing a simple program. So power up your PC, start Windows, load the mighty Visual C++, and begin your journey.

WRITING C++ APPLICATIONS

You have tremendous flexibility in the types of applications and program components that you can develop with Visual C++. Applications that you can develop fall into two broad categories: *desktop applications* and *Windows Store apps*. Desktop applications are the applications that you know and love; they have an application window that typically has a menu bar and a toolbar and frequently a status bar at the bottom of the application window. This book focuses primarily on desktop applications.

Windows Store apps only run under Windows 8 or later versions and have a user interface that is completely different from desktop applications. The focus is on the content where the user interacts directly with the data, rather than interacting with controls such as menu items and toolbar buttons.

Once you have learned C++, this book concentrates on using the Microsoft Foundation Classes (MFC) with C++ for building desktop applications. The application programming interface (API) for Windows desktop applications is referred to as *Win32*. Win32 has a long history and was developed long before the object-oriented programming paradigm emerged, so it has none of the object-oriented characteristics that would be expected if it were written today. The MFC consists of a set of C++ classes that encapsulate the Win32 API for user interface creation and control and greatly eases the process of program development. You are not obliged to use the MFC, though. If you want the ultimate in performance you can write your C++ code to access the Windows API directly, but it certainly won't be as easy.

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

Figure 1-1 is a simplified representation of what is involved. Desktop applications can target Windows 7, Windows 8, or Windows Vista. Windows Store apps execute only with Windows 8 and its successors and you must have Visual Studio 2013 installed under Windows 8 or later to develop them. Windows Store apps communicate with the operating system through the Windows Runtime, WinRT. I'll introduce you to programming Windows 8 applications in Chapter 18.

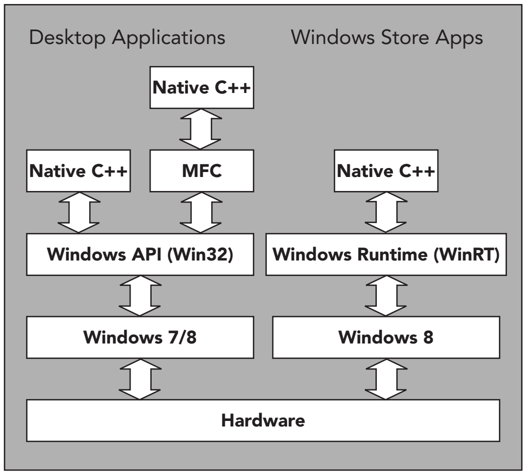


FIGURE 1-1

LEARNING DESKTOP APPLICATIONS PROGRAMMING

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

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

> ***NOTE*** *As I'll explain in Chapter 18, Windows Store apps are different. You specify the GUI in XAML, and the XAML is processed to generate the C++ code for GUI elements.*

Learning C++

Visual C++ supports the C++ language defined by the most recent ISO/IEC C++ standard that was published in 2011. The standard is defined in the document ISO/IEC 14882:2011 and commonly referred to as *C++ 11*. The Visual C++ compiler supports most of the language features introduced by this latest standard, and it includes some features from the draft for the next standard, C++ 14. Programs that you write in standard 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. C++ is the first choice of a great many professional program developers because it is so widely supported, and because it is one of the most powerful programming languages available today.

Chapters 2 through 9 of this book teach you the C++ language and introduce some of the most commonly used C++ standard library facilities along the way. Chapter 10 explains how you can use the Standard Template Library (STL) for C++ for managing collections of data.

C++ Concepts

As with virtually all programming languages, there's a chicken and egg problem in explaining C++. Inevitably there are occasions when I need to reference or make use of a language feature before I have discussed it in detail. This section is intended to help with this conundrum by outlining the principle C++ language elements. Of course, everything I mention here will be explained fully later in the book.

Functions

Every C++ program consists of at least one, and usually many, *functions*. A function is a named block of executable code that you invoke or *call* using its name. There must always be one function with the name `main`, and execution always starts with the `main()` function. The parentheses following the function name can specify what information is passed to a function when you call it. I'll always put parentheses after a function name in the text to distinguish it from other things. All the executable code in a program is contained in functions. The simplest C++ program consists of just the `main()` function.

Data and Variables

You store an item of data in a *variable*. A variable is a named memory area that can store a data item of a particular type. There are several standard *fundamental data types* that store integers, non-integral numerical values, and character data. You can also define your own data types, which makes writing a program that deals with real-world objects much easier. Variables of types that you define store *objects*. Because each variable can only store data of a given type, C++ is said to be a *type-safe* language.

Classes and Objects

A *class* is a block of code that defines a data type. A class has a name that is the name for your data type. An item of data of a class type is referred to as an *object*. You use the class type name when you create variables that can store objects of your data type.

Templates

Circumstances often arise when you need several different classes or functions in a program where the code for these only differs in the kind of data they work with. Templates save a lot of coding effort in such situations.

A *template* is a recipe or specification that you create that can be used by the compiler to generate code automatically in a program when requested. You can define *class templates* that the compiler can use to generate one or more of a family of classes. You can also define *function templates* that the compiler can use to generate functions. Each template has a name that you use when you want the compiler to create an instance of it.

The code for the class or function that the compiler generates from a template depends on one or more *template argument*s. The arguments are usually types, but not always. Typically you specify the template arguments explicitly when you use a class template. The compiler can usually deduce the arguments for a function template from the context.

Program Files

C++ program code is stored in two kinds of files. *Source files* contain executable code and have the extension `.cpp`. *Header files* contain definitions for things, such as classes and templates, that are used by the executable code. Header files have the extension `.h`.

Console Applications

Visual C++ *console applications* enable you to write, compile, and test C++ programs that have none of the baggage required by Windows desktop applications. These programs are called console applications because you communicate with them through the keyboard and the screen in character mode, so they are essentially character-based, command-line programs.

In Chapters 2 through 10 you'll only be working with console applications. Writing console applications might seem to be side-tracking you from the main objective of programming Windows applications with a GUI. However, when it comes to learning C++, it's by far the best way to proceed in my view. 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++. In the early chapters of the book you'll be learning C++ with a few lightweight console applications, before you get to work with the heavyweight sacks of code that are implicit in the world of Windows.

Windows Programming Concepts

The project creation facilities in Visual C++ can generate skeleton code automatically for a variety of applications. A Windows program has a completely different structure from that of the typical console program, and it's much more complicated. In a console program, you can get user input from the keyboard and write output back to the command line directly, and that is essentially it. A Windows application can access the input and output facilities of the computer only by way of functions supplied by the host environment; no direct access to the hardware resources is permitted. Several programs can be executing concurrently under Windows, so the operating system has to determine which application should receive a given raw input, such as a mouse click or the pressing of a key on the keyboard, and signal the program accordingly. Thus, the Windows operating system always manages all communications with the user.

The nature of the interface between a user and a Windows desktop 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 any of several toolbar buttons, or click the mouse somewhere in the application window. A well-designed Windows application has to be prepared to deal with any of these 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 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. 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 block within the Desktop Application block in Figure 1-2 represents a piece of code that deals with a particular kind of event. The program may appear to be somewhat fragmented because of the 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.

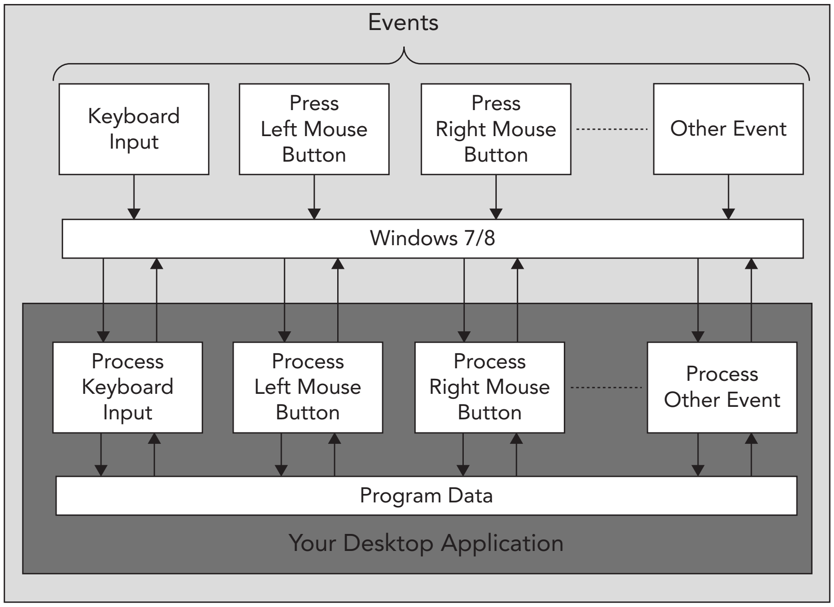


FIGURE 1-2

Of course, modules servicing external events, such as the selection of a menu or a mouse click, will typically need access to a common set of application-specific data. This data contains information that relates to what the program is about — for example, blocks of text recording scoring records for a player 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 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 generated by the application wizards that come with Visual C++, "several" turns out to be "very many". To simplify the process of understanding how C++ works, you need a context that is as uncomplicated as possible and at the same time has the tools to make it easy to create and navigate around sacks of code. Fortunately, Visual C++ comes with an environment that is designed specifically for the purpose.

THE INTEGRATED DEVELOPMENT ENVIRONMENT

The *integrated development environment* (IDE) is a self-contained environment in Visual C++ for creating, compiling, linking, testing, and debugging C++ programs of any kind. It also happens to be a great environment in which to learn the language (particularly when combined with a great book). The IDE incorporates a range of fully integrated tools that make the whole process of writing 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, I'll introduce you to the basics to get a view of how the IDE works and then you'll be able to pick up the rest in context as you go along.

The fundamental elements you'll be working with through the IDE are the editor, the C++ compiler, the linker, and the libraries. These are the basic tools that are essential to writing and executing a C++ program.

The Editor

The *editor* is an interactive environment in which you create and edit C++ source code. As well as the usual facilities such as cut and paste that you are certainly already familiar with, the editor offers a wide range of capabilities to help you get things right. For example:

- Code is automatically laid out with standard indentation and spacing. There's a default arrangement for code, but you can customize how your code is arranged in the dialog that displays when you select Tools ➪ Options from the menu.
- Fundamental words in C++ are recognized automatically and colored according to what they are. This makes your code more readable and easier to follow.
- *IntelliSense* analyzes code as you enter it. Anything that is incorrect or any words IntelliSense doesn't recognize are underlined with a red squiggle. It also provides prompts when it can determine the options for what you need to enter next. This saves typing because you can just select from a list.

> ***NOTE*** *IntelliSense doesn't just work with C++. It works with XAML too.*

The Compiler

You execute the *compiler* when you have entered the C++ code for your program. The compiler converts your source code into *object code*, and detects and reports errors in the compilation process. The compiler detects a wide range of errors caused by invalid or unrecognized program code, as well as structural errors, such as parts of a program that can never be executed. The object code generated by the compiler is stored in *object files* that have the extension `.obj`.

The Linker

The *linker* combines the modules generated by the compiler from source code files, adds required code modules from the standard libraries that are supplied as part of C++, and welds everything into an executable whole, usually in the form of an `.exe` file. 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 a collection of prewritten routines that support and extend the C++ language by providing standard professionally produced code units for common operations that you can incorporate into your programs. The operations implemented by the libraries greatly enhance productivity by saving you the effort of writing and testing the code for such operations yourself.

The Standard C++ Library

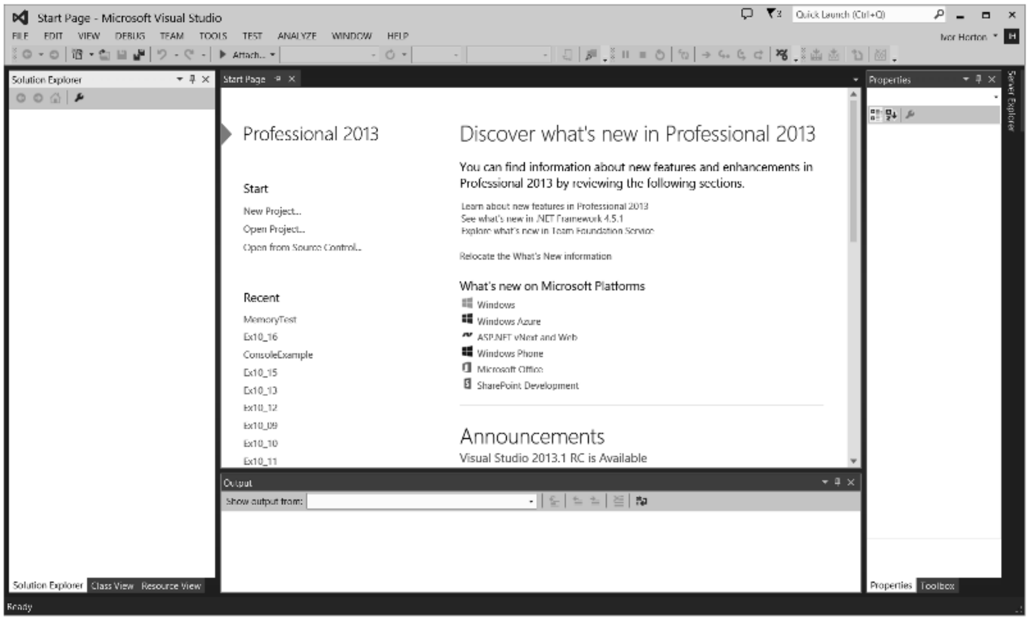
The *Standard C++ Library* defines a set of facilities that are common to all ISO/IEC standard-conforming C++ compilers. It contains a vast range of commonly used routines, including numerical functions, such as calculating square roots and evaluating trigonometrical functions; character- and string-processing functions, such as the classification of characters and the comparison of character strings; and many others. It also defines data types and standard templates for generating customized data types and functions. You'll learn about many of these as you develop your knowledge of C++.

Microsoft Libraries

Windows desktop applications are supported by a library called the *Microsoft Foundation Classes* (MFC). The MFC greatly reduces the effort needed to build the GUI for an application. (You'll see a lot more of the MFC when you finish exploring the nuances of the C++ language.) There are other Microsoft libraries for desktop applications, but you won't be exploring them in this book.

USING THE IDE

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



Professional 2013

Start

New Project...
Open Project...
Open from Source Control...

Recent

MemoryTest
Ex10_16
ConsoleExample
Ex10_15
Ex10_13
Ex10_12
Ex10_09
Ex10_10
Ex10_11

Discover what's new in Professional 2013

You can find information about new features and enhancements in Professional 2013 by reviewing the following sections.

Learn about new features in Professional 2013
See what's new in .NET Framework 4.5.1
Explore what's new in Team Foundation Service

Relocate the What's New information

What's new on Microsoft Platforms

Windows
Windows Azure
ASP.NET vNext and Web
Windows Phone
Microsoft Office
SharePoint Development

Announcements

Visual Studio 2013.1 RC is Available

FIGURE 1-3

The pane to the left in Figure 1-3 is the *Solution Explorer window*, the middle pane presently showing the Start page is the *Editor window*, and the tab visible in the pane at the bottom is the *Output window*. The Properties pane on the right displays properties for a variety of entities in your program. The Solution Explorer pane enables you to navigate through your program files and display their contents in the Editor window, and to add new files to your program. You can dock several windows where the Solution Explorer pane is located. Three are shown in Figure 1-3 and you can select other windows to be displayed here from the View menu. You can rearrange the windows by dragging their labels. The Editor window is where you enter and modify source code and other components of your application. The Output window displays the output from build operations during which a project is compiled and linked. You can choose to display other windows by selecting from the View menu.

Note that a window can be undocked from its position in the Visual Studio application window. Just right-click the title bar of the window you want to undock and select Float from the pop-up menu. In general, I will show windows in their undocked state in the book. You can restore a window to its docked state by right-clicking its title bar and selecting Dock from the pop-up or by dragging it with the left mouse button down to the position that you want in the application window.

Toolbar Options

You can choose which toolbars are displayed by right-clicking in the toolbar area. The range of toolbars in the list depends on which edition of Visual Studio 2013 you have installed. A pop-up menu with a list of toolbars (Figure 1-4) appears, and the toolbars that are currently displayed have checkmarks alongside them.

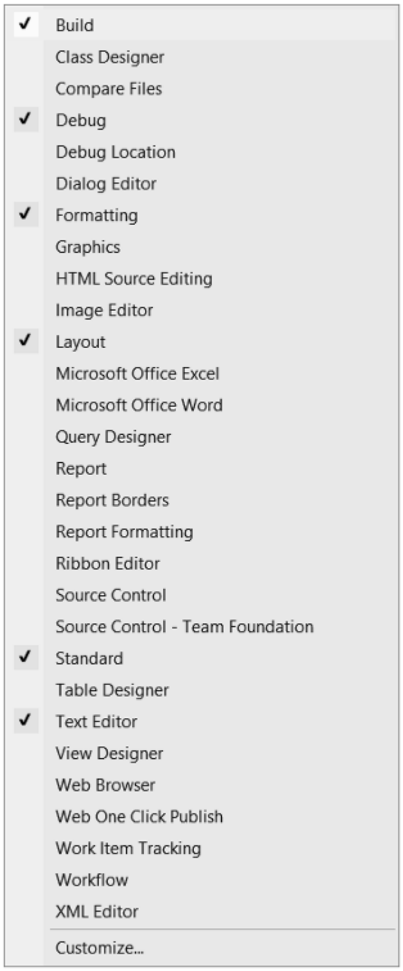


FIGURE 1-4

This is where you decide which toolbars are visible at any one time. To start with, make sure the Build, Debug, Formatting, Layout, Standard, and Text Editor menu items are selected. Clicking a toolbar in the list checks it if it is deselected, and results in it being displayed; clicking a toolbar that is selected deselects it and hides the toolbar.

> ***NOTE*** *A toolbar won't necessarily display all of its buttons. You can add or remove buttons for a toolbar by clicking the down arrow that appears at the right of the button set. The buttons in the Text Editor toolbar that indent and unindent a set of highlighted statements are particularly useful, as are the buttons that comment out or uncomment a selected set of statements, so make sure these are displayed.*

You don't have to clutter up the application window with all the toolbars you think you might need. Some toolbars appear automatically when required, and you'll find that the default set of toolbars is adequate most of the time. As you develop your application, it may sometimes be more convenient to have access to a different set of toolbars. You can change the set of visible toolbars at any time by right-clicking in the toolbar area and choosing from the context menu.

> ***NOTE*** *As in many other Windows applications, the toolbars come complete with tooltips. If you let the mouse pointer linger over a toolbar button for a second or two, a label will display the function of that button.*

Dockable Toolbars

A *dockable toolbar* is one that you can move around to position it at a convenient place in the window. Any of the toolbars can be docked at any of the four sides of the application window. Right-clicking in the toolbar area and selecting Customize from the pop-up will display the Customize dialog. You can choose where a particular toolbar is docked by selecting it and clicking the Modify Selection button. You can then choose from the drop-down list that appears to dock the toolbar where you want.

You'll recognize many of the toolbar icons 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++.

Documentation

There will be plenty of occasions when you'll want to find out more information about Visual C++ and its features and options. Pressing Ctrl+F1 will display the online product documentation in your browser. Pressing F1 with the cursor on a C++ language element in your code or a standard library item will open a browser window showing documentation for the element. The Help menu also provides various routes into the documentation, as well as access to program samples and technical support.

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. A project usually consists of several 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 `.vcxproj`, which is also in the project folder. The project folder contains other folders that are used to store the output from compiling and linking your project.

A *solution* is a mechanism for bringing together one or more 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, each of which is 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, and there will be one or more project folders as subfolders of the solution folder. Information about the projects in a solution is stored in a file with the extension `.sln`. When you create a project, a new solution is created automatically, unless you elect to add the project to an existing solution. The `.suo` file is not that important. You can even delete the `.suo` file and Visual C++ will re-create it when opening the solution.

When you create a project along with a solution, you can add projects to the same solution. You can add any kind of project to an existing solution, but you will usually add only projects that are 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

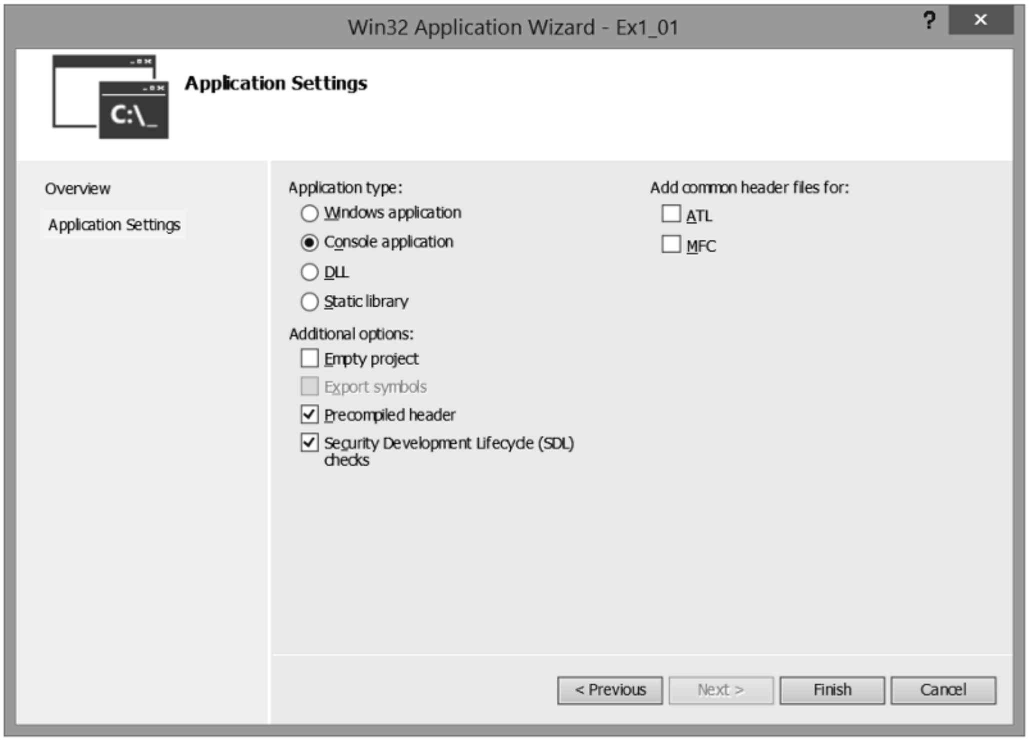
The first step in writing a Visual C++ program is to create a project for it using the File ➪ New ➪ Project menu option from the main menu or by pressing Ctrl+Shift+N. You can also simply click New Project on the Start page. As well as containing files that define the code and any other data that makes up your program, the project XML file in the project folder also records the options you've set for the project. 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

First, select File ➪ New ➪ Project, or use one of the other possibilities I mentioned earlier to bring up the New Project dialog. The left pane in the dialog displays the types of projects you can create; in this case, click `Win32`. This selection identifies an application wizard that creates the initial contents for the project. The right pane displays a list of templates for the project type you have chosen in the left pane. The template you select is used to create the files that make up the project. In the next dialog you can customize the files that are created when you click the OK button in this dialog. For most type/template combinations a basic set of source files is created automatically. Choose Win32 Console Application in this instance.

Enter a suitable name for your project by typing into the Name: text box — for example, you could call it Ex1_01, or you can choose your own project name. Visual C++ supports long filenames, so you have a lot of flexibility. The name of the solution folder appears in the bottom text box and by default it is the same as the project name. You can change this if you prefer. The dialog also enables you to modify the location for the solution that contains your project — this appears in the Location: text 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: text box. By default the solution folder is created for you if it doesn't already exist. To specify a different path for the solution folder, just enter it in the Location: text box. Alternatively, you can use the Browse button to select a path for your solution. Clicking OK displays the Win32 Application Wizard dialog.

This dialog explains the settings currently in effect. You can click Application Settings on the left to display the Application Settings page of the wizard, shown in Figure 1-5.


Win32 Application Wizard - Ex1_01

Application Settings

Application type:
- ○ Windows application
- ● Console application
- ○ DLL
- ○ Static library

Additional options:
- ☐ Empty project
- ☐ Export symbols
- ☑ Precompiled header
- ☑ Security Development Lifecycle (SDL) checks

Add common header files for:
- ☐ ATL
- ☐ MFC

FIGURE 1-5

The Application Settings page enables you to choose options that apply to the project. You can see that you are creating a console application and not a Windows application. The Precompiled header option is a facility for compiling header files such as those from the standard library that do not change frequently. When you compile your program after making changes or additions to your code, the pre-compiled code that has not been changed will be reused as is. This makes compiling your program faster. You can uncheck the Security Development Lifecycle Checks checkbox option; this feature adds functionality for managing large-scale professional projects and we won't be using these. On the right of the dialog there are options for using MFC, which I have mentioned, and ATL, which is outside the scope of this book. For this project you can leave the rest of the options as they are and click Finish. The application wizard will create the project with default files.

The project folder will have the name that you supplied as the project name and will hold 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 four 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 `.sdf` that records data about IntelliSense for the solution. IntelliSense is the facility that I mentioned earlier that provides auto-completion and prompts you for code in the Editor window as you enter it.
- A file with the extension `.opensdf` that records information about the state of the project. This file exists only while the project is open.

If you use Windows Explorer to look in the `Ex1_01` project folder, you will see that there are seven files initially, including a file with the name `ReadMe.txt` that contains a summary of the contents of the files that have been created. The project will automatically open with the Solution Explorer pane, as in Figure 1-6.

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

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

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

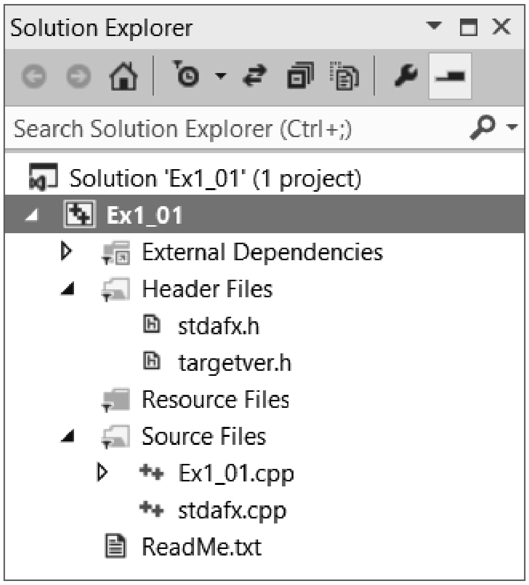


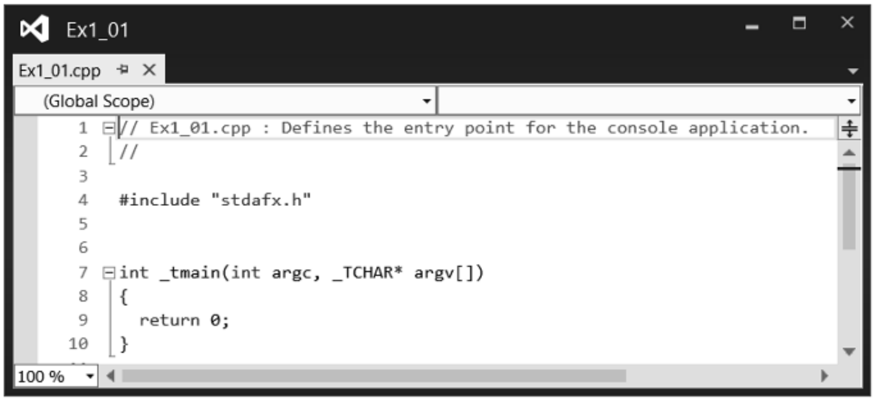
FIGURE 1-6

If it's not already visible, you can display the Resource View tab by selecting from the View menu or by pressing Ctrl+Shift+E. Resource View shows the dialog boxes, icons, menus, toolbars, and other resources used by the project. Because this is a console program, no resources are used; 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.

As with most elements of the 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 when you right-click in the empty space in the tab. If you find that the Solution Explorer pane is in the way when you're writing code, you can hide it by clicking the Auto Hide 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. 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 is the main source file for the program and is shown in Figure 1-7.



```
 1  // Ex1_01.cpp : Defines the entry point for the console application.
 2  //
 3
 4  #include "stdafx.h"
 5
 6
 7  int _tmain(int argc, _TCHAR* argv[])
 8  {
 9    return 0;
10  }
```

FIGURE 1-7

If the line numbers are not displayed, select Tools ➪ Options from the main menu to display the Options dialog. If you extend the C/C++ option in the Text Editor subtree in the left pane and select General from the extended tree, you can check the Line numbers option in the right pane of the dialog. I'll give you a rough guide to what this code in Figure 1-7 does, and you'll see more on all of this 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 with `//`.

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

Line 7 is the first line of the executable code in this file and the beginning of the function called `_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 `_tmain()` function. The executable code is just the single line 9, and this ends 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 new lines you should add are shown in bold; the others are generated for you. To introduce each new line, place the cursor at the end of 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 library files to the `Ex1_01.cpp` 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 be displayed, but you can fix this by right-clicking in the toolbar area and selecting the Build toolbar from those in the list. The program should compile successfully. If there are errors, it may be that you created them 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. You'll see a new subfolder to the solution folder `Ex1_01` called `Debug`. This is the folder `Ex1_01\Debug`, not the folder `Ex1_01\Ex1_01\Debug`. This folder contains the output of the build you just performed. Notice that this folder contains three files.

The `.exe` file is your program in executable form. You don't need to know much about what's in the other files. In case you're curious, the `.ilk` file is used by the linker when you build your project. It enables the linker to incrementally link object files produced from modified source code into the existing `.exe` file. This avoids the need to relink everything each time you change the program. The `.pdb` file contains debugging information that is used when you execute the program in debug mode. In this mode, you can dynamically inspect information generated during program execution.

There's a `Debug` subdirectory in the `Ex1_01` project folder too. This contains a large number of files that were created during the build process, and you can see what kind of information they contain from the Type description in Windows Explorer.

Debug and Release Versions of Your Program

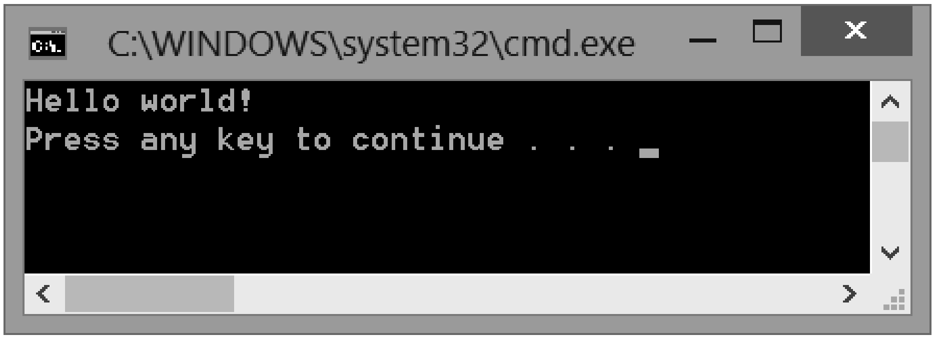
You can set options for a project through the Project ➪ Ex1_01 Properties menu item. These options determine how your source code is processed during the compile and link stages. The set of options that produces a particular executable version of your program is called a *configuration*. When you create a new project workspace, Visual C++ automatically creates configurations for producing two versions of your application. The Debug version includes additional information that helps you debug the program. With the Debug version of a program, you can step through the code when things go wrong, checking on the data values in the program as you go. The Release version has no debug information included and has the code-optimization options for the compiler turned on to provide 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 from the drop-down list in the toolbar. Selecting Configuration Manager from the drop-down list will display the Configuration Manager dialog. You use this dialog when your solution contains multiple projects. Here you can choose configurations for each of the projects and choose which ones you want to build.

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 the program by pressing Ctrl+F5. You should see the window shown in Figure 1-8.



```
Hello world!
Press any key to continue . . .
```

FIGURE 1-8

As you see, you get the text between the double quotes written to the command line. The "`\n`" that was 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. The last line prompting how you continue always appears with console program output. Pressing Enter will close the window. I won't show this last line when I show output from a program in the book.

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 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 it won't be necessary for most 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 OK, click Application Settings on the left side of the dialog box that follows. You can then select Empty project from the additional options and uncheck SDL. When you click Finish, the project is created as before, but this time without any source files.

Next, you can 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 displays: click Code in the left pane and C++ File(.cpp) in the right pane. Enter the filename as **`Ex1_02`**.

When you click Add in the dialog, the new file is added to the project and is displayed in the Editor window. The file is empty, of course, 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 change the indenting by selecting the Tools ➪ Options... menu item to display the Options dialog. Selecting Text Editor ➪ C/C++ ➪ Tabs in the left pane of the dialog displays the indenting options in the right pane. The editor inserts tabs by default, but you can change it to insert spaces if you prefer.

You'll 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 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/IEC standard C++ programs start execution in a function called `main()`. Microsoft uses `wmain` for this function when Unicode characters are used, and the name `_tmain` is defined to be either `main` or `wmain` (in the `tchar.h` header file), depending on whether or not the program is going to use Unicode characters. In the previous example the name `_tmain` is defined behind the scenes to be `wmain` because the project settings were set to Unicode. I'll use the standard name `main` in all the 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`, 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 follows:

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

This is not identical to using `std::endl` though. Using `std::endl` writes a newline and then flushes the output buffer. Using just `\n`, the buffer is not flushed immediately.

The last statement is the `return` statement that ends `main()` and thus the program. This is not strictly necessary here and you could leave it out. If execution reaches the end of `main()` without encountering a return statement, it is equivalent to executing `return 0`.

You can 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 build the project. When you have compiled the program successfully, press Ctrl+F5 to execute it. If everything works as it should, the output will be as follows:

```
This is a simple program that outputs some text.
You can output more lines of text
just by repeating the output statement like this.
```

Note that pressing Ctrl+F5 will build the project before executing it if it is not up to date.

Dealing with Errors

Of course, if you didn't type the program correctly, you get errors reported. To see 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 will include the following 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, but 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.

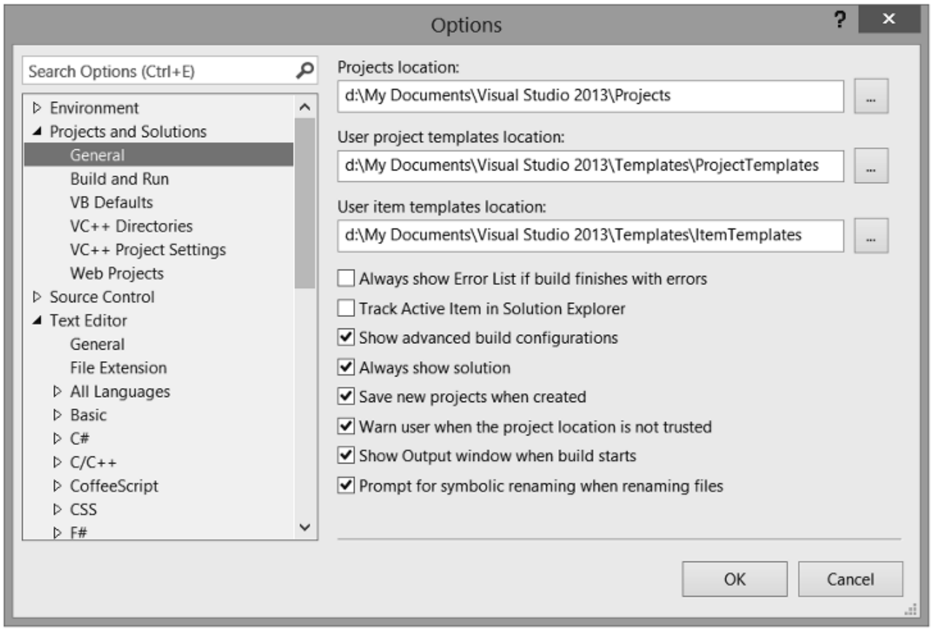
When you have corrected the error, you can rebuild the project. The build operation works efficiently because the project definition keeps track of the status of the files making up the project. During a normal build, Visual C++ recompiles only the files that have changed since the program was last compiled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

Setting Options in Visual C++

Two sets of options are available. You can set options that apply to the tools provided by Visual C++, which apply in every project context. You also can set options that are specific to a project that determine how the project code is to be processed when it is compiled and linked. Options that apply to every project are set through the Options dialog that's displayed when you select Tools ➪ Options from the main menu. You used this dialog earlier to change the code indenting used by the editor. The Options dialog box is shown in Figure 1-9.

Clicking the arrow symbol to the left of any of the items in the left pane displays a list of subtopics. Figure 1-9 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 selected 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 (the one with the question mark) 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 1-9. Just set the path to the location where you want your projects and solutions stored.



Projects location:
d:\My Documents\Visual Studio 2013\Projects

User project templates location:
d:\My Documents\Visual Studio 2013\Templates\ProjectTemplates

User item templates location:
d:\My Documents\Visual Studio 2013\Templates\ItemTemplates

- [ ] Always show Error List if build finishes with errors
- [ ] Track Active Item in Solution Explorer
- [x] Show advanced build configurations
- [x] Always show solution
- [x] Save new projects when created
- [x] Warn user when the project location is not trusted
- [x] Show Output window when build starts
- [x] Prompt for symbolic renaming when renaming files

FIGURE 1-9

You can set options that apply to every project by selecting the Projects and Solutions ➪ VC++ Project Settings topic in the left pane of the Options dialog. You set options specific to the current project through the dialog that displays when you select the Project ➪ Ex1_02 Properties menu item in the main menu, or by pressing Alt+F7. The label for this menu item 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, you can now create a working Windows application. I'll defer discussion of the program until I've covered the necessary ground for you to understand it in detail. You will see, though, that the processes are straightforward.

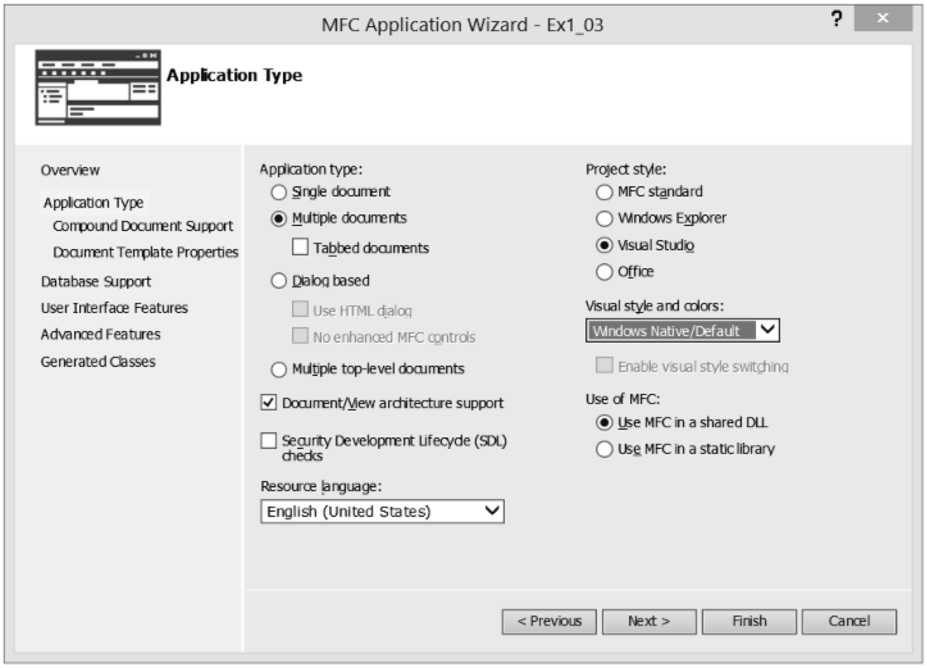
Creating an MFC Application

To start with, if an existing project is active — as indicated by the project name appearing in the title bar of the Visual C++ 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. Create directory for solution is selected by default in the New Project dialog.

To create the Windows program, select New ➪ Project from the File menu or press Ctrl+Shift+N; then set the project type as MFC in the left pane, and select MFC Application as the project template. Enter the project name as `Ex1_03`. When you click OK, the MFC Application Wizard dialog is displayed. The dialog has options for the features you can include in your application. These are identified by the items in the list on the left of the dialog.

Click Application Type to display these options. Click the Tabbed documents option to deselect it and select Windows Native/Default from the drop-down list to the right. The dialog should then look as shown in Figure 1-10.

Click Advanced Features next, and deselect all the options except for the Printing and Print Preview and Common Control Manifest options so that the dialog looks as shown in Figure 1-11. Note how the small image at the top left of the dialog changes as you check or uncheck options.


MFC Application Wizard - Ex1_03

Application Type

Application type:
- ( ) Single document
- [✓ Multiple documents]
  - [ ] Tabbed documents
- ( ) Dialog based
  - [ ] Use HTML dialog
  - [ ] No enhanced MFC controls
- ( ) Multiple top-level documents

[✓] Document/View architecture support

[ ] Security Development Lifecycle (SDL) checks

Resource language: English (United States)

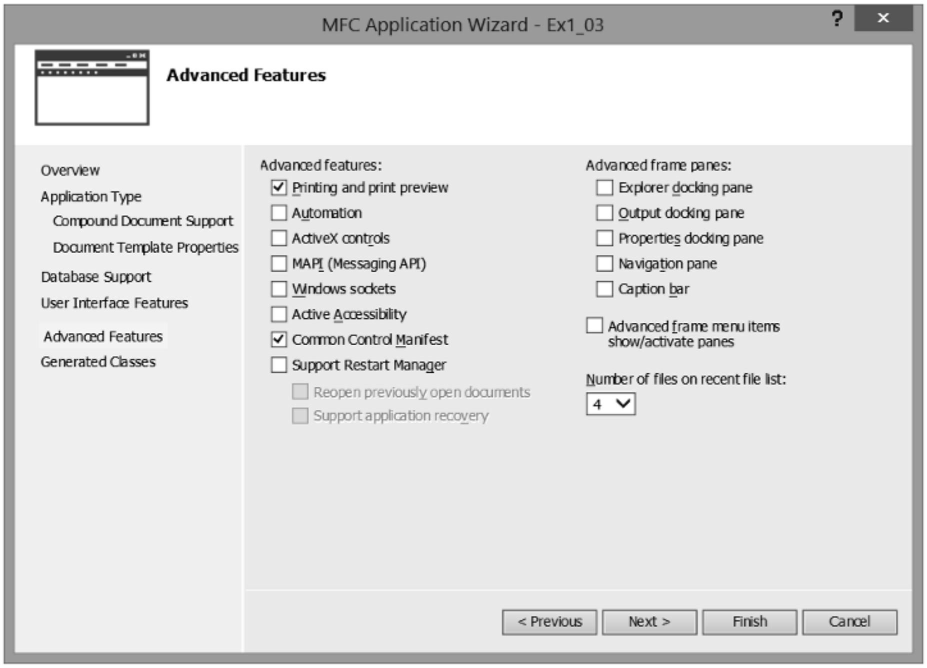
Project style:
- ( ) MFC standard
- ( ) Windows Explorer
- [✓ Visual Studio]
- ( ) Office

Visual style and colors: Windows Native/Default

[ ] Enable visual style switching

Use of MFC:
- [✓ Use MFC in a shared DLL]
- ( ) Use MFC in a static library

FIGURE 1-10



MFC Application Wizard - Ex1_03

Advanced Features

Advanced features:
- [✓] Printing and print preview
- [ ] Automation
- [ ] ActiveX controls
- [ ] MAPI (Messaging API)
- [ ] Windows sockets
- [ ] Active Accessibility
- [✓] Common Control Manifest
- [ ] Support Restart Manager
  - [ ] Reopen previously open documents
  - [ ] Support application recovery

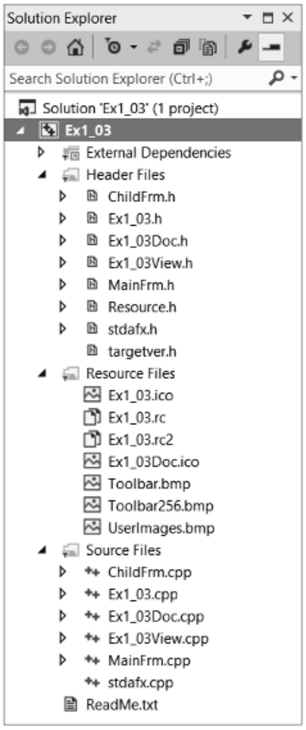
Advanced frame panes:
- [ ] Explorer docking pane
- [ ] Output docking pane
- [ ] Properties docking pane
- [ ] Navigation pane
- [ ] Caption bar
- [ ] Advanced frame menu items show/activate panes

Number of files on recent file list: 4

FIGURE 1-11

Finally, click Finish to create the project. The undocked Solution Explorer pane in the IDE window will look like Figure 1-12. The list shows the large number of source files that have been created, and several resource files. The files with the extension `.cpp` contain executable C++ source code, and the `.h` files, called header files, contain C++ code for definitions such as classes that are used by the executable code. The `.ico` files contain icons. The files are grouped into subfolders in the Solution Explorer pane for ease of access. These aren't real folders though, so they won't appear in the project folder on your disk.

If you look at the contents of the `Ex1_03` solution folder and subfolders using Windows Explorer, you'll notice that you have generated a large number of files. Four of these are in the solution folder that includes the transient `.opensdf` file, there are many more in the project folder, and the rest are in the `res` subfolder of the project folder. The files in the `res` subfolder contain the resources used by the program, such as the toolbars and icons. You get all this as a result of just entering the name for the project. You can see why, with so many files and filenames being created automatically, a separate folder for each project becomes more than just a good idea.



```
Solution 'Ex1_03' (1 project)
  Ex1_03
    External Dependencies
    Header Files
      ChildFrm.h
      Ex1_03.h
      Ex1_03Doc.h
      Ex1_03View.h
      MainFrm.h
      Resource.h
      stdafx.h
      targetver.h
    Resource Files
      Ex1_03.ico
      Ex1_03.rc
      Ex1_03.rc2
      Ex1_03Doc.ico
      Toolbar.bmp
      Toolbar256.bmp
      UserImages.bmp
    Source Files
      ChildFrm.cpp
      Ex1_03.cpp
      Ex1_03Doc.cpp
      Ex1_03View.cpp
      MainFrm.cpp
      stdafx.cpp
    ReadMe.txt
```

FIGURE 1-12

One of the files in the `Ex1_03` project directory is `ReadMe.txt`, and it provides an explanation of the files that the MFC Application Wizard has generated. You can view this file in the Editor window by double-clicking it in the Solution Explorer pane.

Building and Executing the MFC Application

Before you can execute the program, you have to build the project — that is, compile the source code and link the program modules, exactly as 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 program executes. The application window for the program is shown in Figure 1-13.

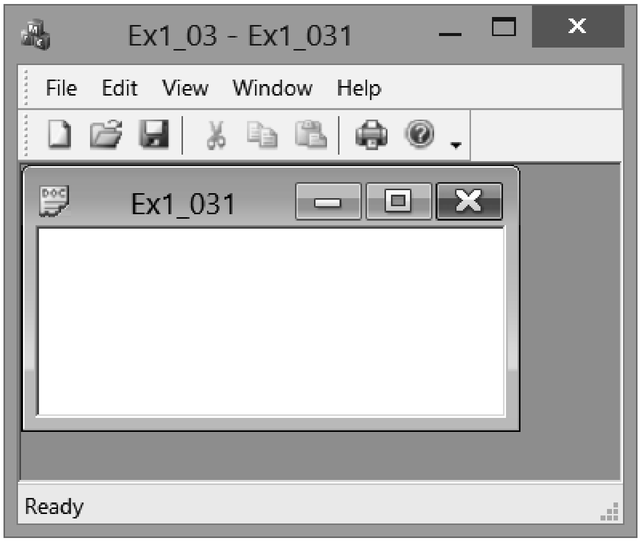


FIGURE 1-13

As you see, the window is complete with menus and a toolbar. Although there is no specific functionality in the program — you must add code for that 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 it comes 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++, required at least a couple of months on a brain-enhancing fish diet before making the attempt. That's why so many programmers used to eat sushi. That's all gone now with Visual C++. However, you never know what's around the corner in programming technology. If you like sushi, it's best to continue eating it — just to be on the safe side.

SUMMARY

In this chapter you have run through the basic mechanics of using Visual C++ to create applications. You created and executed console programs, and with the help of the application wizard you created an MFC-based Windows program. You should be reasonably comfortable with creating and executing projects.

Starting with the next chapter, all the examples illustrating how C++ language elements are used are executed using Win32 console applications. You will return to the application wizard for MFC-based programs as soon as you have finished learning the basics of C++.

➤ WHAT YOU LEARNED IN THIS CHAPTER

TOPIC	CONCEPT
C++	Visual C++ supports C++ that conforms to the C++ 11 language standard that is defined in the document ISO/IEC 14882:2011. Visual C++ implements most of the language features defined by this standard and some features from the draft of the next standard, C++ 14.
Solutions	A solution is a container for one or more projects that form a solution to an information-processing problem of some kind.
Projects	A project is a container for the code and resource elements that make up a functional unit in a program.
Project View Panes	The Solution Explorer pane displays one or more tabs and shows the project files. The Class View pane shows classes in the project. The Resource View pane shows project resources.
Project Options	You can display and modify the options that apply to all C++ projects through the dialog that is displayed when you select Options from the Tools menu.
Project Properties	You can set values for properties for the current project through the dialog that is displayed when you select Properties from the Project menu.
Console Applications	A console application is a basic C++ application with no GUI. Typically, input is from the keyboard and output is to the command line.
The `main()` function	The starting point for a standard C++ program is the `main()` function. The New Project dialog generates a console application that starts with the `_tmain()` function.
Unicode	If you want to use the standard `main()` function in a console program, you can generate an empty Win32 project and add the source file for `main()`.
Windows Store Apps	Windows Store apps target tablet computers and desktop PCs running the Windows 8 operating system.
Windows Runtime	The Windows Runtime, WinRT, provides the interface to the Windows 8 and later operating systems for Windows Store apps.
Windows Desktop Applications	Windows desktop applications have an application window and a GUI incorporating controls such as menus, toolbars, and dialogs. Desktop applications interface to the operating system through the Win32 set of functions. Desktop applications execute under Windows 7 and Windows 8 and it successors.
The Microsoft Foundation Classes	The MFC is a set of C++ classes that encapsulate the functions provided by Win32. MFC makes it easier to develop Windows desktop applications.
MFC Projects	You create an MFC project by selecting MFC then MFC Application in the New Project dialog.