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/remtitle.cgi?isbn=9781118368084`. 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 8 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 8 apps are different from desktop applications. They 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 8 apps execute only with Windows 8 and you must have Visual Studio 2012 installed under Windows 8 to develop them. Windows 8 apps communicate with the operating system through the Windows Runtime, WinRT. I'll introduce you to programming Windows 8 applications in Chapter 18.

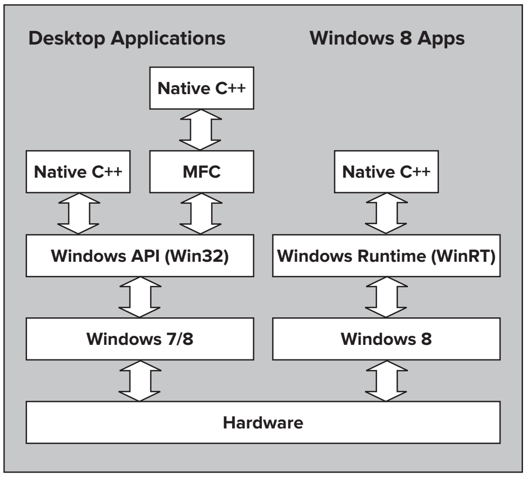


FIGURE 1-1

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 application programs.

NOTE *As I'll explain in Chapter 18, Windows 8 apps are different. You specify the GUI in XAML, and the XAML is used to generate the C++ program 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 does not support all the new language features introduced by this latest standard, just some of the most commonly used features, but it will surely be extended over time. 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. ISO/IEC standard 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.

Console Applications

Visual C++ *console applications* enable you to write, compile, and test C++ programs that have none of the baggage required for Windows programs. 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.

When you write console applications, it might seem as if you are being sidetracked from the main objective of programming applications for Windows, but 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 the C++ language. In the early chapters of the book that 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.

Windows Programming Concepts

The project creation facilities provided with Visual C++ can generate skeleton code for a wide variety of C++ application programs automatically. A Windows program has a different structure from that of the typical console program that you execute from the command line and it's 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 active at one time under Windows, so the operating system 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 always has primary control of 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 on one 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 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.

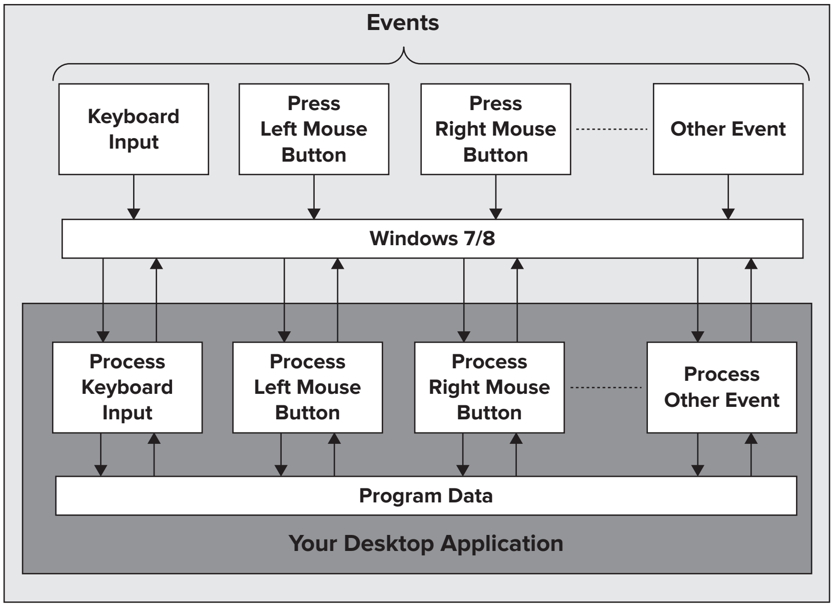


FIGURE 1-2

Each block within the Desktop Application 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 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 external events such as the selection of a menu or a mouse click, all typically have access to a common set of application-specific data in a particular program. 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 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 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 navigate around sacks of code. Fortunately, Visual C++ comes with an environment that is designed specifically for the purpose.

WHAT IS THE INTEGRATED DEVELOPMENT ENVIRONMENT?

The integrated development environment (IDE) that comes with Visual C++ 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++ 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, you can 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.

The fundamental parts of Visual C++, 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.

The Editor

The editor provides an interactive environment in which 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. Another very helpful feature is *IntelliSense*. IntelliSense analyzes the code as you enter it, and underlines anything that is incorrect with a red squiggle. It can also provide prompts where the options for what you need to enter next in the code can be determined.

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 can detect 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 output from the compiler is stored in files called *object files* that 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 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 nonexistent library component is referenced.

The Libraries

A *library* is simply a collection of prewritten 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 implemented by the various libraries provided by Visual C++ greatly enhance productivity by saving you the effort of writing and testing the code for such operations yourself.

The *Standard C++ Library* defines a basic set of facilities that are common to all ISO/IEC standard-conforming C++ compilers. It contains a wide range of commonly used routines, including numerical functions, such as the calculation of square roots and the evaluation of 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 get to know quite a number of these as you develop your knowledge of C++.

Window-based 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.)

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.

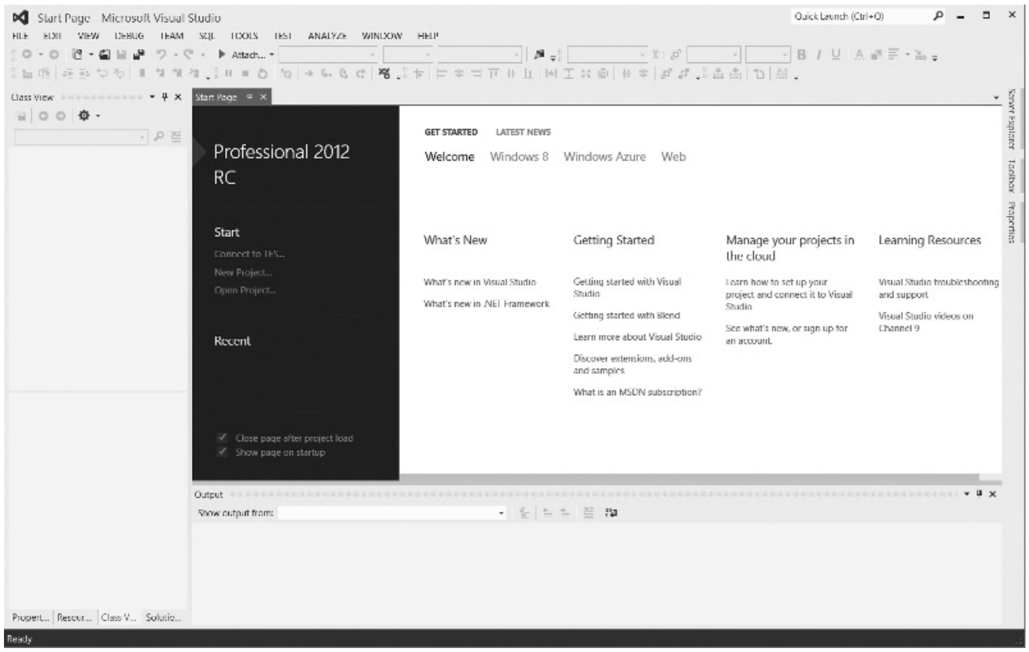

Start Page - Microsoft Visual Studio
Quick Launch (Ctrl+Q)
FILE EDIT VIEW DEBUG TEAM SQL TOOLS TEST ANALYZE WINDOW HELP
Attach...
Class View
Start Page
Professional 2012 RC
Start
Connect to TFS...
New Project...
Open Project...
Recent
Close page after project load
Show page on startup
GET STARTED
LATEST NEWS
Welcome
Windows 8
Windows Azure
Web
What's New
What's new in Visual Studio
What's new in .NET Framework
Getting Started
Getting started with Visual Studio
Getting started with Blend
Learn more about Visual Studio
Discover extensions, add-ons and samples
What is an MSDN subscription?
Manage your projects in the cloud
Learn how to set up your project and connect it to Visual Studio
See what's new, or sign up for an account
Learning Resources
Visual Studio troubleshooting and support
Visual Studio videos on Channel 9
Server Explorer
Toolbox
Properties
Output
Show output from:
Propert... Resour... Class V... Solutio...
Ready

FIGURE 1-3

The pane to the left in Figure 1-3 is the *Solution Explorer window*, the top right 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 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 can display other tabs (only three are shown in Figure 1-3), 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 the output from build operations in 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 generally be undocked from its position in the Visual C++ 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.

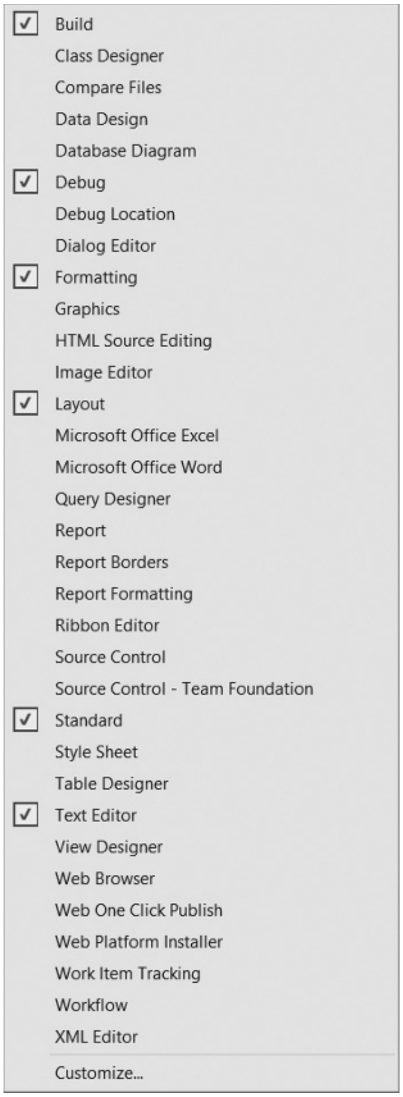


FIGURE 1-4

Toolbar Options

You can choose which toolbars are displayed in your Visual C++ window by right-clicking in the toolbar area. The range of toolbars in the list depends on which edition of Visual Studio 2012 you have installed. A pop-up menu with a list of toolbars (Figure 1-4) appears, and the toolbars currently displayed have checkmarks alongside them.

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, 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 its being displayed; clicking a toolbar that is selected/deselects it and hides the toolbar.

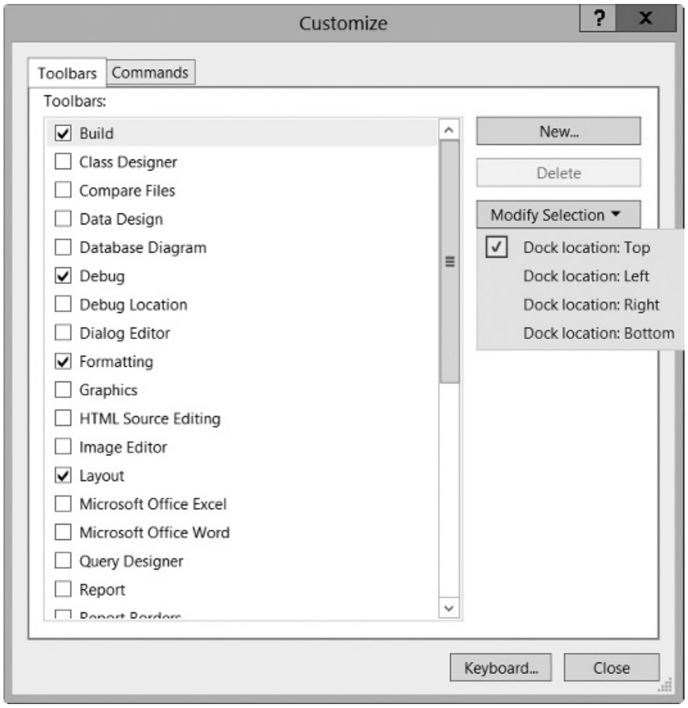
NOTE *A toolbar won't necessarily display all the buttons that are available for it. You can add or remove buttons for a toolbar by clicking the down arrow that appears at the right of the button set. The buttons in the TextEditor toolbar that indent and unindent a set of highlighted statements are particularly useful, as are the buttons that comment out or uncomment a highlighted set of statements.*

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 visible toolbars whenever it suits you by right-clicking in the toolbar area and choosing from the context menu.

> ***NOTE*** *As in many other Windows applications, the toolbars that make up Visual C++ come complete with tooltips. Just let the mouse pointer linger over a toolbar button for a second or two, and a white 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. You can arrange for any of the toolbars to be docked at any of the four sides of the application window. If you right-click in the toolbar area and select Customize from the pop-up, the Customize dialog will be displayed. 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 to dock the toolbar where you want. Figure 1-5 shows how the dialog looks after the user selects the Build toolbar on the left and clicks the Modify Selection drop-down list.



Toolbars:

- [✓] Build
- [ ] Class Designer
- [ ] Compare Files
- [ ] Data Design
- [ ] Database Diagram
- [✓] Debug
- [ ] Debug Location
- [ ] Dialog Editor
- [✓] Formatting
- [ ] Graphics
- [ ] HTML Source Editing
- [ ] Image Editor
- [✓] Layout
- [ ] Microsoft Office Excel
- [ ] Microsoft Office Word
- [ ] Query Designer
- [ ] Report
- [ ] Report Borders

- [✓] Dock location: Top
- Dock location: Left
- Dock location: Right
- Dock location: Bottom

FIGURE 1-5

You'll recognize many of the toolbar icons that Visual C++ 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++.

Documentation

There will be plenty of occasions when you'll want to find out more information about Visual C++ and its features and options. Press Ctrl+F1 to access the product documentation. With the cursor on elements in your code that are part of the C++ language or a standard library item, pressing F1 will usually open 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 — 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*; detailed information about the project is stored in an XML file with the extension `.vcxproj`, 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 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 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, and 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`, respectively. 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 will usually add only a project 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 all the code and any other data that makes up your program, the project XML file in the project folder also records the Visual C++ options you're using. That's enough introductory stuff for the moment. It's time to get your hands dirty.

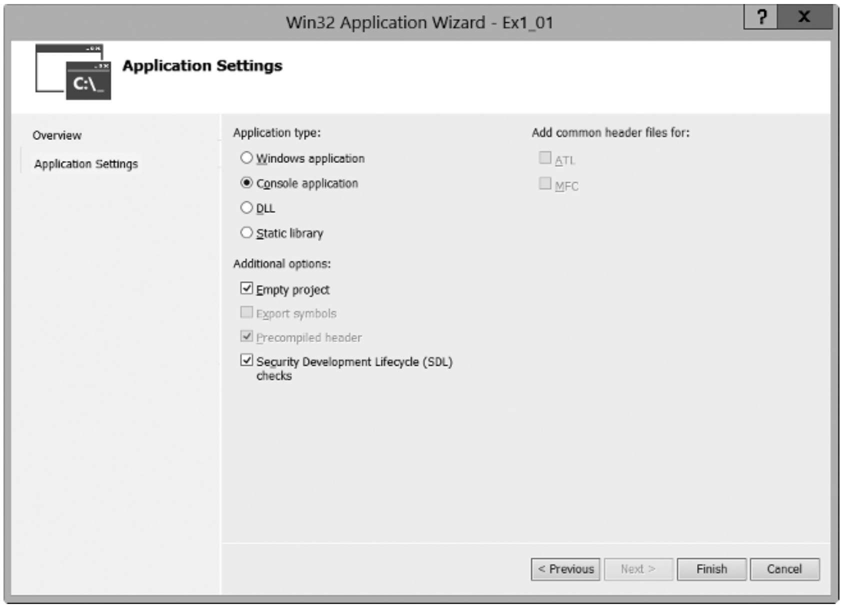
TRY IT OUT Creating a Project for a Win32 Console Application

You'll now take a look at creating a project for a console application. First, select File ➪ New ➪ Project or use one of the other possibilities mentioned earlier to bring up the New Project dialog box.

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 in 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 is created automatically. You can choose Win32 Console Application in this instance.

You can now enter a suitable name for your project by typing into the Name: text box — for example, you could call this one 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, the solution folder has the same name as the project. You can change this if you want. The dialog box 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. If you want 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 another path for your solution. Clicking OK displays the Win32 Application Wizard dialog box.

This dialog box explains the settings currently in effect. In this case, you can click Application Settings on the left to display the Application Settings page of the wizard, shown in Figure 1-6.


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 1-6

The Application Settings page enables you to choose options that you want to 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 source files such as those from the standard library that do not change just once. When you recompile your program after making changes or additions to your code, the precompiled code will be reused as is. You can see on the right of the dialog options for using MFC, which I have mentioned, and ATL — the Application Template Library — which is outside the scope of this book. Here, you can leave things as they are and click Finish. 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 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 for the project. The project you have created will automatically open in Visual C++ with the Solution Explorer pane, as in Figure 1-7.

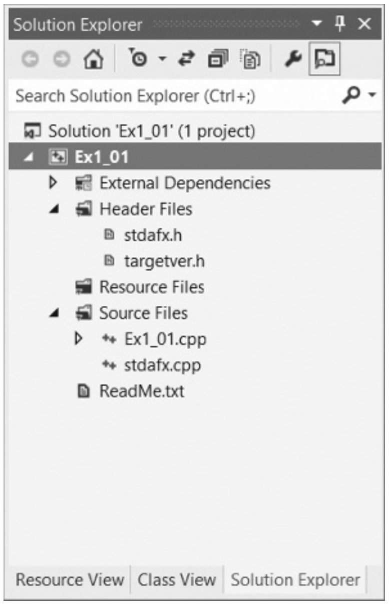


FIGURE 1-7

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 just 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 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 I 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.

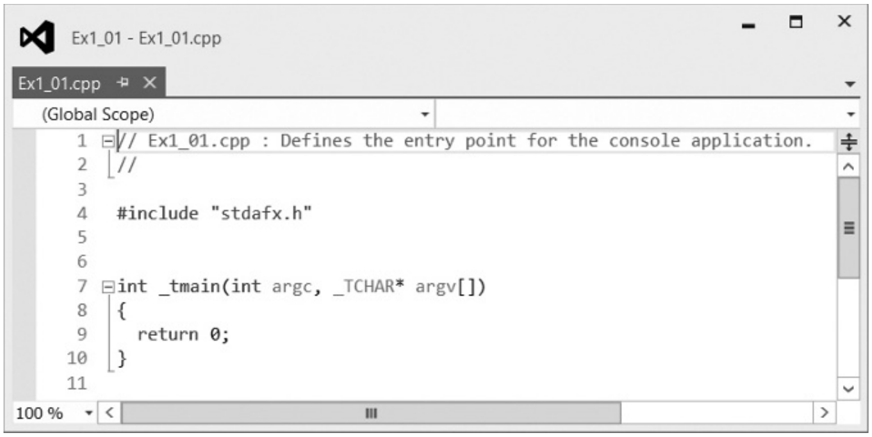
You can also display the Property Manager tab by selecting it from the View menu. It shows the properties that have been set for the Debug and Release versions of your project. I'll explain these a little later in this chapter. You can change any of the properties for a version by right-clicking a version 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 we go into the Debug and Release versions of a program.

If it is not already visible, you can display the Resource View tab by selecting from the View menu or by pressing Ctrl+Shift+E. The Resource View shows the dialog boxes, icons, menus, toolbars, and other resources 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.

As with most elements of the Visual C++ 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, too. If you find that the Solution Explorer pane gets in your 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. 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 is the main source file for the program that the application wizard generated, and is shown in Figure 1-8.



```
 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  }
11
```

FIGURE 1-8

If the line numbers are not displayed on your system, 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 1-8 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 in place of this `#include` directive. This is the standard way to add the contents of `.h` source 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 function `_tmain()`. The executable code is just the single line 9, 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 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 C++ 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 easily 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 to 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 on the project. Notice that this folder contains three 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, the `.ilk` 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 relink everything each time you change your 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.

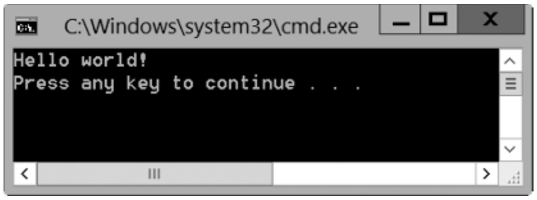
Debug and Release Versions of Your Program

You can set 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++ 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 your 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 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 from the drop-down list in the toolbar. If you select Configuration Manager… from the drop-down list, the Configuration Manager dialog will be displayed. 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



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

FIGURE 1-9

After you have successfully compiled the solution, you can execute your program by pressing Ctrl+F5. You should see the window shown in Figure 1-9.

As you can see, you get the text 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

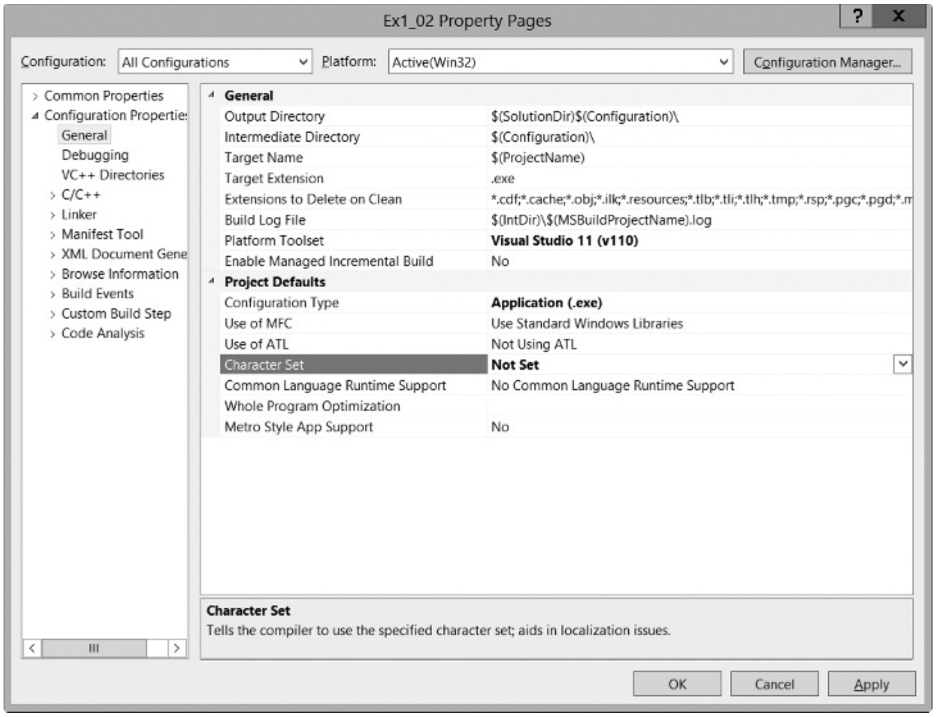
The previous project contained a certain amount of excess baggage that you don't need when working with simple C++ language examples. The precompiled headers option chosen by default resulted in the `stdafx.h` file being created in the project. This is a mechanism for making the compilation process more efficient when there are a lot of files in a program, but 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.

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

By default, the project options will be set to use Unicode libraries. This makes use of a non-standard name for the main function in the program. In order to use standard native C++ in your console programs, you need to switch off the use of Unicode libraries. To do this, select the Project ➪ Properties menu item, or press Alt+F7, to display the Property Pages dialog for the project. Select the All Configurations option from the Configuration: drop-down list at the top. Select the General option under Configuration Properties in the left pane and select the Character Set property in the right pane. You will then be able to set the value of this property to Not Set from the drop-down list to the right of the property name, as shown in Figure 1-10. Click OK to close the dialog. You should do this for all the C++ console program examples in the book. If you forget to do so, they won't build. You will be using Unicode libraries in the Windows examples, though.

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:



General	
Output Directory	$(SolutionDir)$(Configuration)\
Intermediate Directory	$(Configuration)\
Target Name	$(ProjectName)
Target Extension	.exe
Extensions to Delete on Clean	*.cdf;*.cache;*.obj;*.ilk;*.resources;*.tlb;*.tli;*.tlh;*.tmp;*.rsp;*.pgc;*.pgd;*.m
Build Log File	$(IntDir)\$(MSBuildProjectName).log
Platform Toolset	**Visual Studio 11 (v110)**
Enable Managed Incremental Build	No
Project Defaults	
Configuration Type	**Application (.exe)**
Use of MFC	Use Standard Windows Libraries
Use of ATL	Not Using ATL
Character Set	**Not Set**
Common Language Runtime Support	No Common Language Runtime Support
Whole Program Optimization	
Metro Style App Support	No

Character Set
Tells the compiler to use the specified character set; aids in localization issues.

FIGURE 1-10

```
// 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 want.

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/IEC standard 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` (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 `main`. I'll use the standard name `main` in all the C++ examples, which is why you need to change the `Character Set` property value for these projects to `Not Set`.

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

```
    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";
```

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. 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.
```

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:

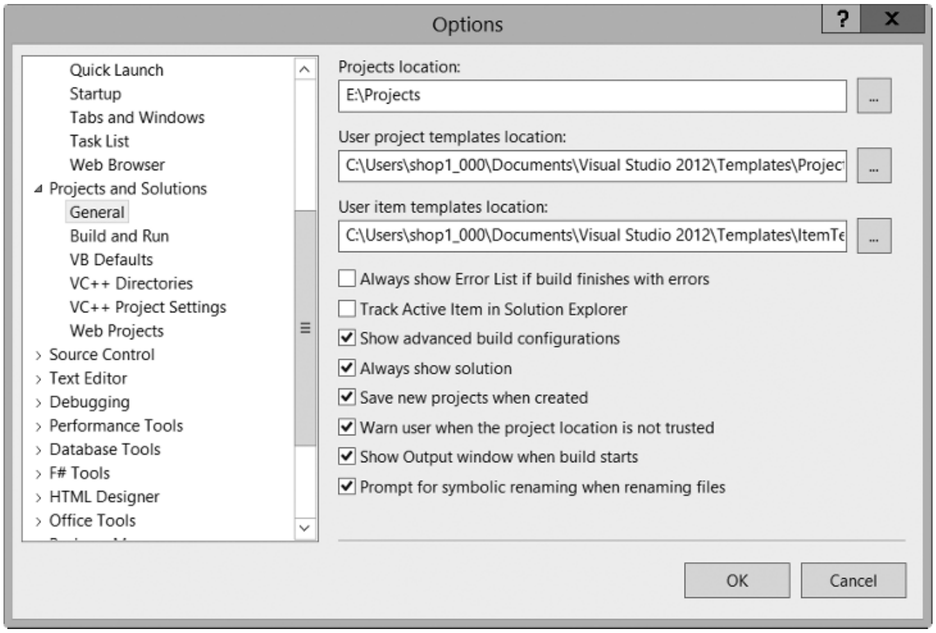
```
C2143: syntax error : missing ';' before 'return'
```

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 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++ 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. If you modify a header file, all files that include that header will be recompiled, along with the header file itself of course.

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, and 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 1-11.



Projects location: E:\Projects

User project templates location: C:\Users\shop1_000\Documents\Visual Studio 2012\Templates\Projec

User item templates location: C:\Users\shop1_000\Documents\Visual Studio 2012\Templates\ItemTe

- [ ] 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 1-11

Clicking the symbol to the left of any of the items in the left pane displays a list of subtopics. Figure 1-11 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.

One option you should set right now relates to how IntelliSense works. Expand Text Editor ➪ C/C++ ➪ Advanced in the Options dialog and select the last property in the IntelliSense group, Member List Commit Characters. Delete the characters that appear as the property value in the right column. This will prevent IntelliSense from selecting wholly inappropriate choices from the list of possibilities by default. Just to make sure, I'll remind you about this again later in the book.

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 1-11. 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 in the Options dialog that is displayed by selecting the Projects and Solutions ➪ VC++ Project Settings topic in the left pane. You can 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. This menu item label is tailored to reflect the name of the current project. You used this to change the value of the Character Set property for the `Ex1_02` console program.

Creating and Executing Windows Applications

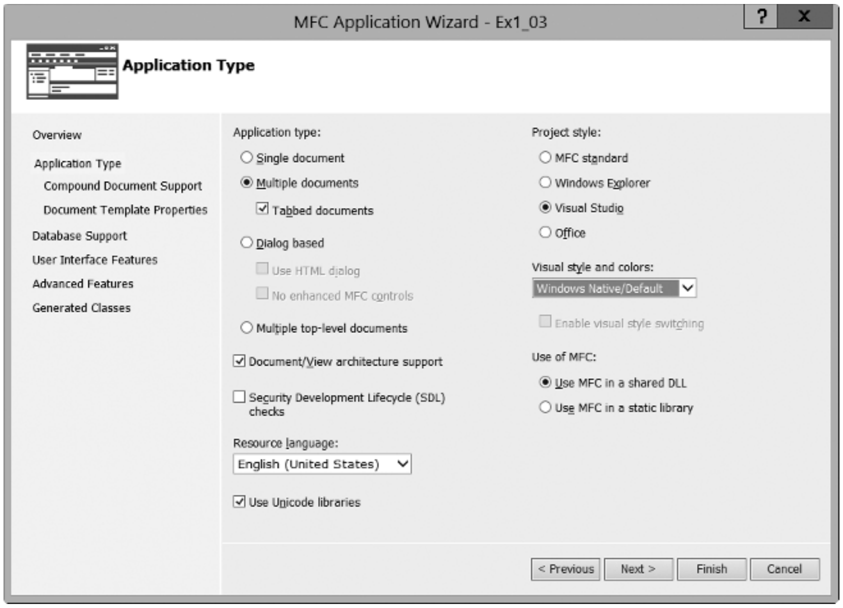
Just to show how easy it's going to be, you can now create a working Windows application. I'll defer discussion of the program that you'll generate until I've covered the necessary ground for you to understand it in detail. You will see, though, that the processes are straightforward.

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++ 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, and select MFC Application as the project template. You can then enter the project name as **`Ex1_03`**. When you click OK the MFC Application Wizard dialog is displayed. The dialog has a range of options that let you choose the features you'd like to have included 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 1-12.



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)

☑ Use Unicode libraries

Project style:

○ MFC standard
○ Windows Explorer
● Visual Studio
○ Office

Visual style and colors: Windows Native/Default

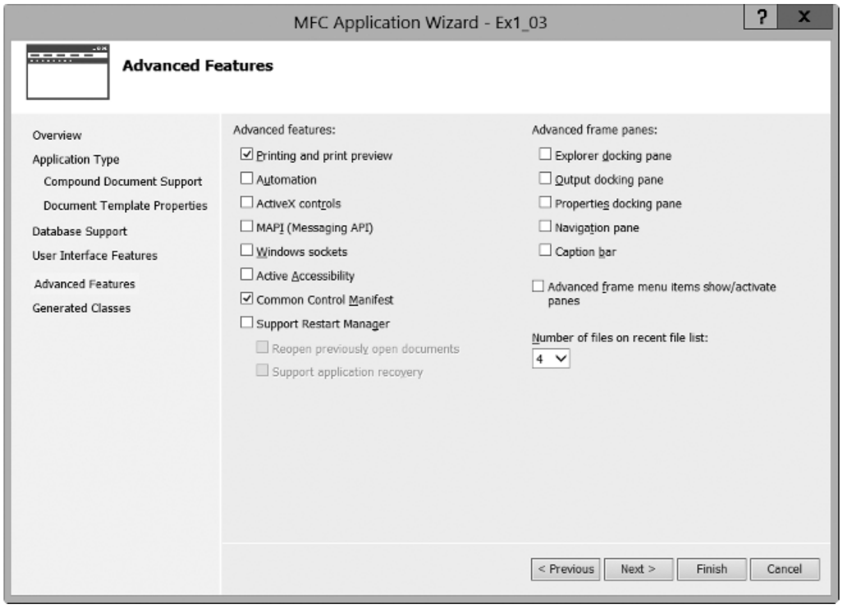
☐ Enable visual style switching

Use of MFC:

● Use MFC in a shared DLL
○ Use MFC in a static library

FIGURE 1-12

Click Advanced Features next, and deselect Explorer docking pane, Output docking pane, Properties docking pane, ActiveX controls, Common Control Manifest, and Support Restart Manager so that the dialog looks as shown in Figure 1-13.



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 1-13

Finally, click Finish to create the project. The undocked Solution Explorer pane in the IDE window will look as shown in Figure 1-14.

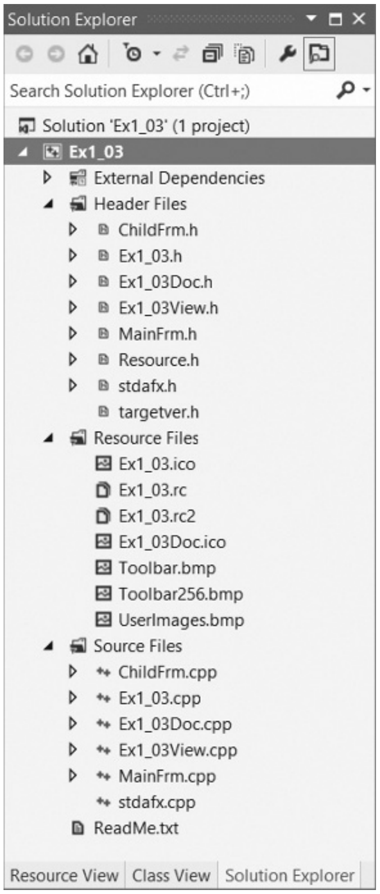


FIGURE 1-14

The list shows the large number of source files that have been created, and several resource files. 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 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_03` solution folder and subfolders using Windows Explorer or whatever else you may have handy for looking at the files on your hard disk, you'll notice that you have generated a total of 28 files. Four of these are in the solution folder that includes the transient `.opensdf` file, a further 19 are in the project folder, and the rest are in a subfolder, `res`, of the project folder. The files in the `res` subfolder contain the resources used by the program, such as the menus and icons. 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 filenames being created automatically, a separate directory for each project becomes more than just a good idea.

One of the files in the `Ex1_03` project directory is `ReadMe.txt`, and it provides an explanation of the purpose of each of the files that the MFC Application Wizard has generated. You can take a look at using Notepad, WordPad, or even the Visual C++ 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 — that is, compile the source code and link the program modules. You do this in exactly the same way as 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 1-15.

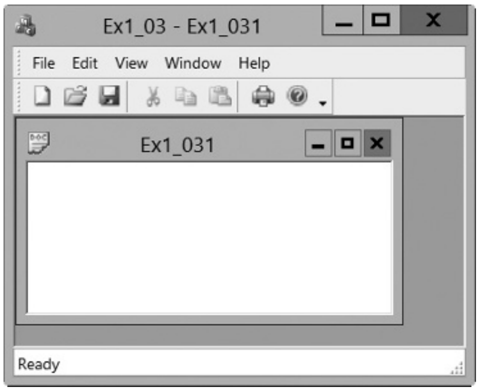


FIGURE 1-15

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 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++. You never know, however, what's around the corner in programming technology. If you like sushi, it's best to continue eating it 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.

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 delving into the secrets of C++.

▶ WHAT YOU LEARNED IN THIS CHAPTER

TOPIC	CONCEPT
C++	Visual C++ supports C++ language statements that conform to the C++ 11 language standard that is defined in the document ISO/IEC 14882:2011. Visual C++ does not yet implement all language features defined by this standard.
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.
The Solution Explorer Pane	The Solution Explorer pane displays one or more tabs showing different aspects of a project. The Solution Explorer tab shows the project files. The Class View tab shows classes in the project. The Resource View tab 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	The default console program uses Unicode characters by default. 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()` after disabling the `Character Set` project property value that selects the Unicode character set.
Windows 8 Apps	Windows 8 Apps target tablet computers and desktop PCs running the Windows 8 operating system.
Windows Runtime	The Windows Runtime, WinRT, provides the interface to the operating system for Windows 8 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 Vista, Windows 7 and Windows 8.
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.