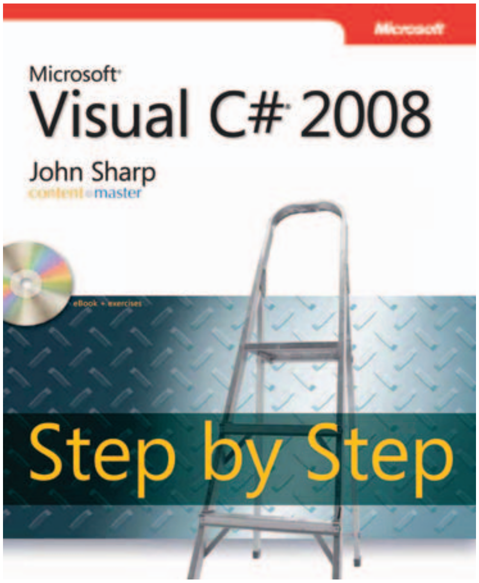


Microsoft® Visual C#® 2008 Step by Step

John Sharp (Content Master)

To learn more about this book, visit Microsoft Learning at
http://www.microsoft.com/MSPress/books/11298.aspx

9780735624306

Table of Contents

Part I Introducing Microsoft Visual C# and Microsoft Visual Studio 2008

Part II Understanding the C# Language

Part III Creating Components

Part IV Working with Windows Applications

Part V Managing Data

Part VI Building Web Applications

Chapter 1

Welcome to C#

After completing this chapter, you will be able to:

- Use the Microsoft Visual Studio 2008 programming environment.
- Create a C# console application.
- Explain the purpose of namespaces.
- Create a simple graphical C# application.

Microsoft Visual C# is Microsoft's powerful component-oriented language. C# plays an important role in the architecture of the Microsoft .NET Framework, and some people have drawn comparisons to the role that C played in the development of UNIX. If you already know a language such as C, C++, or Java, you'll find the syntax of C# reassuringly familiar. If you are used to programming in other languages, you should soon be able to pick up the syntax and feel of C#; you just need to learn to put the braces and semicolons in the right place. Hopefully, this is just the book to help you!

In Part I, you'll learn the fundamentals of C#. You'll discover how to declare variables and how to use arithmetic operators such as the plus sign (+) and minus sign (–) to manipulate the values in variables. You'll see how to write methods and pass arguments to methods. You'll also learn how to use selection statements such as *if* and iteration statements such as *while*. Finally, you'll understand how C# uses exceptions to handle errors in a graceful, easy-to-use manner. These topics form the core of C#, and from this solid foundation, you'll progress to more advanced features in Part II through Part VI.

Beginning Programming with the Visual Studio 2008 Environment

Visual Studio 2008 is a tool-rich programming environment containing all the functionality you need to create large or small C# projects. You can even create projects that seamlessly combine modules compiled using different programming languages. In the first exercise, you start the Visual Studio 2008 programming environment and learn how to create a console application.

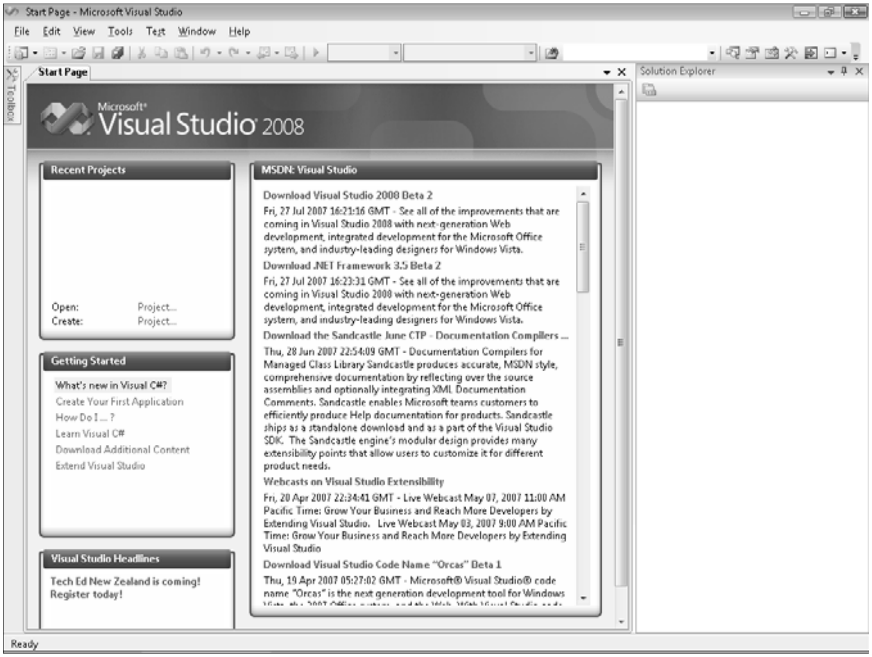
Note A console application is an application that runs in a command prompt window, rather than providing a graphical user interface.

Create a console application in Visual Studio 2008

- If you are using Visual Studio 2008 Standard Edition or Visual Studio 2008 Professional Edition, perform the following operations to start Visual Studio 2008:

  1. On the Microsoft Windows task bar, click the *Start* button, point to *All Programs,* and then point to the *Microsoft Visual Studio 2008* program group.
  2. In the Microsoft Visual Studio 2008 program group, click *Microsoft Visual Studio 2008*.

  Visual Studio 2008 starts, like this:



Download Visual Studio 2008 Beta 2

Fri, 27 Jul 2007 16:21:16 GMT - See all of the improvements that are coming in Visual Studio 2008 with next-generation Web development, integrated development for the Microsoft Office system, and industry-leading designers for Windows Vista.

Download .NET Framework 3.5 Beta 2

Fri, 27 Jul 2007 16:23:31 GMT - See all of the improvements that are coming in Visual Studio 2008 with next-generation Web development, integrated development for the Microsoft Office system, and industry-leading designers for Windows Vista.

Download the Sandcastle June CTP - Documentation Compilers ...

Thu, 28 Jun 2007 22:54:09 GMT - Documentation Compilers for Managed Class Library Sandcastle produces accurate, MSDN style, comprehensive documentation by reflecting over the source assemblies and optionally integrating XML Documentation Comments. Sandcastle enables Microsoft teams customers to efficiently produce Help documentation for products. Sandcastle ships as a standalone download and as a part of the Visual Studio SDK. The Sandcastle engine's modular design provides many extensibility points that allow users to customize it for different product needs.

Webcasts on Visual Studio Extensibility

Fri, 20 Apr 2007 22:34:41 GMT - Live Webcast May 07, 2007 11:00 AM Pacific Time: Grow Your Business and Reach More Developers by Extending Visual Studio. Live Webcast May 03, 2007 9:00 AM Pacific Time: Grow Your Business and Reach More Developers by Extending Visual Studio

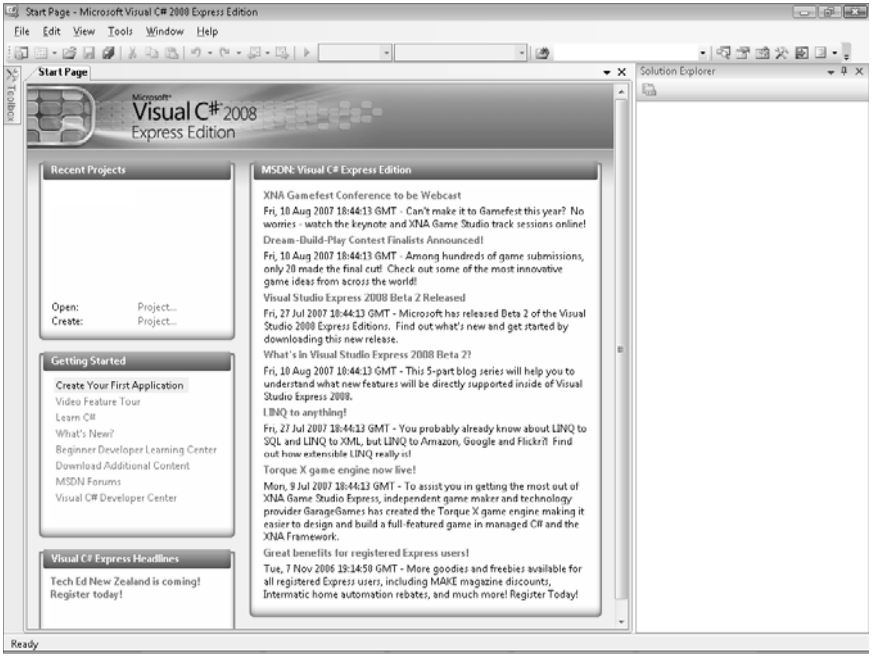
Download Visual Studio Code Name "Orcas" Beta 1

Thu, 19 Apr 2007 05:27:02 GMT - Microsoft® Visual Studio® code name "Orcas" is the next generation development tool for Windows

Note If this is the first time you have run Visual Studio 2008, you might see a dialog box prompting you to choose your default development environment settings. Visual Studio 2008 can tailor itself according to your preferred development language. The various dialog boxes and tools in the integrated development environment (IDE) will have their default selections set for the language you choose. Select *Visual C# Development Settings* from the list, and then click the *Start Visual Studio* button. After a short delay, the Visual Studio 2008 IDE appears.

- If you are using Visual C# 2008 Express Edition, on the Microsoft Windows task bar, click the *Start* button, point to *All Programs*, and then click *Microsoft Visual C# 2008 Express Edition*.

Visual C# 2008 Express Edition starts, like this:


Start Page - Microsoft Visual C# 2008 Express Edition
File
Edit
View
Tools
Window
Help
Toolbox
Start Page
Solution Explorer
Microsoft
Visual C# 2008
Express Edition
Recent Projects
Open: Project...
Create: Project...
Getting Started
Create Your First Application
Video Feature Tour
Learn C#
What's New?
Beginner Developer Learning Center
Download Additional Content
MSDN Forums
Visual C# Developer Center
Visual C# Express Headlines
MSDN: Visual C# Express Edition
Ready

Tech Ed New Zealand is coming! Register today!

XNA Gamefest Conference to be Webcast

Fri, 10 Aug 2007 18:44:13 GMT - Can't make it to Gamefest this year? No worries - watch the keynote and XNA Game Studio track sessions online!

Dream-Build-Play Contest Finalists Announced!

Fri, 10 Aug 2007 18:44:13 GMT - Among hundreds of game submissions, only 20 made the final cut! Check out some of the most innovative game ideas from across the world!

Visual Studio Express 2008 Beta 2 Released

Fri, 27 Jul 2007 18:44:13 GMT - Microsoft has released Beta 2 of the Visual Studio 2008 Express Editions. Find out what's new and get started by downloading this new release.

What's In Visual Studio Express 2008 Beta 2?

Fri, 10 Aug 2007 18:44:13 GMT - This 5-part blog series will help you to understand what new features will be directly supported inside of Visual Studio Express 2008.

LINQ to anything!

Fri, 27 Jul 2007 18:44:13 GMT - You probably already know about LINQ to SQL and LINQ to XML, but LINQ to Amazon, Google and Flickr?! Find out how extensible LINQ really is!

Torque X game engine now live!

Mon, 9 Jul 2007 18:44:13 GMT - To assist you in getting the most out of XNA Game Studio Express, independent game maker and technology provider GarageGames has created the Torque X game engine making it easier to design and build a full-featured game in managed C# and the XNA Framework.

Great benefits for registered Express users!

Tue, 7 Nov 2006 19:14:50 GMT - More goodies and freebies available for all registered Express users, including MAKE magazine discounts, Intermatic home automation rebates, and much more! Register Today!

Note To avoid repetition, throughout this book, I simply state, "Start Visual Studio" when you need to open Visual Studio 2008 Standard Edition, Visual Studio 2008 Professional Edition, or Visual C# 2008 Express Edition. Additionally, unless explicitly stated, all references to Visual Studio 2008 apply to Visual Studio 2008 Standard Edition, Visual Studio 2008 Professional Edition, and Visual C# 2008 Express Edition.

- If you are using Visual Studio 2008 Standard Edition or Visual Studio 2008 Professional Edition, perform the following tasks to create a new console application.

  1. On the *File* menu, point to *New*, and then click *Project*.

     The *New Project* dialog box opens. This dialog box lists the templates that you can use as a starting point for building an application. The dialog box categorizes templates according to the programming language you are using and the type of application.

  2. In the *Project types* pane, click *Visual C#*. In the *Templates* pane, click the *Console Application* icon.

3. In the *Location* field, if you are using the Windows Vista operating system, type **C:\Users\*YourName*\Documents\Microsoft Press\Visual CSharp Step By Step\Chapter 1**. If you are using Microsoft Windows XP or Windows Server 2003, type **C:\Documents and Settings\YourName\My Documents\Microsoft Press\Visual CSharp Step by Step\Chapter 1**.

   Replace the text *YourName* in these paths with your Windows user name.

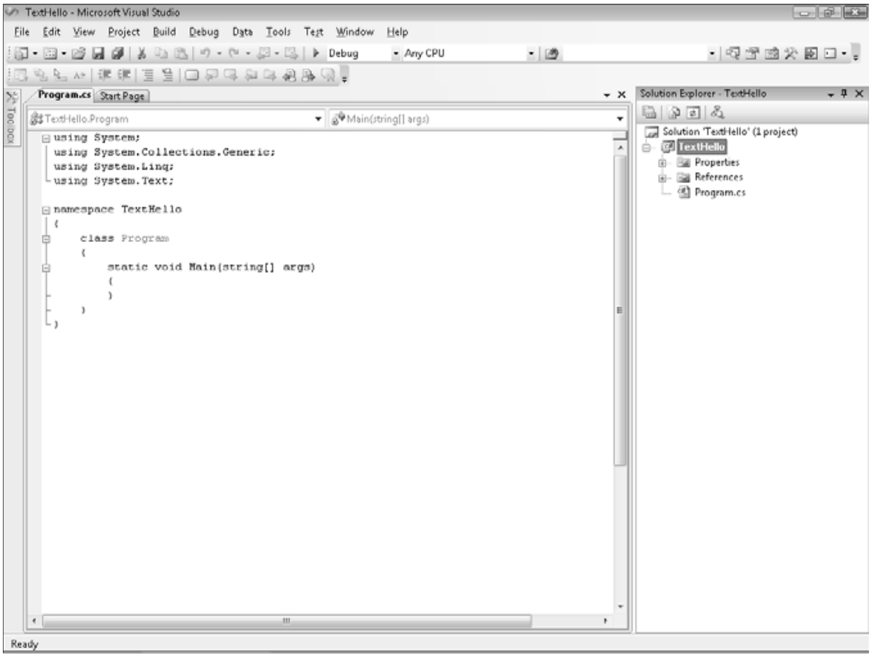
> **Note** To save space throughout the rest of this book, I will simply refer to the path "C:\Users\YourName\Documents" or "C:\Documents and Settings\YourName\My Documents" as your Documents folder.

> **Tip** If the folder you specify does not exist, Visual Studio 2008 creates it for you.

4. In the *Name* field, type **TextHello**.
5. Ensure that the *Create directory for solution* check box is selected, and then click *OK*.

- If you are using Visual C# 2008 Express Edition, the *New Project* dialog box won't allow you to specify the location of your project files; it defaults to the C:\Users\YourName\AppData\Local\Temporary Projects folder. Change it by using the following procedure:
  1. On the *Tools* menu, click *Options*.
  2. In the *Options* dialog box, turn on the *Show All Settings* check box, and then click *Projects and Solutions* in the tree view in the left pane.
  3. In the right pane, in the *Visual Studio projects location* text box, specify the *Microsoft Press\Visual CSharp Step By Step\Chapter 1* folder under your Documents folder.
  4. Click *OK*.
- If you are using Visual C# 2008 Express Edition, perform the following tasks to create a new console application.
  1. On the *File* menu, click *New Project*.
  2. In the *New Project* dialog box, click the *Console Application* icon.
  3. In the *Name* field, type **TextHello**.
  4. Click *OK*.

Visual Studio creates the project using the Console Application template and displays the starter code for the project, like this:



TextHello - Microsoft Visual Studio
File Edit View Project Build Debug Data Tools Test Window Help
Debug
Any CPU
Toolbox
Program.cs
Start Page
TextHello.Program
Main(string[] args)
Solution Explorer - TextHello
Solution 'TextHello' (1 project)
TextHello
Properties
References
Program.cs
Ready

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace TextHello
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
```

The *menu bar* at the top of the screen provides access to the features you'll use in the programming environment. You can use the keyboard or the mouse to access the menus and commands exactly as you can in all Windows-based programs. The *toolbar* is located beneath the menu bar and provides button shortcuts to run the most frequently used commands. The *Code and Text Editor* window occupying the main part of the IDE displays the contents of source files. In a multi-file project, when you edit more than one file, each source file has its own tab labeled with the name of the source file. You can click the tab to bring the named source file to the foreground in the *Code and Text Editor* window. The *Solution Explorer* displays the names of the files associated with the project, among other items. You can also double-click a file name in the *Solution Explorer* to bring that source file to the foreground in the *Code and Text Editor* window.

Before writing the code, examine the files listed in the *Solution Explorer*, which Visual Studio 2008 has created as part of your project:

- **Solution 'TextHello'** This is the top-level solution file, of which there is one per application. If you use Windows Explorer to look at your Documents\Microsoft Press\Visual CSharp Step by Step\Chapter 1\TextHello folder, you'll see that the actual name of this file is TextHello.sln. Each solution file contains references to one or more project files.

- **TextHello** This is the C# project file. Each project file references one or more files containing the source code and other items for the project. All the source code in a single project must be written in the same programming language. In Windows Explorer, this file is actually called TextHello.csproj, and it is stored in your \My Documents\Microsoft Press\Visual CSharp Step by Step\Chapter 1\TextHello\TextHello folder.
- **Properties** This is a folder in the TextHello project. If you expand it, you will see that it contains a file called AssemblyInfo.cs. AssemblyInfo.cs is a special file that you can use to add attributes to a program, such as the name of the author, the date the program was written, and so on. You can specify additional attributes to modify the way in which the program runs. Learning how to use these attributes is outside the scope of this book.
- **References** This is a folder that contains references to compiled code that your application can use. When code is compiled, it is converted into an assembly and given a unique name. Developers use assemblies to package useful bits of code they have written so they can distribute it to other developers who might want to use the code in their applications. Many of the features that you will be using when writing applications using this book make use of assemblies provided by Microsoft with Visual Studio 2008.
- **Program.cs** This is a C# source file and is the one currently displayed in the Code and Text Editor window when the project is first created. You will write your code for the console application in this file. It also contains some code that Visual Studio 2008 provides automatically, which you will examine shortly.

Writing Your First Program

The Program.cs file defines a class called *Program* that contains a method called *Main*. All methods must be defined inside a class. You will learn more about classes in Chapter 7, “Creating and Managing Classes and Objects.” The *Main* method is special—it designates the program’s entry point. It must be a static method. (You will look at methods in detail in Chapter 3, “Writing Methods and Applying Scope,” and I discuss static methods in Chapter 7.)

Important C# is a case-sensitive language. You must spell *Main* with a capital *M*.

In the following exercises, you’ll write the code to display the message Hello World in the console; you’ll build and run your Hello World console application; and you’ll learn how namespaces are used to partition code elements.

Write the code by using IntelliSense

1. In the *Code and Text Editor* window displaying the Program.cs file, place the cursor in the *Main* method immediately after the opening brace, {, and then press Enter to create a new line. On the new line, type the word **Console**, which is the name of a built-in class. As you type the letter *C* at the start of the word *Console*, an IntelliSense list appears. This list contains all of the C# keywords and data types that are valid in this context. You can either continue typing or scroll through the list and double-click the Console item with the mouse. Alternatively, after you have typed *Con*, the IntelliSense list will automatically home in on the *Console* item and you can press the Tab or Enter key to select it.

   Main should look like this:

   ```
   static void Main(string[] args)
   {
       Console
   }
   ```

> **Note** *Console* is a built-in class that contains the methods for displaying messages on the screen and getting input from the keyboard.

2. Type a period immediately after *Console*. Another IntelliSense list appears, displaying the methods, properties, and fields of the *Console* class.

3. Scroll down through the list, select *WriteLine*, and then press Enter. Alternatively, you can continue typing the characters *W*, *r*, *i*, *t*, *e*, *L* until *WriteLine* is selected, and then press Enter.

   The IntelliSense list closes, and the word *WriteLine* is added to the source file. *Main* should now look like this:

   ```
   static void Main(string[] args)
   {
       Console.WriteLine
   }
   ```

4. Type an opening parenthesis , (. Another IntelliSense tip appears.

   This tip displays the parameters that the *WriteLine* method can take. In fact, *WriteLine* is an *overloaded method*, meaning that the *Console* class contains more than one method named *WriteLine*—it actually provides 19 different versions of this method. Each version of the *WriteLine* method can be used to output different types of data. (Chapter 3 describes overloaded methods in more detail.) *Main* should now look like this:

   ```
   static void Main(string[] args)
   {
       Console.WriteLine(
   }
   ```

Tip You can click the up and down arrows in the tip to scroll through the different overloads of *WriteLine*.

5. Type a closing parenthesis,) followed by a semicolon, ;.

   Main should now look like this:

```
static void Main(string[] args)
{
    Console.WriteLine();
}
```

6. Move the cursor, and type the string **“Hello World”**, including the quotation marks, between the left and right parentheses following the *WriteLine* method.

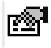
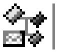
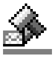
   Main should now look like this:

```
static void Main(string[] args)
{
    Console.WriteLine(“Hello World”);
}
```

Tip Get into the habit of typing matched character pairs, such as (and) and { and }, before filling in their contents. It’s easy to forget the closing character if you wait until after you’ve entered the contents.

IntelliSense Icons

When you type a period after the name of a class, IntelliSense displays the name of every member of that class. To the left of each member name is an icon that depicts the type of member. Common icons and their types include the following:

Icon	Meaning
	method (discussed in Chapter 3)
	property (discussed in Chapter 15)
	class (discussed in Chapter 7)
	struct (discussed in Chapter 9)
	enum (discussed in Chapter 9)

Icon	Meaning
	interface (discussed in Chapter 13)
	delegate (discussed in Chapter 17)
	extension method (discussed in Chapter 12)

You will also see other IntelliSense icons appear as you type code in different contexts.

Note You will frequently see lines of code containing two forward slashes followed by ordinary text. These are comments. They are ignored by the compiler but are very useful for developers because they help document what a program is actually doing. For example:

```
Console.ReadLine(); // Wait for the user to press the Enter key
```

The compiler will skip all text from the two slashes to the end of the line. You can also add multiline comments that start with a forward slash followed by an asterisk (/*). The compiler will skip everything until it finds an asterisk followed by a forward slash sequence (*/), which could be many lines lower down. You are actively encouraged to document your code with as many meaningful comments as necessary.

Build and run the console application

1. On the *Build* menu, click *Build Solution*.

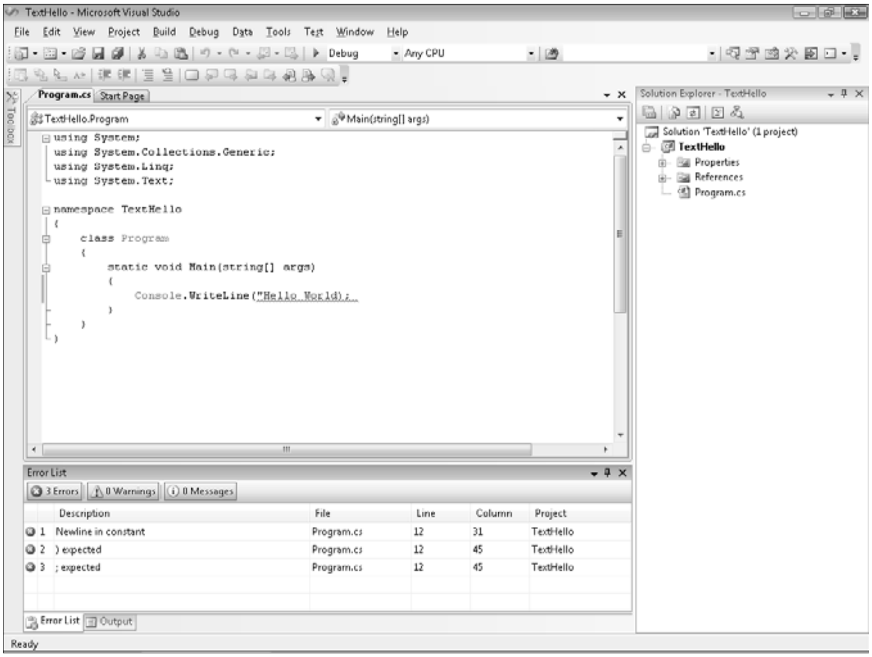
   This action compiles the C# code, resulting in a program that you can run. The *Output* window appears below the *Code and Text Editor* window.

Tip If the *Output* window does not appear, on the *View* menu, click *Output* to display it.

In the *Output* window, you should see messages similar to the following indicating how the program is being compiled.

```
------ Build started: Project: TextHello, Configuration: Debug Any CPU ----
C:\Windows\Microsoft.NET\Framework\v3.5\Csc.exe /config /nowarn:1701;1702 …
Compile complete -- 0 errors, 0 warnings
TextHello -> C:\Documents and Settings\John\My Documents\Microsoft Press\…
========== Build: 1 succeeded or up-to-date, 0 failed, 0 skipped ========
```

If you have made some mistakes, they will appear in the *Error List* window. The following image shows what happens if you forget to type the closing quotation marks after the text Hello World in the *WriteLine* statement. Notice that a single mistake can sometimes cause multiple compiler errors.



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace TextHello
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World);
        }
    }
}
```

	Description	File	Line	Column	Project
1	Newline in constant	Program.cs	12	31	TextHello
2	) expected	Program.cs	12	45	TextHello
3	; expected	Program.cs	12	45	TextHello

Tip You can double-click an item in the *Error List* window, and the cursor will be placed on the line that caused the error. You should also notice that Visual Studio displays a wavy red line under any lines of code that will not compile when you enter them.

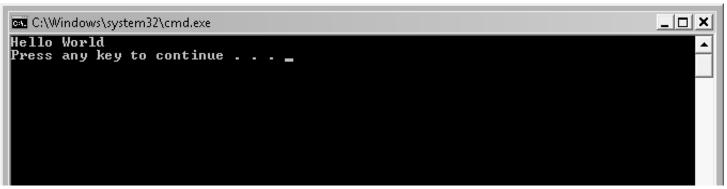
If you have followed the previous instructions carefully, there should be no errors or warnings, and the program should build successfully.

Tip There is no need to save the file explicitly before building because the *Build Solution* command automatically saves the file. If you are using Visual Studio 2008 Standard Edition or Visual Studio 2008 Professional Edition, the project is saved in the location specified when you created it. If you are using Visual C# 2008 Express Edition, the project is saved in a temporary location and is copied to the folder you specified in the *Options* dialog box only when you explicitly save the project by using the *Save All* command on the *File* menu or when you close Visual C# 2008 Express Edition.

An asterisk after the file name in the tab above the *Code and Text Editor* window indicates that the file has been changed since it was last saved.

2. On the *Debug* menu, click *Start Without Debugging*.

   A command window opens, and the program runs. The message Hello World appears, and then the program waits for you to press any key, as shown in the following graphic:

   ```
   Hello World
   Press any key to continue . . . _
   ```

> **Note** The prompt "Press any key to continue . . ." is generated by Visual Studio; you did not write any code to do this. If you run the program by using the *Start Debugging* command on the *Debug* menu, the application runs, but the command window closes immediately without waiting for you to press a key.

3. Ensure that the command window displaying the program's output has the focus, and then press Enter.

   The command window closes, and you return to the Visual Studio 2008 programming environment.

4. In *Solution Explorer*, click the TextHello project (not the solution), and then click the *Show All Files* toolbar button on the *Solution Explorer* toolbar—this is the second button from the left on the toolbar in the Solution Explorer window.

   Entries named *bin* and *obj* appear above the Program.cs file. These entries correspond directly to folders named *bin* and *obj* in the project folder (Microsoft Press\Visual CSharp Step by Step\Chapter 1\TextHello\TextHello). Visual Studio creates these folders when you build your application, and they contain the executable version of the program together with some other files used to build and debug the application.

5. In *Solution Explorer*, click the plus sign (**+**) to the left of the *bin* entry.

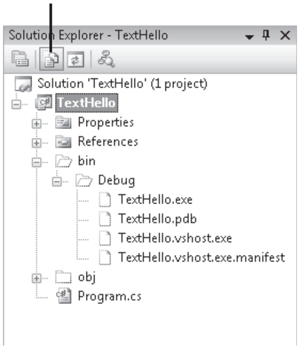
   Another folder named *Debug* appears.

> **Note** You may also see a folder called *Release*.

6. In *Solution Explorer*, click the plus sign (**+**) to the left of the *Debug* folder.

   Four more items named TextHello.exe, TextHello.pdb, TextHello.vshost.exe, and TextHello.vshost.exe.manifest appear, like this:

   Show All Files



> **Note** If you are using Visual C# 2008 Express Edition, you might not see all of these files.

The file TextHello.exe is the compiled program, and it is this file that runs when you click *Start Without Debugging* on the *Debug* menu. The other files contain information that is used by Visual Studio 2008 if you run your program in *Debug* mode (when you click *Start Debugging* on the *Debug* menu).

Using Namespaces

The example you have seen so far is a very small program. However, small programs can soon grow into much bigger programs. As a program grows, two issues arise. First, it is harder to understand and maintain big programs than it is to understand and maintain smaller programs. Second, more code usually means more names, more methods, and more classes. As the number of names increases, so does the likelihood of the project build failing because two or more names clash (especially when a program also uses third-party libraries written by developers who have also used a variety of names).

In the past, programmers tried to solve the name-clashing problem by prefixing names with some sort of qualifier (or set of qualifiers). This solution is not a good one because it's not scalable; names become longer, and you spend less time writing software and more time typing (there is a difference) and reading and rereading incomprehensibly long names.

Namespaces help solve this problem by creating a named container for other identifiers, such as classes. Two classes with the same name will not be confused with each other if they live in different namespaces. You can create a class named *Greeting* inside the namespace named *TextHello*, like this:

```
namespace TextHello
{
    class Greeting
    {
        ...
    }
}
```

You can then refer to the *Greeting* class as *TextHello.Greeting* in your programs. If another developer also creates a *Greeting* class in a different namespace, such as *NewNamespace*, and installs it on your computer, your programs will still work as expected because they are using the *TextHello.Greeting* class. If you want to refer to the other developer's *Greeting* class, you must specify it as *NewNamespace.Greeting*.

It is good practice to define all your classes in namespaces, and the Visual Studio 2008 environment follows this recommendation by using the name of your project as the top-level namespace. The .NET Framework software development kit (SDK) also adheres to this recommendation; every class in the .NET Framework lives inside a namespace. For example, the *Console* class lives inside the *System* namespace. This means that its full name is actually *System.Console*.

Of course, if you had to write the full name of a class every time you used it, the situation would be no better than prefixing qualifiers or even just naming the class with some globally unique name such *SystemConsole* and not bothering with a namespace. Fortunately, you can solve this problem with a *using* directive in your programs. If you return to the TextHello program in Visual Studio 2008 and look at the file Program.cs in the *Code and Text Editor* window, you will notice the following statements at the top of the file:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
```

A *using* statement brings a namespace into scope. In subsequent code in the same file, you no longer have to explicitly qualify objects with the namespace to which they belong. The four namespaces shown contain classes that are used so often that Visual Studio 2008 automatically adds these *using* statements every time you create a new project. You can add further *using* directives to the top of a source file.

The following exercise demonstrates the concept of namespaces in more depth.

Try longhand names

1. In the *Code and Text Editor* window displaying the Program.cs file, comment out the first *using* directive at the top of the file, like this:

   ```
   //using System;
   ```

2. On the *Build* menu, click *Build Solution*.

   The build fails, and the *Error List* window displays the following error message:

   ```
   The name 'Console' does not exist in the current context.
   ```

3. In the *Error List* window, double-click the error message.

   The identifier that caused the error is selected in the Program.cs source file.

4. In the *Code and Text Editor* window, edit the *Main* method to use the fully qualified name *System.Console*.

   Main should look like this:

   ```
   static void Main(string[] args)
   {
       System.Console.WriteLine("Hello World");
   }
   ```

Note When you type *System*. the names of all the items in the *System* namespace are displayed by IntelliSense.

5. On the *Build* menu, click *Build Solution*.

   The build should succeed this time. If it doesn't, make sure that *Main* is exactly as it appears in the preceding code, and then try building again.

6. Run the application to make sure it still works by clicking *Start Without Debugging* on the *Debug* menu.

Namespaces and Assemblies

A *using* statement simply brings the items in a namespace into scope and frees you from having to fully qualify the names of classes in your code. Classes are compiled into *assemblies*. An assembly is a file that usually has the *.dll* file name extension, although strictly speaking, executable programs with the *.exe* file name extension are also assemblies.

An assembly can contain many classes. The classes that the .NET Framework class library comprises, such as *System.Console,* are provided in assemblies that are installed on your computer together with Visual Studio. You will find that the .NET Framework class library contains many thousands of classes. If they were all held in the same assembly, the assembly would be huge and difficult to maintain. (If Microsoft updated a single method in a single class, it would have to distribute the entire class library to all developers!)

For this reason, the .NET Framework class library is split into a number of assemblies, partitioned by the functional area to which the classes they contain relate. For example, there is a "core" assembly that contains all the common classes, such as *System.Console*, and there are further assemblies that contain classes for manipulating databases, accessing Web services, building graphical user interfaces, and so on. If you want to make use of a class in an assembly, you must add to your project a reference to that assembly. You can then add *using* statements to your code that bring the items in namespaces in that assembly into scope.

You should note that there is not necessarily a 1:1 equivalence between an assembly and a namespace; a single assembly can contain classes for multiple namespaces, and a single namespace can span multiple assemblies. This all sounds very confusing at first, but you will soon get used to it.

When you use Visual Studio to create an application, the template you select automatically includes references to the appropriate assemblies. For example, in *Solution Explorer* for the TextHello project, click the plus sign (**+**) to the left of the *References* folder. You will see that a Console application automatically includes references to assemblies called *System*, *System.Core*, *System.Data*, and *System.Xml*. You can add references for additional assemblies to a project by right-clicking the *References* folder and clicking *Add Reference*—you will practice performing this task in later exercises.

Creating a Graphical Application

So far, you have used Visual Studio 2008 to create and run a basic Console application. The Visual Studio 2008 programming environment also contains everything you need to create graphical Windows-based applications. You can design the form-based user interface of a Windows-based application interactively. Visual Studio 2008 then generates the program statements to implement the user interface you've designed.

Visual Studio 2008 provides you with two views of a graphical application: the *design view* and the *code view*. You use the *Code and Text Editor* window to modify and maintain the code and logic for a graphical application, and you use the *Design View* window to lay out your user interface. You can switch between the two views whenever you want.

In the following set of exercises, you'll learn how to create a graphical application by using Visual Studio 2008. This program will display a simple form containing a text box where you can enter your name and a button that displays a personalized greeting in a message box when you click the button.

> **Note** Visual Studio 2008 provides two templates for building graphical applications—the Windows Forms Application template and the WPF Application template. Windows Forms is a technology that first appeared with the .NET Framework version 1.0. WPF, or Windows Presentation Foundation, is an enhanced technology that first appeared with the .NET Framework version 3.0. It provides many additional features and capabilities over Windows Forms, and you should consider using it in preference to Windows Forms for all new development.

Create a graphical application in Visual Studio 2008

- If you are using Visual Studio 2008 Standard Edition or Visual Studio 2008 Professional Edition, perform the following operations to create a new graphical application:

  1. On the *File* menu, point to *New*, and then click *Project*.

     The *New Project* dialog box opens.

  2. In the *Project Types* pane, click *Visual C#*.
  3. In the *Templates* pane, click the *WPF Application* icon.
  4. Ensure that the *Location* field refers to your *Documents\Microsoft Press\Visual CSharp Step by Step\Chapter 1* folder.
  5. In the *Name* field, type **WPFHello**.
  6. In the *Solution* field, ensure that *Create new solution* is selected.

     This action creates a new solution for holding the project. The alternative, *Add to Solution*, adds the project to the TextHello solution.

  7. Click *OK*.

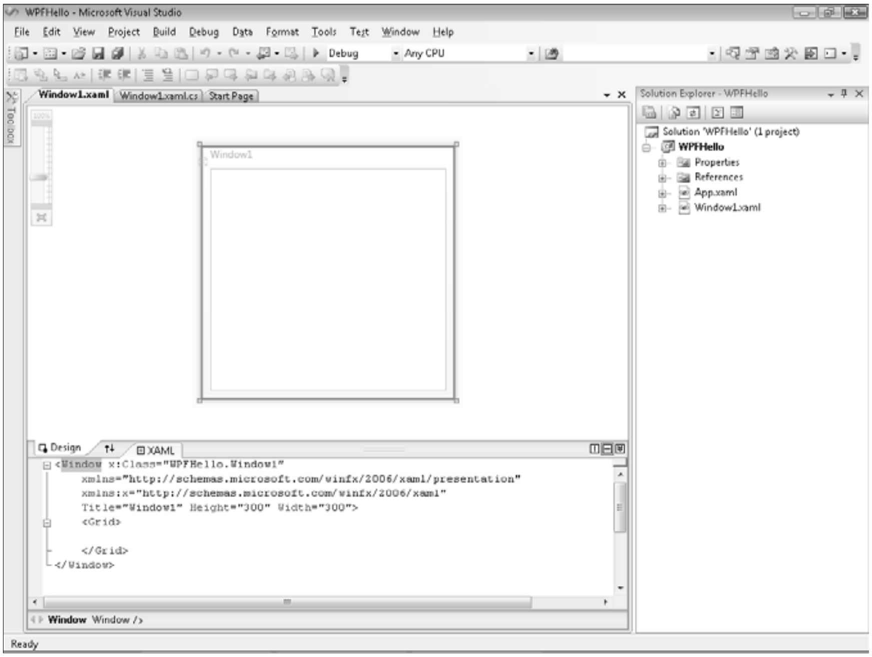
- If you are using Visual C# 2008 Express Edition, perform the following tasks to create a new graphical application.

  1. On the *File* menu, click *New Project*.
  2. If the *New Project* message box appears, click *Save* to save your changes to the TextHello project. In the *Save Project* dialog box, verify that the *Location* field is set to *Microsoft Press\Visual CSharp Step By Step\Chapter 1* under your Documents folder, and then click *Save*.

3. In the *New Project* dialog box, click the *WPF Application* icon.
4. In the *Name* field, type **WPFHello**.
5. Click *OK*.

Visual Studio 2008 closes your current application and creates the new WPF application. It displays an empty WPF form in the *Design View* window, together with another window containing an XAML description of the form, as shown in the following graphic:

Tip Close the *Output* and *Error List* windows to provide more space for displaying the *Design View* window.


WPFHello - Microsoft Visual Studio
File Edit View Project Build Debug Data Format Tools Test Window Help
Debug
Any CPU
Window1.xaml
Window1.xaml.cs
Start Page
Toolbox
Window1
Solution Explorer - WPFHello
Solution 'WPFHello' (1 project)
WPFHello
Properties
References
App.xaml
Window1.xaml
Design
XAML
Window Window />
Ready

```
<Window x:Class="WPFHello.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Window1" Height="300" Width="300">
    <Grid>

    </Grid>
</Window>
```

XAML stands for Extensible Application Markup Language and is an XML-like language used by WPF applications to define the layout of a form and its contents. If you have knowledge of XML, XAML should look familiar. You can actually define a WPF form completely by writing an XAML description if you don't like using the Design View window of Visual Studio or if you don't have access to Visual Studio; Microsoft provides an XAML editor called XMLPad that you can download free of charge from the MSDN Web site.

In the following exercise, you'll use the Design View window to add three controls to the Windows form and examine some of the C# code automatically generated by Visual Studio 2008 to implement these controls.

Create the user interface

1. Click the *Toolbox* tab that appears to the left of the form in the Design View window.

   The Toolbox appears, partially obscuring the form, and displaying the various components and controls that you can place on a Windows form. The Common section displays a list of controls that are used by most WPF applications. The Controls section displays a more extensive list of controls.

2. In the Common section, click Label, and then click the visible part of the form.

   A label control is added to the form (you will move it to its correct location in a moment), and the *Toolbox* disappears from view.

> **Tip** If you want the *Toolbox* to remain visible but not to hide any part of the form, click the *Auto Hide* button to the right in the *Toolbox* title bar (it looks like a pin). The *Toolbox* appears permanently on the left side of the Visual Studio 2008 window, and the *Design View* window shrinks to accommodate it. (You may lose a lot of space if you have a low-resolution screen.) Clicking the *Auto Hide* button once more causes the *Toolbox* to disappear again.

3. The label control on the form is probably not exactly where you want it. You can click and drag the controls you have added to a form to reposition them. Using this technique, move the label control so that it is positioned toward the upper-left corner of the form. (The exact placement is not critical for this application.)

> **Note** The XAML description of the form in the lower pane now includes the label control, together with properties such as its location on the form, governed by the *Margin* property. The *Margin* property consists of four numbers indicating the distance of each edge of the label from the edges of the form. If you move the control around the form, the value of the *Margin* property changes. If the form is resized, the controls anchored to the form's edges that move are resized to preserve their margin values. You can prevent this by setting the *Margin* values to zero. You learn more about the *Margin* and also the *Height* and *Width* properties of WPF controls in Chapter 22, "Introducing Windows Presentation Foundation."

4. On the *View* menu, click *Properties Window*.

   The *Properties* window appears on the lower-right side of the screen, under *Solution Explorer* (if it was not already displayed). The *Properties* window provides another way for you to modify the properties for items on a form, as well as other items in a project. It is context sensitive in that it displays the properties for the currently selected item. If you click the title bar of the form displayed in the *Design View* window, you can see that the *Properties* window displays the properties for the form itself. If you click the label control, the window displays the properties for the label instead. If you click anywhere else on the form, the *Properties* window displays the properties for a mysterious item called a *grid*. A grid acts as a container for items on a WPF form, and you can use the grid, among other things, to indicate how items on the form should be aligned and grouped together.

5. Click the label control on the form. In the *Properties* window, locate the *Text* section.

   By using the properties in this section, you can specify the font and font size for the label but not the actual text that the label displays.

6. Change the *FontSize* property to **20**, and then click the title bar of the form.

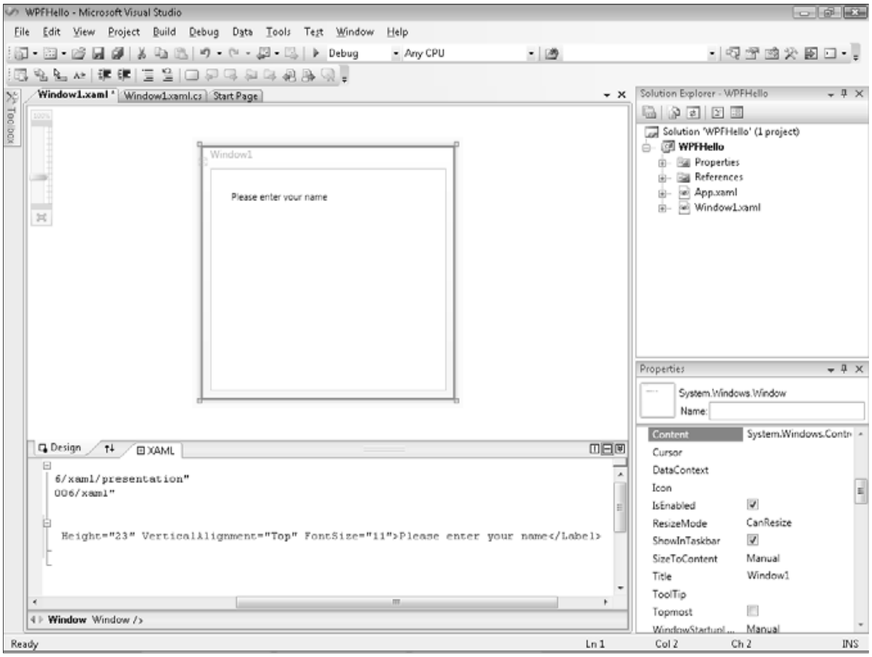
   The size of the text in the label changes, although the label is no longer big enough to display the text. Change the *FontSize* property back to **12**.

> **Note** The text displayed in the label might not resize itself immediately in the *Design View* window. It will correct itself when you build and run the application, or if you close and open the form in the *Design View* window.

7. Scroll the XAML description of the form in the lower pane to the right, and examine the properties of the label control.

   The label control consists of a `<Label>` tag containing property values, followed by the text for the label itself ("Label"), followed by a closing `</Label>` tag.

8. Change the text Label (just before the closing tag) to **Please enter your name**, as shown in the following image.



```
6/xaml/presentation"
006/xaml"

 Height="23" VerticalAlignment="Top" FontSize="11">Please enter your name</Label>
```

Properties

System.Windows.Window

Name:

Content	System.Windows.Contr
Cursor	
DataContext	
Icon	
IsEnabled	☑
ResizeMode	CanResize
ShowInTaskbar	☑
SizeToContent	Manual
Title	Window1
ToolTip	
Topmost	☐
WindowStartupL...	Manual

Notice that the text displayed in the label on the form changes, although the label is still too small to display it correctly.

9. Click the form in the *Design View* window, and then display the *Toolbox* again.

> **Note** If you don't click the form in the *Design View* window, the *Toolbox* displays the message "There are no usable controls in this group."

10. In the *Toolbox*, click *TextBox*, and then click the form. A text box control is added to the form. Move the text box control so that it is directly underneath the label control.

> **Tip** When you drag a control on a form, alignment indicators appear automatically when the control becomes aligned vertically or horizontally with other controls. This gives you a quick visual cue for making sure that controls are lined up neatly.

11. While the text box control is selected, in the *Properties* window, change the value of the *Name* property displayed at the top of the window to **userName**.

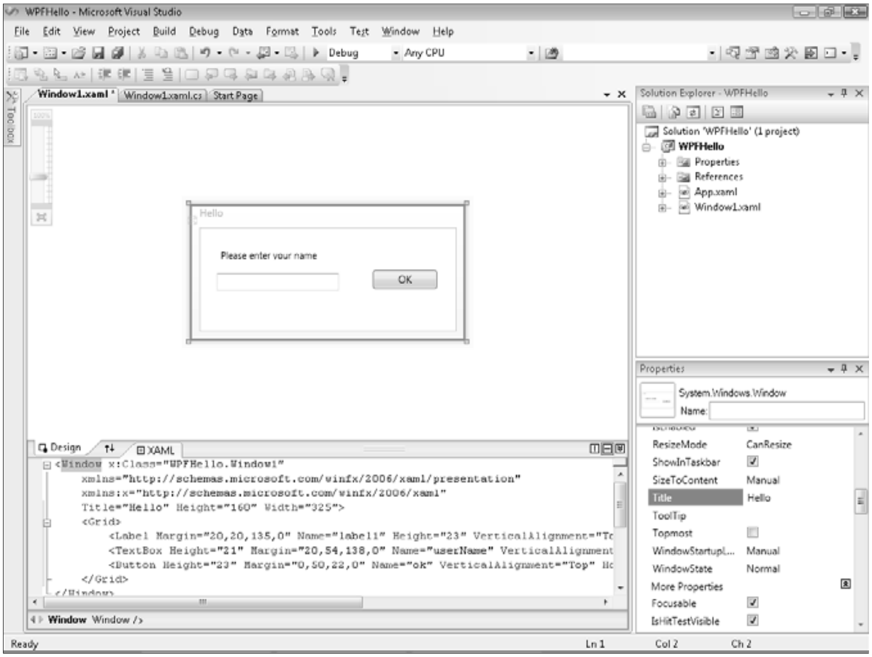
> **Note** You will learn more about naming conventions for controls and variables in Chapter 2, "Working with Variables, Operators, and Expressions."

12. Display the *Toolbox* again, click *Button*, and then click the form. Drag the button control to the right of the text box control on the form so that the bottom of the button is aligned horizontally with the bottom of the text box.
13. Using the *Properties* window, change the *Name* property of the button control to **ok**.
14. In the XAML description of the form, scroll the text to the right to display the caption displayed by the button, and change it from Button to **OK**. Verify that the caption of the button control on the form changes.
15. Click the title bar of the Window1.xaml form in the *Design View* window. In the *Properties* window, change the *Title* property to **Hello**.
16. In the *Design View* window, notice that a resize handle (a small square) appears on the lower right-hand corner of the form when it is selected. Move the mouse pointer over the resize handle. When the pointer changes to a diagonal double-headed arrow, click and drag the pointer to resize the form. Stop dragging and release the mouse button when the spacing around the controls is roughly equal.

> **Important** Click the title bar of the form and not the outline of the grid inside the form before resizing it. If you select the grid, you will modify the layout of the controls on the form but not the size of the form itself.

> **Note** If you make the form narrower, the *OK* button remains a fixed distance from the right-hand edge of the form, determined by its *Margin* property. If you make the form too narrow, the *OK* button will overwrite the text box control. The right-hand margin of the label is also fixed, and the text for the label will start to disappear when the label shrinks as the form becomes narrower.

The form should now look similar to this:



```
<Window x:Class="WPFHello.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Hello" Height="160" Width="325">
    <Grid>
        <Label Margin="20,20,135,0" Name="label1" Height="23" VerticalAlignment="To
        <TextBox Height="21" Margin="20,54,138,0" Name="userName" VerticalAlignment
        <Button Height="23" Margin="0,50,22,0" Name="ok" VerticalAlignment="Top" Ho
    </Grid>
</Window>
```

ResizeMode	CanResize
ShowInTaskbar	☑
SizeToContent	Manual
Title	Hello
ToolTip	
Topmost	☐
WindowStartupL...	Manual
WindowState	Normal
More Properties	
Focusable	☑
IsHitTestVisible	☑

17. On the *Build* menu, click *Build Solution*, and verify that the project builds successfully.
18. On the *Debug* menu, click *Start Without Debugging*.

    The application should run and display your form. You can type your name in the text box and click *OK*, but nothing happens yet. You need to add some code to process the *Click* event for the *OK* button, which is what you will do next.
19. Click the *Close* button (the *X* in the upper-right corner of the form) to close the form and return to Visual Studio.

You have managed to create a graphical application without writing a single line of C# code. It does not do much yet (you will have to write some code soon), but Visual Studio actually generates a lot of code for you that handles routine tasks that all graphical applications must perform, such as starting up and displaying a form. Before adding your own code to the application, it helps to have an understanding of what Visual Studio has generated for you.

In *Solution Explorer*, click the plus sign (**+**) beside the file Window1.xaml. The file Window1.xaml.cs appears. Double-click the file Window1.xaml.cs. The code for the form is displayed in the *Code and Text Editor* window. It looks like this:

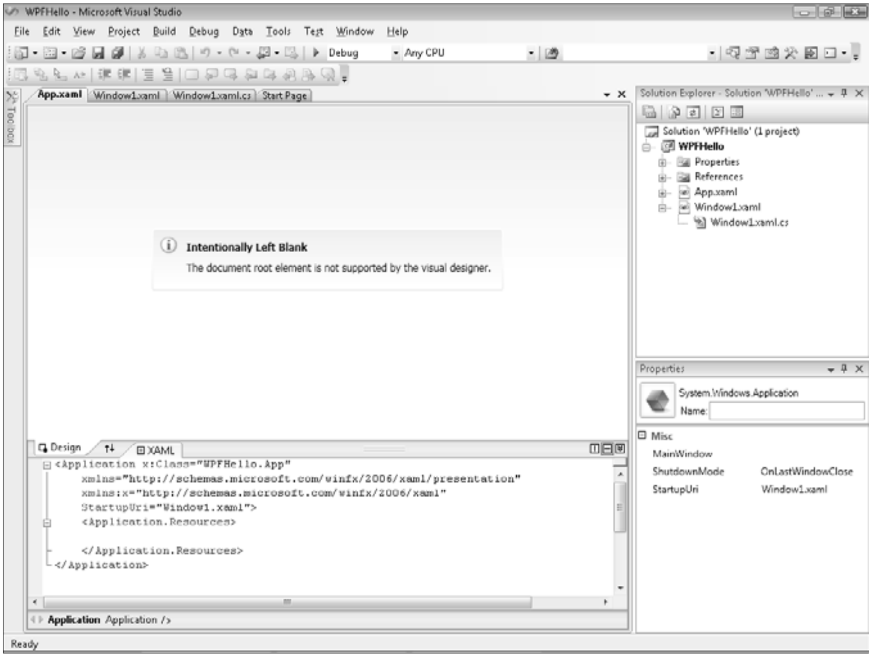
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace WPFHello
{
    /// <summary>
    /// Interaction logic for Window1.xaml
    /// </summary>

    public partial class Window1 : Window
    {

        public Window1()
        {
            InitializeComponent();
        }

    }
}
```

Apart from a good number of *using* statements bringing into scope some namespaces that most WPF applications use, the file contains the definition of a class called *Window1* but not much else. There is a little bit of code for the *Window1* class known as a constructor that calls a method called *InitializeComponent*, but that is all. (A *constructor* is a special method with the same name as the class. It is executed when an instance of the class is created and can contain code to initialize the instance. You will learn about constructors in Chapter 7.) In fact, the application contains a lot more code, but most of it is generated automatically based on the XAML description of the form, and it is hidden from you. This hidden code performs operations such as creating and displaying the form, and creating and positioning the various controls on the form.

The purpose of the code that you *can* see in this class is so that you can add your own methods to handle the logic for your application, such as what happens when the user clicks the *OK* button.

Tip You can also display the C# code file for a WPF form by right-clicking anywhere in the *Design View* window and then clicking *View Code*.

At this point you might well be wondering where the *Main* method is and how the form gets displayed when the application runs; remember that *Main* defines the point at which the program starts. In *Solution Explorer*, you should notice another source file called App.xaml. If you double-click this file, the *Design View* window displays the message “Intentionally Left Blank,” but the file has an XAML description. One property in the XAML code is called *StartupUri*, and it refers to the Window1.xaml file as shown here:


WPFHello - Microsoft Visual Studio
File Edit View Project Build Debug Data Tools Test Window Help
Debug
Any CPU
App.xaml
Window1.xaml
Window1.xaml.cs
Start Page
Toolbox
Solution Explorer - Solution 'WPFHello' ...
Solution 'WPFHello' (1 project)
WPFHello
Properties
References
App.xaml
Window1.xaml
Window1.xaml.cs
Properties
System.Windows.Application
Name:
Misc
Design
XAML
Application Application />
Ready

Intentionally Left Blank

The document root element is not supported by the visual designer.

```
<Application x:Class="WPFHello.App"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    StartupUri="Window1.xaml">
    <Application.Resources>

    </Application.Resources>
</Application>
```

MainWindow	
ShutdownMode	OnLastWindowClose
StartupUri	Window1.xaml

If you click the plus sign (+) adjacent to App.xaml in *Solution Explorer*, you will see that there is also an Application.xaml.cs file. If you double-click this file, you will find it contains the following code:

```
using System;
using System.Collections.Generic;
using System.Configuration;
using System.Data;
using System.Linq;
using System.Windows;
```

```
namespace WPFHello
{
    /// <summary>
    /// Interaction logic for App.xaml
    /// </summary>

    public partial class App : Application
    {

    }
}
```

Once again, there are a number of *using* statements, but not a lot else, not even a *Main* method. In fact, *Main* is there, but it is also hidden. The code for *Main* is generated based on the settings in the App.xaml file; in particular, *Main* will create and display the form specified by the *StartupUri* property. If you want to display a different form, you edit the App.xaml file.

The time has come to write some code for yourself!

Write the code for the OK button

1. Click the *Window1.xaml* tab above the *Code and Text Editor* window to display Window1 in the *Design View* window.
2. Double-click the *OK* button on the form.

   The Window1.xaml.cs file appears in the *Code and Text Editor* window, but a new method has been added called *ok_Click*. Visual Studio automatically generates code to call this method whenever the user clicks the *OK* button. This is an example of an event, and you will learn much more about how events work as you progress through this book.
3. Add the code shown in bold type to the *ok_Click* method:

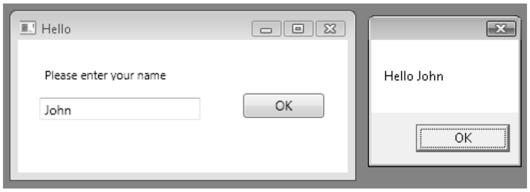
   ```
   void ok_Click(object sender, RoutedEventArgs e)
   {
       MessageBox.Show(“Hello “ + userName.Text);
   }
   ```

   This is the code that will run when the user clicks the *OK* button. Do not worry too much about the syntax of this code just yet (just make sure you copy it exactly as shown) because you will learn all about methods in Chapter 3. The interesting part is the *MessageBox.Show* statement. This statement displays a message box containing the text “Hello” with whatever name the user typed into the username text box on the appended form.
4. Click the *Window1.xaml* tab above the *Code and Text Editor* window to display Window1 in the *Design View* window again.

5. In the lower pane displaying the XAML description of the form, examine the *Button* element, but be careful not to change anything. Notice that it contains an element called *Click* that refers to the *ok_Click* method:

   ```
   <Button Height="23" … Click="ok_Click">OK</Button>
   ```

6. On the *Debug* menu, click *Start Without Debugging*.
7. When the form appears, type your name in the text box, and then click *OK*. A message box appears, welcoming you by name.

8. Click *OK* in the message box.

   The message box closes.

9. Close the form.
   - If you want to continue to the next chapter

     Keep Visual Studio 2008 running, and turn to Chapter 2.

   - If you want to exit Visual Studio 2008 now

     On the *File* menu, click *Exit*. If you see a *Save* dialog box, click *Yes* (if you are using Visual Studio 2008) or *Save* (if you are using Visual C# 2008 Express Edition) and save the project.

Chapter 1 Quick Reference

To	Do this	Key combination
Create a new console application using Visual Studio 2008 Standard or Professional Edition	On the *File* menu, point to *New*, and then click *Project* to open the *New Project* dialog box. For the project type, select *Visual C#*. For the template, select *Console Application*. Select a directory for the project files in the *Location* box. Choose a name for the project. Click *OK*.	
Create a new console application using Visual C# 2008 Express Edition	On the *Tools* menu, click *Options*. In the *Options* dialog box, click *Projects and Solutions*. In the *Visual Studio projects location* box, specify a directory for the project files. On the *File* menu, click *New Project* to open the *New Project* dialog box. For the template, select *Console Application*. Choose a name for the project. Click *OK*.	
Create a new graphical application using Visual Studio 2008 Standard or Professional Edition	On the *File* menu, point to *New*, and then click *Project* to open the *New Project* dialog box. For the project type, select Visual C#. For the template, select *WPF Application*. Select a directory for the project files in the *Location* box. Choose a name for the project. Click *OK*.	
Create a new graphical application using Visual C# 2008 Express Edition	On the *Tools* menu, click *Options*. In the *Options* dialog box, click *Projects and Solutions*. In the *Visual Studio projects location* box, specify a directory for the project files. On the *File* menu, click *New Project* to open the *New Project* dialog box. For the template, select *WPF Application*. Choose a name for the project. Click *OK*.	
Build the application	On the *Build* menu, click *Build Solution*.	F6
Run the application	On the *Debug* menu, click *Start Without Debugging*.	Ctrl+F5