

SECTION I

The Visual Studio IDE and Controls

The lessons in this section of the book explain how to use the Visual Studio integrated development environment (IDE). They explain how to use the IDE to create forms, place controls on the forms, and set control properties. These lessons describe some of Visual C#'s most useful controls and give you practice using them.

You can do practically all of this in the IDE without writing a single line of code! That makes Visual C# a great environment for rapid prototyping. You can build a form, add controls, and run the program to see what it looks like without ever creating a variable, declaring a function, or getting stuck in an infinite loop.

The lessons in this section explain how to get that far. A few of these lessons show how to add a line or two of code to a form to make it a bit more responsive, but for now the focus is on using the IDE to build forms and controls. Writing code (and fixing the inevitable bugs) comes later.

- ▶ **LESSON 1:** Getting Started with the Visual Studio IDE
- ▶ **LESSON 2:** Creating Controls
- ▶ **LESSON 3:** Making Controls Arrange Themselves
- ▶ **LESSON 4:** Handling Events
- ▶ **LESSON 5:** Making Menus
- ▶ **LESSON 6:** Making Tool Strips and Status Strips
- ▶ **LESSON 7:** Using RichTextBoxes
- ▶ **LESSON 8:** Using Standard Dialogs
- ▶ **LESSON 9:** Creating and Displaying New Forms
- ▶ **LESSON 10:** Building Custom Dialog

1

Getting Started with the Visual Studio IDE

The Visual Studio integrated development environment (IDE) plays a central role in Visual C# development. In this lesson you explore the IDE. You learn how to configure it for Visual C# development, and you learn about some of the more useful of the IDE's windows and what they do. When you finish this lesson, you'll know how to create a new project. It may not do much, but it will run and will prepare you for the lessons that follow.



Visual Studio is a development environment that you can use with several programming languages including Visual C#, Visual Basic, Visual C++, and F#. C# is a high-level programming language that can read inputs, calculate results, display outputs to the user, and perform other operations typical of high-level programming languages.

Visual C# is the combination of C# used in the Visual Studio development environment. You can use a text editor to write C# programs without Visual Studio, but it's a lot of work and is not the focus of this book.

Visual C# and C# go together like politicians and bickering: if you mention one, most people assume you're also talking about the other. Most people simply say C#, so this book does, too, unless there's a reason to distinguish between C# and Visual C#.

The .NET Framework also plays an important role in C# programs. It includes classes that make performing certain tasks easier, runtime tools that make it possible to execute C# programs, and other plumbing necessary to build and run C# programs.

Normally you don't need to worry about whether a feature is provided by Visual Studio, the C# language, or the .NET Framework. They all go together in this book, so for the purposes of this book at least you can ignore the difference.

INSTALLING C#

Before you can use C# to write the next blockbuster first-person Xbox game, you need to install it. So if you haven't done so already, install C#.

You can install the Express Edition at www.microsoft.com/express/Windows. If you think you need some other version (for example, you're working on a big project and you need test management, source code control, and other team programming tools), go to msdn.microsoft.com/vcsharp and install the version that's right for you.

It's a big installation, so it could take a while.

TALKIN' 'BOUT MY GENERATION

Developers talk about different generations of programming languages ranging from the very primitive to quite advanced. In a nutshell, the different generations of languages are:

- **1GL** — Machine language. This is a series of 0s and 1s that the machine can understand directly.
- **2GL** — Assembly language. This is a translation of machine language into terse mnemonics that can be easily translated into machine language. It provides no structure.
- **3GL** — A higher-level language such as FORTRAN or BASIC. These provide additional structure (such as looping and subroutines) that makes building complex programs easier.
- **4GL** — An even higher-level language or development environment that helps build programs, typically in a specific problem domain.
- **5GL** — A language where you specify goals and constraints and the language figures out how to satisfy them. For example, the database Structured Query Language (SQL) allows you to use statements like `SELECT FirstName FROM Employees`. You don't need to tell the database how to get the names; it figures that out for you.

Visual Studio provides code snippets that let you copy standard chunks of code into your program, IntelliSense that helps you select and use functions and other pieces of code, refactoring tools that help you rearrange and restructure your code, and more. That makes Visual C# a 4GL (or perhaps a 3.5GL depending on how high your standards are).

CONFIGURING THE IDE

When you first run Visual Studio, it asks how you want to configure the IDE. You can pick settings for general development, Visual Basic, Visual C#, and so forth. Because you're going to be focusing on C# development, select that option.



These settings determine such things as what keystrokes activate certain development features. You can certainly write C# programs with the Visual C++ settings but we may as well be on the same page, so when I say, "Press F5," the IDE starts your program instead of displaying a code window or whatever Visual C++ thinks F5 means.

If you ever want to switch to different settings (for example, if you got carried away during installation and selected the general settings and now want the C# settings), you can always change them later.

To change the settings later, open the Tools menu and select Import and Export Settings to display the Import and Export Settings Wizard. You can use this tool to save your current settings, reload previously saved settings, or reset settings to default values.

To reset settings, select the Reset All Settings option on the wizard's first page and click Next.

On the next page, indicate whether you want to save your current settings. When you've made your choice, click Next to display the page shown in Figure 1-1.

Select the Visual C# Development Settings choice and click Finish. (Then sit back and wait. Or better still, go get a coffee because this could take a while. Visual Studio has a lot of settings to reset, and it could take several minutes depending on how fast and busy your computer is.)

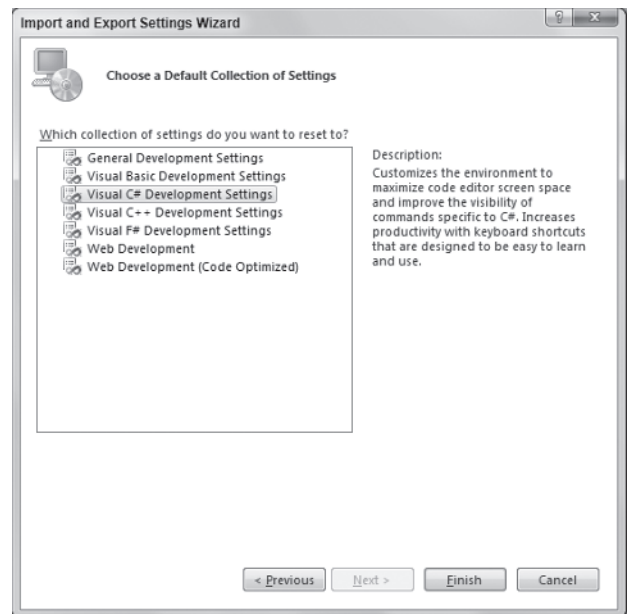


FIGURE 1-1

BUILDING YOUR FIRST PROGRAM

Now that you've installed C#, you're ready to get started. Launch Visual Studio by double-clicking its desktop icon or by selecting it from the system's Start menu.

To create a new project, press [Ctrl]+[Shift]+N to display the New Project dialog box shown in Figure 1-2. Alternatively, you can open the File menu, expand the New submenu, and select Project.

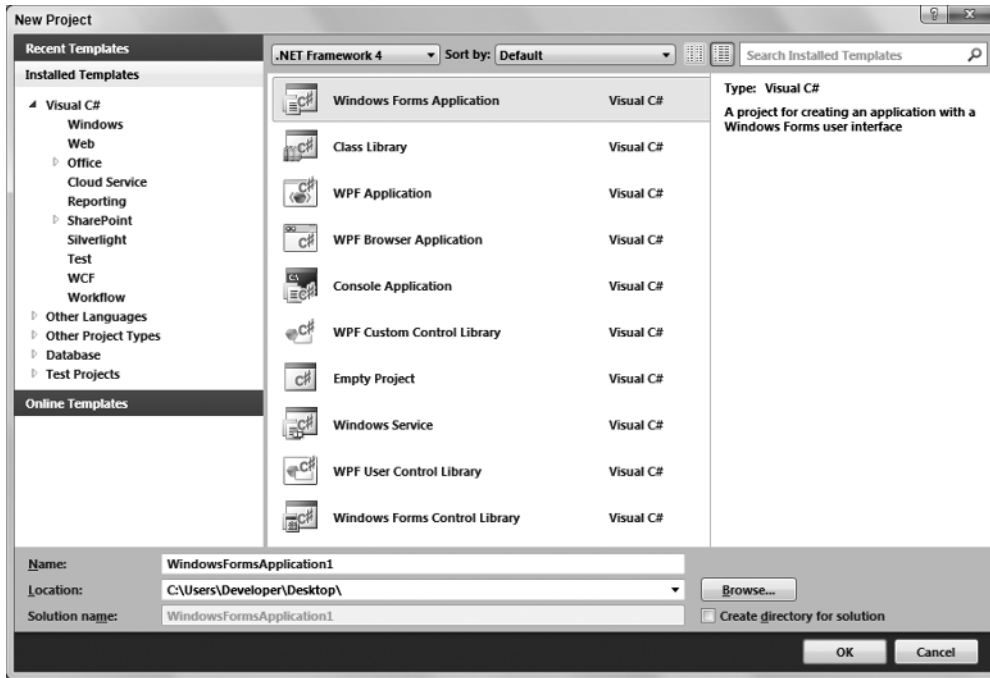


FIGURE 1-2

Expand the Visual C# project type folder on the left and select the template for the type of project that you want to build on the right. For most of this book, that will be a Visual C# Windows Forms Application.

Below the list of project types, you need to enter several pieces of information.

- **Name** — This is the application's name. Visual Studio creates a folder with this name to hold the program's files. It also sets some key values in the project to this name.
- **Location** — This is where you want Visual Studio to put the project's folder.
- **Solution Name** — If the Create Directory for Solution box is checked (which it is by default), Visual Studio creates a folder with this name at the location you entered. It then places the application's folder inside the solution's folder.

So if the Create Directory for Solution box is checked, you get a filesystem layout that looks like this:

```
SolutionFolder
  SolutionFiles
```

```
ApplicationFolder
  ApplicationFiles
```

If the Create Directory for Solution box is not checked, you get a filesystem layout that looks like this:

```
ApplicationFolder
  ApplicationFiles
```



An application contains a single program. A solution can contain several applications. A solution is useful when you want to build applications that go closely together. It's particularly useful if you want to build a library of routines plus an executable program to test the library.

The applications you build in this book are single programs so they don't really need to be inside a separate solution folder. Most of the time, I uncheck the Create Directory for Solution box to keep my filesystem simpler.



By default, Visual Studio places new projects in your Projects folder at some obscure location such as C:\Users\MyUserName\Documents\Visual Studio 2010\Projects. Later it can be hard finding these projects in Windows Explorer (for example, to make a copy).

To make finding projects easier, set the location to something more intuitive such as the desktop or a folder on the desktop. The next time you create a new project, Visual Studio will initialize the location textbox to this same location, so from now on it'll be easy to find your projects.

If you open the New Project dialog box while you have another project open, you'll see an additional dropdown that lists the choices Create New Solution and Add to Solution. The first choice closes the current solution and creates a new one. The second choice adds the new application to the solution you currently have open. Normally you'll want to create a new solution.

After you display the New Project dialog box and enter a Name, Location, and Solution Name, click OK. The result should look like Figure 1-3.



If you have previously edited a project, you can quickly reload it from the File menu's Recent Projects submenu. You can also load a solution into the IDE by double-clicking the solution's .sln file.

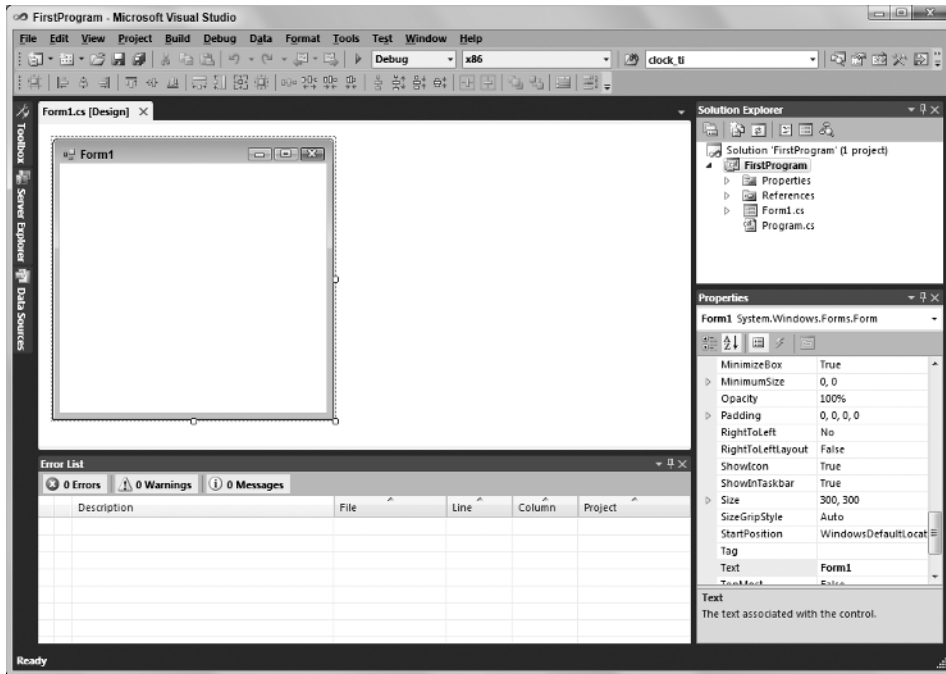


FIGURE 1-3

The rest of this lesson deals with the features available in Visual Studio, some of which are displayed in Figure 1-3. Before you launch into an inventory of useful features, however, press F5 or open the Debug menu and select Start Debugging to run your new program. Figure 1-4 shows the result. Admittedly this first program isn't very fancy, but by the same token you didn't need to do much to build it.

This first program may not seem terribly impressive but there's a lot going on behind the scenes. C# has built a form with a bunch of useful features, including:

- A resizable border and draggable title bar.
- Minimize, maximize, and close buttons in the upper-right corner.
- A system menu in the upper-left corner that contains the commands Restore, Move, Size, Minimize, Maximize, and Close.
- An icon in the system taskbar.
- The ability to use [Alt]+[Tab] and Flip3D ([Win]+[Tab]) to move between the application and others.
- Other standard window behaviors. For example, if you double-click the form's title bar it maximizes (or restores if it is already maximized), and if you press [Alt]+F4, the form closes.

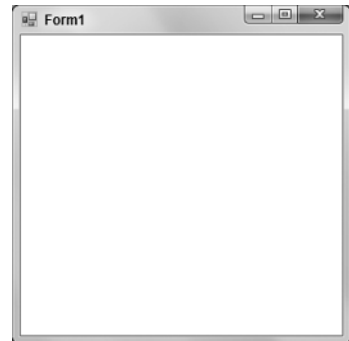


FIGURE 1-4

Unless you're an absolute beginner to Windows, you probably take all of these features for granted, but providing them is actually a lot of work. Not too long ago you would have had to write around 100 lines of code to handle these sorts of issues. Now Visual Studio automatically builds a form that handles most of these details for you.

You can still get in and change the way things work if you want to (for example, you can set a form's minimum and maximum sizes) but usually you can ignore all of these issues and concentrate on your particular application, not the Windows decorations.

A SUITABLE EXECUTABLE

Whenever you run a program in the IDE, Visual Studio builds an executable program, normally in the project's `bin\Debug` subdirectory. You can run the executable by simply double-clicking it, for example, in Windows Explorer.

That doesn't mean the executable is suitable to run on any old computer. If you copy that file to another computer, it won't run unless the .NET Framework runtime libraries have been installed there. If that computer has Visual Studio installed, you're all set, but if it doesn't you'll need to install the redistributable yourself.

To install these libraries, go to Microsoft's download web page www.microsoft.com/downloads and search for ".NET Framework redistributable." Pick the version that matches the one you're using (version 4.0 if you're using Visual C# 2010) and install it on the target computer.

Now you can copy C# executables onto the system and run them.

COPYING PROJECTS

Sometimes you may want to copy a project. For example, you might want to save the current version and then make a new one to try things out. Or you may want to give a copy of the project to a friend or your programming instructor.

To make a copy, you might look in the File menu and see the Copy As commands. Don't be tempted! Those commands copy single files, not the entire project. Later when you try to open one of those files, you'll discover that Visual Studio cannot find all of the other project pieces that it needs and you'll be left with nothing usable.

To correctly copy a project, find the solution or application folder in Windows Explorer and copy the project's *entire* directory hierarchy. Alternatively, you can compress the project directory into a compressed or zipped file and then copy that. Just be sure that whatever copying method you use brings along *all* of the project's files.

Note that you can delete the `bin` and `obj` subdirectories if you like to save space. Visual Studio will re-create them when it needs them later.



Compressing a project into an archive is very useful because it keeps all of its files together in a package. In particular, if you ever need to e-mail a project to someone (for example, if you e-mail me at RodStephens@CSharpHelper.com for help), you can remove the `bin` and `obj` directories, compress the project folder, and e-mail the package as a single file. (If you're sending the project to your instructor as part of an assignment, rename the compressed file so it contains your name and the name of the assignment, for example, `RodStephens6-1.zip`.)

EXPLORING THE IDE

The Visual Studio IDE contains a huge number of menus, toolbars, windows, wizards, editors, and other components to help you build applications. Some of these, such as the Solution Explorer and the Properties window, you will use every time you work on a program. Others, such as the Breakpoints window and the Connect to Device dialog box, are so specialized that it may be years before you need them.

Figure 1-5 shows the IDE with a simple project loaded with some of the IDE's most important pieces marked. The following list describes those pieces.

- 1. Menus** — The menus provide all sorts of useful commands. Exactly which commands are available, which are enabled, and even which menus are visible depends on what kind of editor is open in the editing area (#4). Some particularly useful menus include File (opening old projects and creating new ones), View (finding windows), Project (adding new forms and other items to a project), Debug (build, run, and debug the project), and Format (arrange controls on a form).
- 2. Toolbars** — The toolbars provide shortcuts for executing commands similar to those in the menus. Use the Tools menu's Customize command to determine which toolbars are visible.
- 3. Solution Explorer** — The Solution Explorer lists the files in the project. One of the most important is `Form1.cs`, which defines the controls and code for the form named `Form1`. If you double-click a file in the Solution Explorer, the IDE opens it in the editing area.
- 4. Editing Area** — The editing area displays files in appropriate editors. Most often you will use this area to design a form (place controls on it and set their properties) and write code for the form, but you can also use this area to edit other files such as text files, bitmaps, and icons.
- 5. Toolbox** — The Toolbox contains controls and components that you can place on a form. Select a tool and then click and drag to put a copy of the tool on the form. Notice that the Toolbox groups controls in tabs (All Windows Forms, Common Controls, Containers, Menus & Toolbars, and so on) to make finding the controls you need easier.

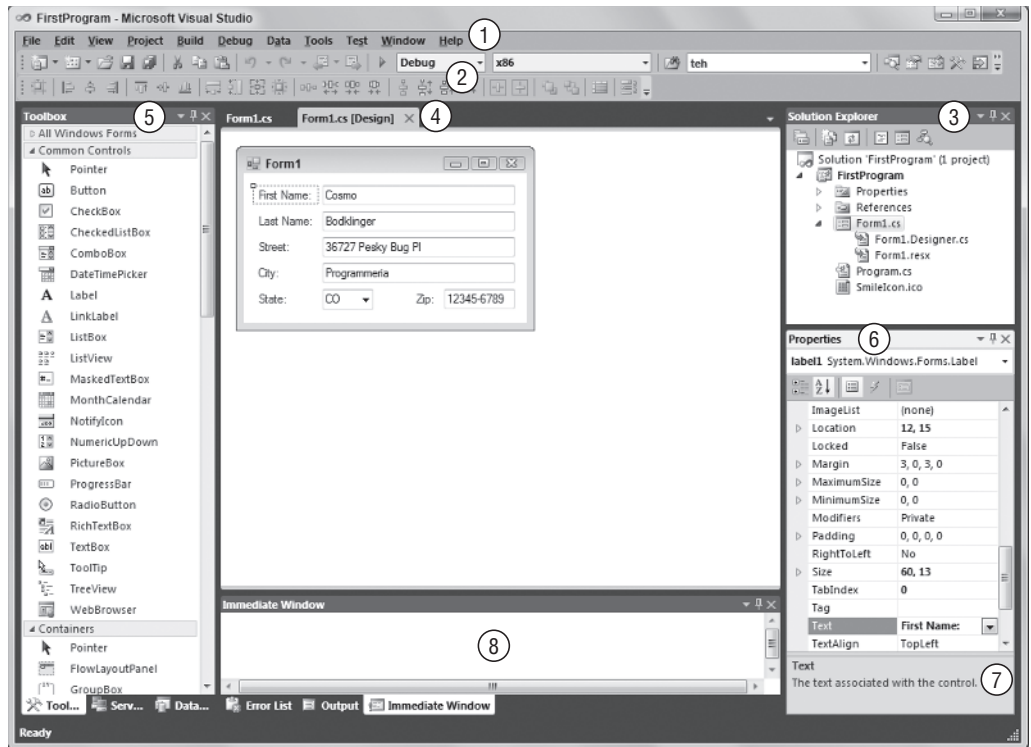


FIGURE 1-5

6. **Properties Window** — The Properties window lets you set control properties. Click a control on the Form Designer (shown in the editing area in Figure 1-5) to select it, or click and drag to select multiple controls. Then use the Properties window to set the control(s) properties. Notice that the top of the Properties window shows the name (`label1`) and type (`System.Windows.Forms.Label`) of the currently selected control. The currently selected property in Figure 1-5 is `Text`, and it has the value `First Name:.`
7. **Property Description** — The property description gives you a reminder about the current property's purpose. In Figure 1-5, it says that the `Text` property gives the text associated with the control. (Duh!)
8. **Other Windows** — This area typically contains other useful windows. The tabs at the bottom let you quickly switch between different windows.

Figure 1-5 shows a fairly typical arrangement of windows but Visual Studio is extremely flexible so you can rearrange the windows if you like. You can hide or show windows; make windows floating or docked to various parts of the IDE; make windows part of a tab group; and make windows automatically hide themselves if you don't need them constantly.

If you look closely at the right side of the title bar above one of the windows in Figure 1-5, for example, the Properties window, you'll see three icons: a dropdown arrow (▼), a thumbtack (📌), and an X.

If you click the dropdown arrow (or right-click the window's title bar), a menu appears with the following choices:

- **Float** — The window breaks free of wherever it's docked and floats above the IDE. You can drag it around and it will not re-dock. To make it dockable again, open the menu again and select Dock.
- **Dock** — The window can dock to various parts of the IDE. I'll say more about this shortly.
- **Dock as Tabbed Document** — The window becomes a tab in a tabbed area similar to #8 in Figure 1-5. Unfortunately, it's not always obvious which area will end up holding the window. To make the window a tab in a specific tabbed area, make it dockable and drag it onto a tab (described shortly).
- **Auto Hide** — The window shrinks itself to a small label stuck to one of the IDE's edges and its thumbtack icon turns sideways (→) to indicate that the window is auto-hiding. If you float the mouse over the label, the window reappears. As long as the mouse remains over the expanded window, it stays put, but if you move the mouse off the window, it auto-hides itself again. Select Auto Hide again or click the sideways thumbtack to turn off auto-hiding. Auto-hiding gets windows out of the way so you can work in a bigger editing area.
- **Hide** — The window disappears completely. To get the window back, you'll need to find it in the menus. You can find many of the most useful windows in the View menu, the View menu's Other Windows submenu, and the Debug menu's Windows submenu.

The thumbtack in a window's title bar works just like the dropdown menu's Auto Hide command does. Click the thumbtack to turn on auto-hiding. Expand the window and click the sideways thumbtack to turn off auto-hiding. (Turning auto-hiding off is sometimes called *pinning* the window.)

The ✕ symbol in the window's title bar hides the window just like the dropdown menu's Hide command does.

In addition to using a window's title bar menu and icons, you can drag windows into new positions. As long as a window is dockable or part of a tabbed window, you can grab its title bar and drag it to a new position.

As you drag the window, the IDE displays little drop targets to let you dock the window in various positions. If you move the window so the mouse is over a drop target, the IDE displays a translucent blue area to show where the window will land if you drop it. If you drop when the mouse is not over a drop target, the window becomes floating.

Figure 1-6 shows the Properties window being dragged in the IDE. The mouse is over the right drop target above the editing area so, as the translucent blue area shows, dropping it there would dock the window to the right side of the editing area.

The drop area just to the left of the mouse represents a tabbed area. If you drop on this kind of target, the window becomes a tab in that area.

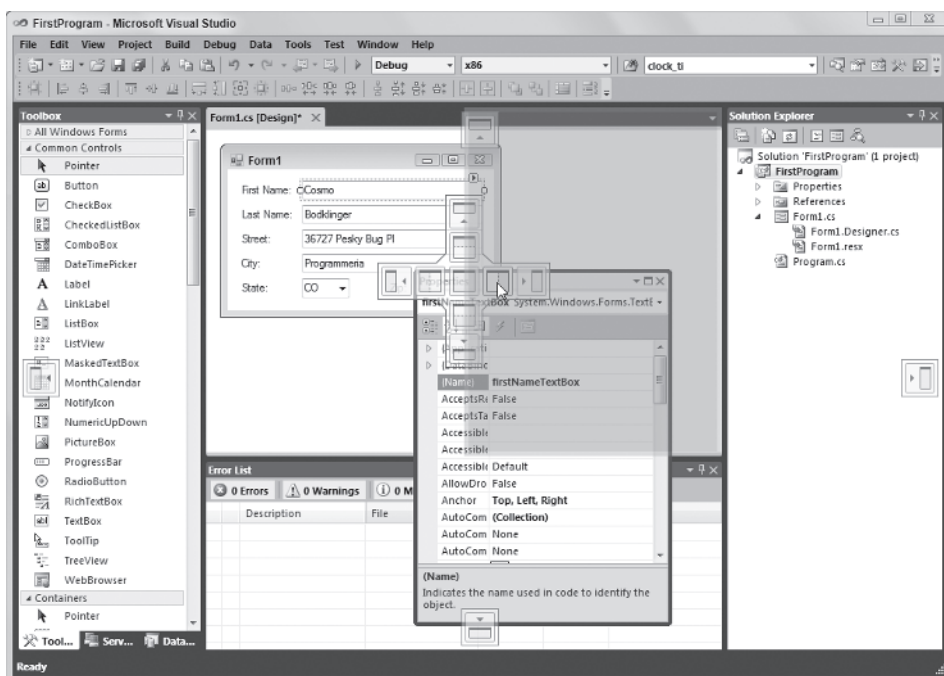


FIGURE 1-6

CUSTOMIZATION MODERATION

Visual Studio lets you move, dock, float, hide, auto-hide, and tabify windows. If you have multiple monitors, you can float a window and move them to other monitors, giving you a larger editing area. It's so flexible that it can present as many different faces as a politician during an election year.

Feel free to customize the IDE to suit your needs, but if you do, keep in mind that your version of Visual Studio may look nothing like the pictures in this book. To minimize confusion, you may want to keep the IDE looking more or less like Figure 1-5, at least until you get a better sense of which tools will be most useful to you.

TRY IT

In this Try It, you prepare for later work throughout the book. You locate web resources that you can use when you have questions or run into trouble. You create and run a program, explore the project's folder hierarchy, and make a copy of the project. You also get a chance to experiment a bit with the IDE, displaying new toolbars, moving windows around, and generally taking the IDE for a test drive and kicking the tires.



Note that the solutions for this lesson's Try It and exercises are not all available on the book's web site. The Try It and some of the exercises ask you to experiment with the IDE rather than producing a finished program, so there's really nothing to download. In later lessons, example solutions to the Try It and exercises are available on the book's web sites.

Lesson Requirements

In this lesson, you:

- Find and bookmark useful web resources.
- Launch Visual Studio and start a new Visual C# project.
- Experiment with the IDE's layout by displaying the Debug toolbar, pinning the Toolbox, and displaying the Output window.
- Run the program.
- Find the program's executable, copy it to the desktop, and run it there.
- Copy the project folder to a new location and make changes to the copy.
- Compress the project folder to make a backup.

Hints

- When you create a new project, be sure to specify a good location so you can find it later.
- Before you compress the project, remove the `bin` and `obj` directories to save space.

Step-by-Step

- Find and bookmark useful web resources.
 1. Open your favorite web browser.
 2. Create a new bookmark folder named C#. (See the browser's documentation if you don't know how to make a bookmark folder.)
 3. Go to the following web sites and bookmark the ones you like (feel free to search for others, too):
 - My C# Helper web site (CSharpHelper.com).
 - This book's web page (CSharpHelper.com/24hour.html).
 - This book's Wrox web page (go to www.wrox.com and search for *Stephens' C# Programming with Visual Studio 2010 24-Hour Trainer*).
 - Visual C# Express Edition MSDN forum (social.msdn.microsoft.com/Forums/en-US/Vsexpressvcs/threads)

- Visual C# IDE MSDN forum (social.msdn.microsoft.com/Forums/en-US/csharpide/threads)
- Visual C# Language MSDN forum (social.msdn.microsoft.com/Forums/en-US/csharplanguage/threads)
- Visual C# General MSDN forum (social.msdn.microsoft.com/Forums/en-US/csharpgeneral/threads)
- Launch Visual Studio and start a new Visual C# project.
 1. If you don't have a desktop icon for Visual Studio, create one.
 - a. Open the Windows Start menu and find Visual Studio. Either browse for it (it's probably in a folder named Visual Studio 2010 and the program is called Visual Studio 2010) or use the menu's search textbox to find it.
 - b. Right-click the program, open the Send To submenu, and select Desktop (Create Shortcut).
 2. Launch Visual Studio. Double-click the desktop icon or open the system's Start menu and select the Visual Studio 2010 program.
 3. Create a new project.
 - a. Press [Ctrl]+[Shift]+N or open the IDE's File menu, expand the New submenu, and select Project.
 - b. Expand the Visual C# project types folder and select the Windows Forms Application template.
 - c. Enter a project name and a good, easy-to-find location like the desktop or a folder named C# Projects on the desktop. Uncheck the Create Directory for Solution box and click OK.
- Experiment with the IDE's layout by displaying the Debug toolbar, pinning the Toolbox, and displaying the Output window.
 1. Open the Tools menu and select Customize. On the Customize dialog box, select the Toolbars tab and check the box next to the Debug toolbar. Experiment with the other toolbars if you like. Close the dialog box when you're done.
 2. If the Toolbox is auto-hiding (it should be after you first install Visual Studio), float the mouse over it until it expands. Click the thumbtack to pin it.
 3. To display the Output window, open the View menu and select Output. Grab the Output window's title bar and drag it around. Move it over some drop targets to see where it lands. When you're finished, drop it at the bottom of the IDE as shown in Figure 1-5.
- Run the program.
 1. Press F5 or open the Debug menu and select Start Debugging.

2. Try out the form's minimize, maximize, and close buttons, and the commands in the form's system menu. Move the form around and resize it. Marvel at the fact that you didn't need to write any code!
- Find the program's executable, copy it to the desktop, and run it there.
 1. Start Windows Explorer and navigate to the location that you specified when you created the new program.
 2. There you should find a folder named after the program. Open that folder and examine the files inside. Notice the `.sln` file that you can double-click to reopen the solution in Visual Studio. Notice also the `bin` and `obj` directories.
 3. Enter the `bin` directory and move into its `Debug` subdirectory. It contains several files including the executable, named after the program but with the `.exe` extension. Right-click the executable and select Copy.
 4. Right-click the desktop and select Paste to copy the executable to the desktop.
 5. Double-click the copy of the executable on the desktop.
 - Copy the project folder to a new location and make changes to the copy.
 1. In Windows Explorer, return to the location where you created the project and you can see the project's folder.
 2. Right-click the project's folder and select Copy.
 3. Right-click the desktop and select Paste to copy the project folder.
 4. Open the copied project folder and double-click the `.sln` file to open the copied project in Visual Studio. If the form doesn't open in the Form Designer (#4 in Figure 1-5), look in Solution Explorer and double-click the file `Form1.cs`.
 5. In the Form Designer, grab the handle on the form's lower-left corner and resize the form to make it tall and skinny.
 6. Run the modified program. Then go back to the original project (which should still be running in another instance of Visual Studio) and run it. Notice that the two versions display forms of different sizes.
 - Compress the project folder to make a backup.
 1. In Windows Explorer, return to the project's folder. Find and delete the `bin` and `obj` directories.
 2. Move up one level so you are in the location you specified when you created the project and you can see the project's folder. Right-click the folder, expand the Send To submenu, and select Compressed (Zipped) Folder.
 3. E-mail copies of your first project to all of your friends and relatives!



Please select Lesson 1 on the DVD to view the video that accompanies this lesson.

EXERCISES

1. Build a solution that contains two projects. (Create a project named Project1. Check the Create Directory for Solution box and name the solution TwoProjects. Then open the File menu, expand the Add submenu, and select New Project to add a new project named Project2.)
2. This chapter explains only a tiny fraction of the ways you can customize Visual Studio. Try another one by making your own toolbar. Select the Tools menu's Customize command. On the Toolbars tab, click the New button and name the new toolbar MyTools. Then on the Commands tab, drag commands onto the toolbar. Search the command categories for useful tools. The Debug and Format categories contain some useful commands.
3. This chapter also describes only a few of the windows Visual Studio offers. Use the menus to find and display the Output, Immediate, Error List, and Task List windows. Put them all in tabs at the bottom of Visual Studio (#8 in Figure 1-5).
4. Some tools are only available when Visual Studio is in a certain state. Look in the Debug menu's Windows submenu. Then start the program and look there again. Most of those windows are useful only when the program is running and you are debugging it. (I talk about some of them in later lessons.)
5. Later lessons spend a lot of time describing the form and code editors, but Visual Studio includes a lot of other editors, too. To try out the icon editor, open the Project menu and select Add New Item. Open the General category, select Icon File, give the file a good name, and click Add. Use the icon editor to make an icon similar to the one shown in Figure 1-7. Later you can double-click the icon file in the Solution Explorer to reopen the icon in the editor.

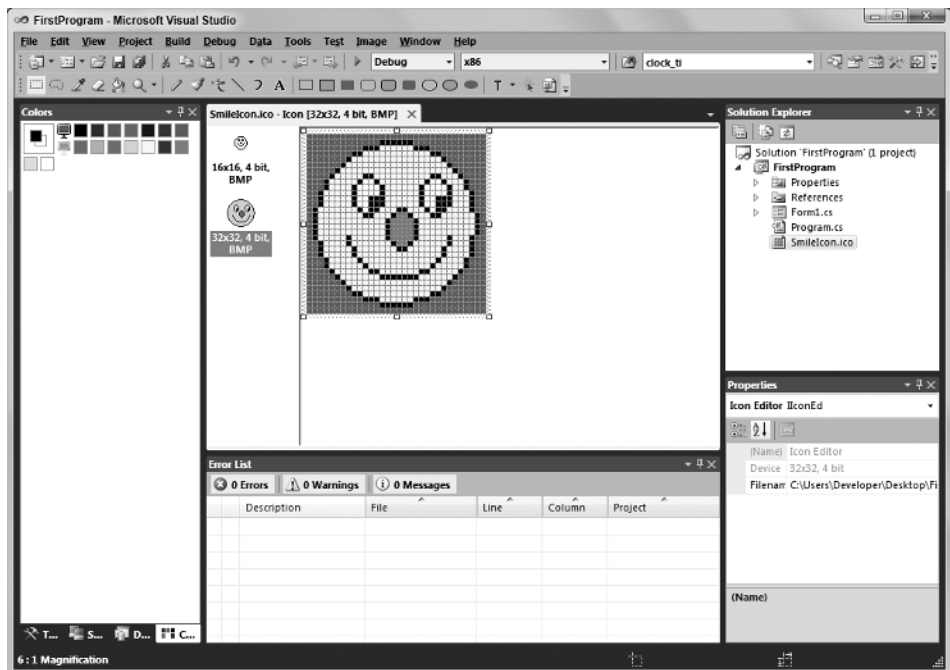


FIGURE 1-7

6. Make a cursor that looks similar to the icon you built in Exercise 5. After you create the cursor file, right-click below the list of cursor types (initially just “32×32, 1 bit, BMP”) and select New Image Type to give the cursor the types “16×16, 24 bit, BMP” and “32×32, 24 bit, BMP.” You can copy and paste the images from the icon into the appropriate cursor types. Set each cursor type’s hotspot to be the center of the smiley’s nose.



You can download the solutions to exercises 1, 5, and 6 in the download available on the book’s web site at www.wrox.com or www.CSharpHelper.com/24hour.html. You can find them in the Lesson01 folder.