CHAPTER 1

Introduction to Programming and Visual Basic

TOPICS

Microsoft Visual Basic is a powerful software development system for creating applications that run in Windows XP and Windows Vista. With Visual Basic, you can do the following:

- Create applications with graphical windows, dialog boxes, and menus
- Create applications that work with databases
- Create Web applications and applications that use Internet technologies
- Create applications that display graphics

Visual Basic is a favorite tool among professional programmers. Its combination of visual design tools and BASIC programming language make it intuitive, allowing developers to create powerful real-world applications in a relatively short time.

Before plunging into learning Visual Basic, we will review the fundamentals of computer hardware and software, and then build an understanding of how a Visual Basic application is organized.

1.1 Computer Systems: Hardware and Software

CONCEPT: **Computer systems consist of similar hardware devices and hardware components. This section provides an overview of computer hardware and software organization.**

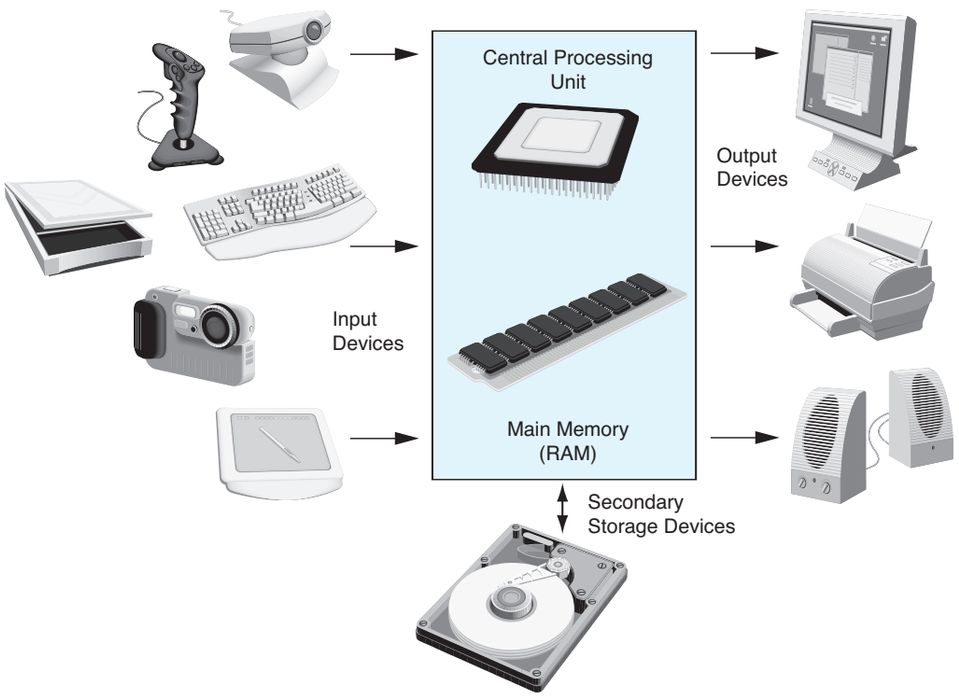
Hardware

The term **hardware** refers to a computer's physical components. A computer, as we generally think of it, is not an individual device, but rather a system of devices. Like the instruments in a symphony orchestra, each device plays its own part. A typical computer system consists of the following major components:

1. The central processing unit (CPU)
2. Main memory
3. Secondary storage devices
4. Input devices
5. Output devices

The organization of a computer system is shown in Figure 1-1.

Figure 1-1 The organization of a computer system



1. The CPU

When a computer is performing the tasks that a program tells it to do, we say that the computer is running or executing the program. The **central processing unit**, or **CPU**, is the part of a computer that actually runs programs. The CPU is the most important component in a computer because without it, the computer could not run software.

A **program** is a set of instructions that a computer's CPU follows to perform a task. The program's instructions are stored in the computer's memory, and the CPU's job is to fetch those instructions, one by one, and carry out the operations that they command. In memory, the instructions are stored as a series of **binary numbers**. A binary number is a sequence of 1s and 0s, such as

```
11011011
```

This number has no apparent meaning to people, but to the computer it might be an instruction to multiply two numbers or read another value from memory.

2. Main Memory

You can think of main memory as the computer's work area. This is where the computer stores a program while the program is running, as well as the data that the program is working with. For example, suppose you are using a word processing program to write an essay for one of your classes. While you do this, both the word processing program and the essay are stored in main memory.

Main memory is commonly known as **random-access memory**, or **RAM**. It is called this because the CPU is able to quickly access data stored at any random location in RAM. RAM is usually a volatile type of memory that is used only for temporary storage while a program is running. When the computer is turned off, the contents of RAM are erased. Inside your computer, RAM is stored in microchips.

3. Secondary Storage

The most common type of secondary storage device is the **disk drive**. A disk drive stores data by magnetically encoding it onto a circular disk. Most computers have a disk drive mounted inside their case. External disk drives, which connect to one of the computer's communication ports, are also available. External disk drives can be used to create backup copies of important data or to move data to another computer.

In addition to external disk drives, many types of devices have been created for copying data, and for moving it to other computers. For many years floppy disk drives were popular. A floppy disk drive records data onto a small floppy disk, which can be removed from the drive. The use of floppy disk drives has declined dramatically in recent years, in favor of superior devices such as USB drives. USB drives are small devices that plug into the computer's USB (universal serial bus) port, and appear to the system as a disk drive. USB drives, which use flash memory to store data, are inexpensive, reliable, and small enough to be carried in your pocket.

Optical devices such as the CD (compact disc) and the DVD (digital versatile disc) are also popular for data storage. Data is not recorded magnetically on an optical disc, but is encoded as a series of pits on the disc surface. CD and DVD drives use a laser to detect the pits and thus read the encoded data. Optical discs hold large amounts of data, and because recordable CD and DVD drives are now commonplace, they are good mediums for creating backup copies of data.

4. Input Devices

Input is any data the computer collects from the outside world. The device that collects the data and sends it to the computer is called an **input device**. Common input devices are the keyboard, mouse, scanner, and digital camera. Disk drives and CD drives can also be considered input devices because programs and data are retrieved from them and loaded into the computer's memory.

5. Output Devices

Output is any data the computer sends to the outside world. It might be a sales report, a list of names, a graphic image, or a sound. The data is sent to an **output device**, which formats and presents it. Common output devices are monitors and printers. Disk drives and CD recorders can also be considered output devices because the CPU sends data to them in order to be saved.

Software

Software refers to the programs that run on a computer. There are two general categories of software: operating systems and application software. An **operating system** or **OS** is a set of programs that manages the computer's hardware devices and controls their processes. Windows XP, Windows Vista, Mac OSX, and Linux are all operating systems.

Application software refers to programs that make the computer useful to the user. These programs, which are generally called applications, solve specific problems or perform general operations that satisfy the needs of the user. Word processing, spreadsheet, and database packages are all examples of application software. As you work through this book, you will develop application software using Visual Basic.

Checkpoint

1.1 List the five major hardware components of a computer system.

1.2 What is main memory? What is its purpose?

1.3 Explain why computers have both main memory and secondary storage.

1.4 What are the two general categories of software?

1.2 Programs and Programming Languages

CONCEPT: **A program is a set of instructions a computer follows in order to perform a task. A programming language is a special language used to write computer programs.**

What Is a Program?

Computers are designed to follow instructions. A computer program is a set of instructions that enables the computer to solve a problem or perform a task. For example, suppose we want the computer to calculate someone's gross pay—a *Wage Calculator* application. Figure 1-2 shows a list of things the computer should do.

Collectively, the instructions in Figure 1-2 are called an **algorithm**. An algorithm is a set of well-defined steps for performing a task or solving a problem. Notice these steps are sequentially ordered. Step 1 should be performed before Step 2, and so on. It is important that these instructions are performed in their proper sequence.

Figure 1-2 Program steps—*Wage Calculator* application

1. Display a message on the screen: *How many hours did you work?*
2. Allow the user to enter the number of hours worked.
3. Once the user enters a number, store it in memory.
4. Display a message on the screen: *How much do you get paid per hour?*
5. Allow the user to enter an hourly pay rate.
6. Once the user enters a number, store it in memory.
7. Once both the number of hours worked and the hourly pay rate are entered, multiply the two numbers and store the result in memory as the amount earned.
8. Display a message on the screen that shows the amount of money earned. The message must include the result of the calculation performed in Step 7.

States and Transitions

It is helpful to think of a running computer program as a combination of states and transitions. Each state is represented by a snapshot (like a picture) of the computer's memory. Using the *Wage Calculator* application example from Figure 1-2, the following is a memory snapshot taken when the program starts:

Program Starting State	
hours worked	??
hourly pay rate	??
amount earned	??

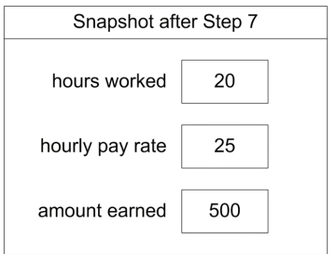
In Step 3, the number of hours worked by the user is stored in memory. Suppose the user enters the value 20. A new program state is created:

Snapshot after Step 3	
hours worked	20
hourly pay rate	??
amount earned	??

In Step 6, the hourly pay rate entered by the user is stored in memory. Suppose the user enters the value 25. The following memory snapshot shows the new program state:

Snapshot after Step 6	
hours worked	20
hourly pay rate	25
amount earned	??

In Step 7, the application calculates the amount of money earned, saving it in a variable. The following memory snapshot shows the new program state:



The memory snapshot produced by Step 7 represents the final program state.

Programming Languages

In order for a computer to perform instructions such as the wage calculator algorithm, the steps must be converted to a format the computer can process. As mentioned earlier, a program is stored in memory as a series of binary numbers. These numbers are known as **machine language instructions**. The CPU only processes instructions written in machine language. Our *Wage Calculator* application might look like the following at the moment when it is executed by the computer:

```
10101101110101000111100001101110100011110001110011010101110 etc.
```

The CPU interprets these binary or machine language numbers as commands. As you might imagine, the process of encoding an algorithm in machine language is tedious and difficult. Programming languages, which use words instead of numbers, were invented to ease this task. Programmers can write their applications in programming language statements, and then use special software called a **compiler** to convert the program into machine language. Names of some popular recent programming languages are shown in Table 1-1. This list is only a small sample—there are thousands of programming languages.

Visual Basic is more than just a programming language. It is a programming environment, with tools for creating screen elements and programming language statements. Although Visual Basic, as a whole, is radically different from the original BASIC programming language, its programming statements are similar.

Table 1-1 Popular programming languages

Language	Description
Visual Basic, C#	Popular programming languages for building Windows and Web applications. Use a graphical user interface.
C, C++	Powerful advanced programmng languages that emphasize flexibility and fast running times. C++ is also object-oriented.
Java	Flexible and powerful programming language that runs on many different computer systems. Often used to teach object-oriented programming.
Python	Simple, yet powerful programming language used for graphics and small applications.
PHP	Programming language used for creating interactive Web sites.
JavaScript	Scripting language used in Web applications that provides rich user interfaces for Web browsers.

Procedural and Object-Oriented Programming

There are primarily two methods of programming used today: procedural programming and object-oriented programming.

Procedural Programming

The earliest programming languages were procedural. Procedural programming means that a program is made of one or more procedures. A **procedure** is a set of programming language statements that are executed by the computer. The statements might gather input from the user, manipulate information stored in the computer's memory, perform calculations, or any other operation necessary to complete its task. The wage calculator algorithm shown in Figure 1-2 can be thought of as a procedure. If the algorithm's eight steps are performed in order, one after the other, it will succeed in calculating and displaying the user's gross pay.

Procedural programming was the standard when users were interacting with text-based computer terminals. For example, Figure 1-3 shows the screen of an older MS-DOS computer running a program that performs the wage calculator algorithm. The user has entered the numbers shown in bold.

Object-Oriented Programming

Object-oriented programming or **OOP** is an industry standard model for designing and coding programs. When designing applications, designers use real-world objects to express patterns, called classes in software. An example is a student registration application, in which we would choose students, transcripts, and accounts as possible classes. The program we write would create objects, or instances of these classes.

Classes contain attributes, expressed as variables. For example, a class named Account would probably contain attributes such as balance, account ID, and payment history. In Visual Basic, classes are used to describe the objects that appear on the screen. When a program runs, these objects are created and displayed.

Figure 1-3 *Wage Calculator* application

```
How many hours did you work? 10
How much are you paid per hour? 15
You have earned $150.00
C>_
```

Graphical User Interface

In text-based environments using procedural programs, the user responds to the program. Modern operating systems, such as the Windows family, use a **graphical user interface**, or **GUI** (pronounced *gooey*). Although GUIs have made programs friendlier and easier to interact with, they have not simplified the task of programming. GUIs require on-screen elements such as windows, dialog boxes, buttons, and menus. The program must handle the user's interactions with these on-screen elements, in any order the user might choose to select them. No longer does the user respond to a program—now the program responds to a user.

GUIs have helped influence the shift from procedural programming to object-oriented programming. Whereas procedural programming is centered on creating procedures, object-oriented programming is centered on creating **objects**. An object is a programming

element that contains data and actions. The data contained in an object is known as its **attributes**. In Visual Basic, an object's attributes are called **properties**. The actions that an object performs are known as the object's **methods**. The object is, conceptually, a self-contained unit consisting of data (properties) and actions (methods).

Perhaps the best way to understand objects is to experience a program that uses them. The following steps guide you through the process of running a demonstration program located on the Student CD. The program was created with Visual Basic.

Required Software Setup

To use this book, you must install two pieces of software:

- **Microsoft Visual Studio 2008** or **Visual Basic 2008 Express**. From now on, we will drop the "2008" from both names. If you will be covering Chapter 11, you will also need **Visual Web Developer 2008 Express**.
- **The student sample program files**, located on the CD-ROM packaged with this book.

VideoNote
Forms, Controls, and Properties

Installing the Sample Program Files

The Student CD included with this book contains sample program files that are required in many of the tutorials. Before you can use these files you must copy them from the Student CD to your computer's hard drive. If you are working in a college computer lab, it is possible that the sample program files have already been copied to the computers in the lab. If this is the case, your professor will tell you where they are located. In Tutorial 1-1, you will execute the *Wage Calculator* application.

Tutorial 1-1:
Running the *Wage Calculator* application

Assuming you have installed Visual Studio or Visual Basic Express and the sample programs from the Student CD on your computer, you're ready to begin Tutorial 1-1.

Step 1: In Windows, double-click the *My Computer* icon or the *Windows Explorer* icon.

Step 2: In Windows, navigate to the folder on your computer containing the student sample programs. Then navigate to the *Chap1\Wage Calculator\bin* folder. Double-click the file *Wage Calculator.exe* (the *.exe* filename extension may not be visible). The program's window should display.

The window shown in Figure 1-4 can be thought of as an object. In Visual Basic terminology, this window object is known as a **Form object**. The form also contains numerous other objects. As shown in Figure 1-5, it has four Label objects, two TextBox objects, and two Button objects. In Visual Basic, these objects are known as **controls**.

The appearance of a screen object, such as a form or other control, is determined by the object's properties. For example, each of the Label controls has a property known as Text. The value stored in the **Text property** becomes the text displayed by the label. For instance, the Text property of the topmost label on the form is set to the value *Number of Hours Worked*. Beneath it is another Label control, whose Text property is set to

Figure 1-4 *Wage Calculator* screen

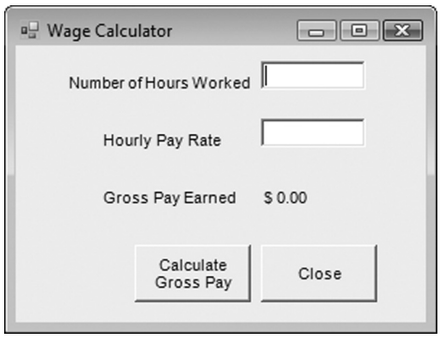
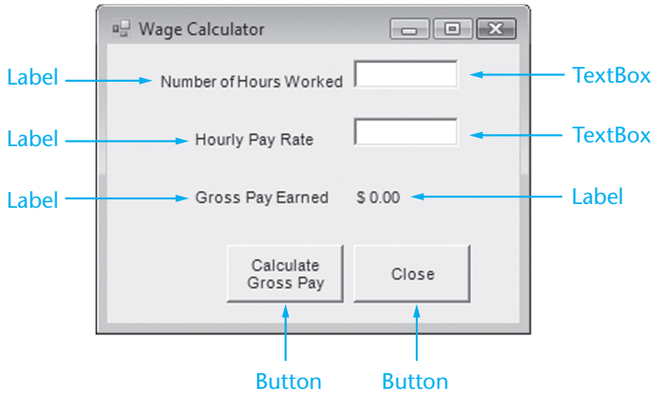


Figure 1-5 Types of controls



Hourly Pay Rate. The Button controls also have a Text property. The Text property of the leftmost button is set to *Calculate Gross Pay*, and the rightmost button has its Text property set to *Close*. Even the window, or form, has a Text property, which determines the text displayed in the window's title bar. *Wage Calculator* is the value stored in this form's Text property. Part of the process of creating a Visual Basic application is deciding what values to store in each object's properties.

VideoNote
Event-Driven Programming

Event-Driven Programming

Programs that operate in a GUI environment must be **event-driven**. An event is an action that takes place within a program, such as the clicking of a control. All Visual Basic controls are capable of detecting various events. For example, a Button control can detect when it has been clicked and a TextBox control can detect when its contents have changed.

Names are assigned to all of the events that can be detected. For instance, when the user clicks a Button control, a `Click` event occurs. When the contents of a TextBox control changes, a `TextChanged` event occurs. If you wish for a control to respond to a specific event, you must write a special type of method known as an **event procedure**. An event procedure is a method that is executed when a specific event occurs. If an event occurs, and there is no event procedure to respond to that event, the event is ignored.

Part of the Visual Basic programming process is designing and writing event procedures. Tutorial 1-2 demonstrates an event procedure using the *Wage Calculator* application you executed in Tutorial 1-1.

Tutorial 1-2:
Running an application that demonstrates event procedures

Step 1: With the *Wage Calculator* application from Tutorial 1-1 still running, enter the value **10** in the first TextBox control. This is the number of hours worked.

Step 2: Press the Tab key. Notice that the cursor moves to the next TextBox control. Enter the value **15**. This is the hourly pay rate. The window should look like that shown in Figure 1-6.

Step 3: Click the *Calculate Gross Pay* button. Notice that in response to the mouse click, the application multiplies the values you entered in the TextBox controls and displays the result in a Label control. This action is performed by an event procedure that responds to the button being clicked. The window should look like that shown in Figure 1-7.

Figure 1-6 Text boxes filled in on the *Wage Calculator* form

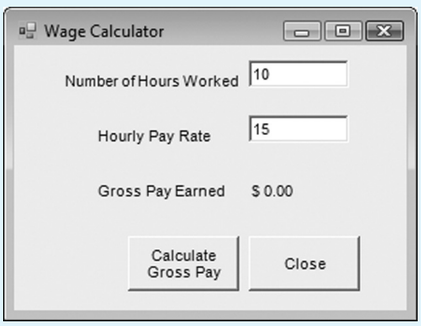
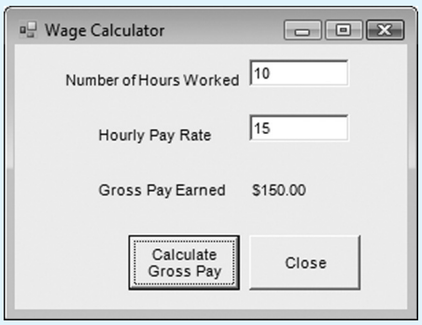


Figure 1-7 Gross pay calculated



Step 4: Next, click the *Close* button. The application responds to this event by terminating. This is because an event procedure closes the application when the button is clicked.

This simple application demonstrates the essence of object-oriented, event-driven programming. In the next section, we examine the controls and event procedures more closely.

1.3 More about Controls and Programming

CONCEPT: **As a Visual Basic programmer, you must design and create the two major components of an application: the GUI elements (forms and other controls) and the programming statements that respond to and/or perform actions (event procedures).**

While creating a Visual Basic application, you will spend much of your time doing three things: creating the GUI elements that make up the application's user interface, setting the properties of the GUI elements, and writing programming language statements that respond to events and perform other operations. In this section, we take a closer look at these aspects of Visual Basic programming.

Visual Basic Controls

In the previous section, you saw examples of several GUI elements, or controls. Visual Basic provides a wide assortment of controls for gathering input, displaying information, selecting values, showing graphics, and more. Table 1-2 lists some of the commonly used controls.

Table 1-2 Visual Basic controls

Control Type	Description
CheckBox	A box that is checked or unchecked when clicked with the mouse
ComboBox	A control that is the combination of a ListBox and a TextBox
Button	A rectangular button-shaped object that performs an action when clicked with the mouse
Form	A window, onto which other controls may be placed
GroupBox	A rectangular border that functions as a container for other controls
HScrollBar	A horizontal scroll bar that, when moved with the mouse, increases or decreases a value
Label	A box that displays text that cannot be changed or entered by the user
ListBox	A box containing a list of items
RadioButton	A round button that is either selected or deselected when clicked with the mouse
PictureBox	A control that displays a graphic image
TextBox	A rectangular area in which the user can enter text, or the program can display text
VScrollBar	A vertical scroll bar that, when moved with the mouse, increases or decreases a value

If you have any experience using Microsoft Windows, you are already familiar with most of the controls listed in Table 1-2. The Student CD contains a simple demonstration program in Tutorial 1-3 that shows you how a few of them work.

Tutorial 1-3:
Running an application that demonstrates various controls

Step 1: In Windows, navigate to the location where the sample program files have been copied from the Student CD.

Step 2: Navigate to the the *Chap1\Controls Tour\bin* folder.

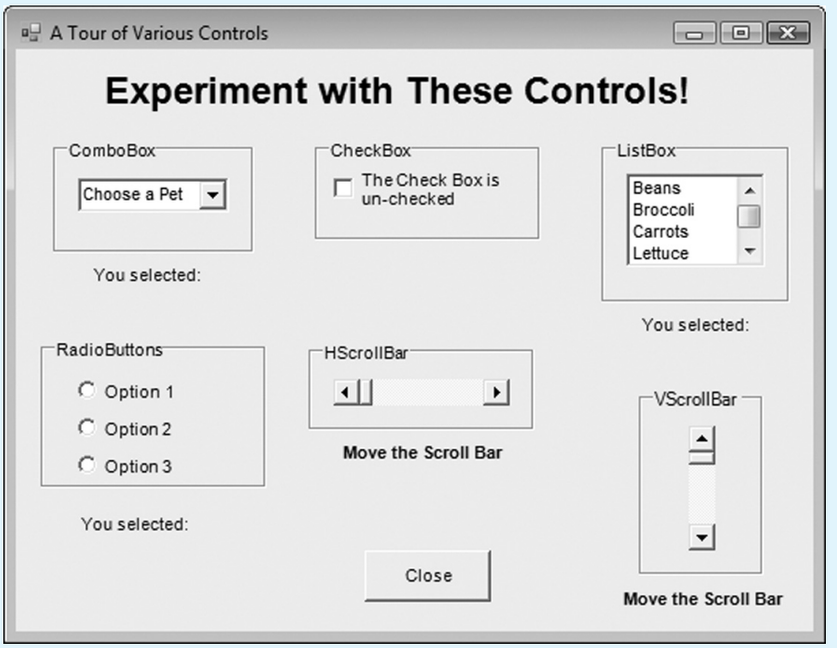
Step 3: Double-click the file *Program2.exe*.

Step 4: Once the program loads and executes, the window shown in Figure 1-8 should appear on the screen.

Step 5: The program presents several Visual Basic controls. Experiment with each one, noticing the following actions, which are performed by event procedures:

- When you click the small down arrow ( ) in the ComboBox control, you see a list of pets. When you select one, the name of the pet appears below the combo box.
- When you click the CheckBox control, its text changes to indicate that the check box is checked or unchecked.
- When you click an item in the ListBox control, the name of that item appears below the list box.
- When you select one of the RadioButton controls, the text below them changes to indicate which one you selected. You may only select one at a time.

Figure 1-8 Control demonstration screen



- You move the horizontal scroll bar (HScrollBar) and the vertical scroll bar (VScrollBar) by doing the following:
  - Clicking either of the small arrows at each end of the bar
  - Clicking inside the bar on either side of the slider
  - Clicking on the slider and while holding down the mouse button, moving the mouse to the right or left for the horizontal scroll bar, or up or down for the vertical scroll bar.

  When you move either of the scroll bars, the text below it changes to a number. Moving the scroll bar in one direction increases the number, and moving it in the other direction decreases the number.

Step 6: Click the *Close* button to end the application.

The Name Property

The appearance of a control is determined by its properties. Some properties, however, establish nonvisual characteristics. An example is the control's **Name property**. When the programmer wishes to manipulate or access a control in a programming statement, he or she must refer to the control by its name.

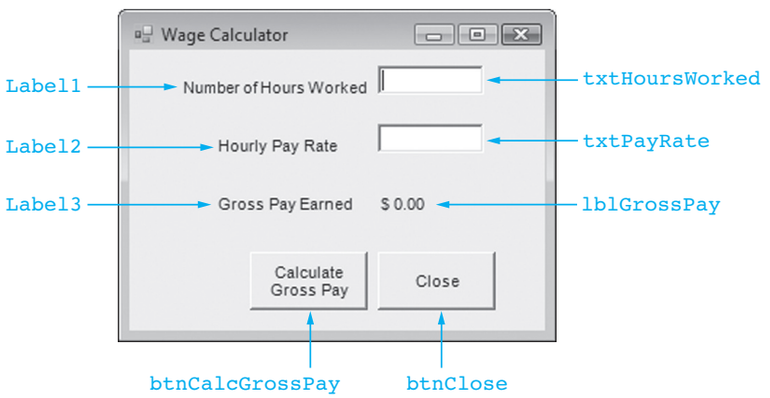
When you create a control in Visual Basic, it automatically receives a default name. The first Label control created in an application receives the default name `Label1`. The second Label control created receives the default name `Label2`, and the default names continue in this fashion. The first TextBox control created in an application is automatically named `TextBox1`. As you can imagine, the names for each subsequent TextBox control are `TextBox2`, `TextBox3`, and so on. You can change the control's default name to something more descriptive.

Table 1-3 lists all the controls, by name, in the *Wage Calculator* application (Section 1.2), and Figure 1-9 shows where each is located.

Table 1-3 *Wage Calculator* controls

Control Name	Control Type	Description
`Label1`	Label	Displays the message *Number of Hours Worked*
`Label2`	Label	Displays the message *Hourly Pay Rate*
`Label3`	Label	Displays the message *Gross Pay Earned*
`txtHoursWorked`	TextBox	Allows the user to enter the number of hours worked
`txtPayRate`	TextBox	Allows the user to enter the hourly pay rate
`lblGrossPay`	Label	Displays the gross pay, after the `btnCalcGrossPay` button has been clicked
`btnCalcGrossPay`	Button	When clicked, multiplies the number of hours worked by the hourly pay rate
`btnClose`	Button	When clicked, terminates the application

Figure 1-9 *Wage Calculator* controls



Control Naming Rules and Conventions

Three controls shown in Figure 1-9 (`Label1`, `Label2`, and `Label3`), still have their default names. The other five controls have programmer-defined names because those controls play an active role in the application's event procedures, and their names appear in the application's programming statements. Any control that activates programming statements or whose name appears in a programming statement should have a descriptive, programmer-defined name.

NOTE: Some programmers prefer to give all the controls in their application meaningful names, including ones that do not activate programming statements or whose names do not appear in programming statements.

Although you have a great deal of flexibility in naming controls, there are some standard rules. First, control names must start with a letter. The remaining characters may be letters, digits, or underscore characters only. You cannot use spaces, special symbols, or punctuation characters in a control name. In addition to the mandatory rules, there are three general guidelines to follow when naming controls:

1. The first three letters of the name should be a lowercase prefix indicating the control's type. In the *Wage Calculator* application, programmer-defined names use the following standard three-letter prefixes:

   - `lbl` indicates a Label control.
   - `txt` indicates a TextBox control.
   - `btn` indicates a Button control.

   There are standard prefixes for other controls as well. They are discussed in Chapter 2.
2. The first letter after the prefix should be uppercase. In addition, if the name consists of multiple words, the first letter of each word should be capitalized. This makes the name more readable. For example, `txtHoursWorked` is easier to read than `txthoursworked`.
3. The part of the control name that appears after the three-letter prefix should describe the control's purpose in the application. This makes the control name very helpful to anyone reading the application's programming statements. For example, it is evident that the `btnCalcGrossPay` control is a button that calculates the gross pay.

These guidelines are not mandatory rules, but they are standard conventions that programmers follow. You should use these guidelines when naming the controls in your applications as well. Table 1-4 describes several fictitious controls and suggests appropriate programmer-defined names for them.

Table 1-4 Programmer-defined control name examples

Control Description	Suggested Name
A text box in which the user enters his or her age	`txtAge`
A button that, when clicked, calculates the total of an order	`btnCalcTotal`
A label that is used to display the distance from one city to another	`lblDistance`
A text box in which the user enters his or her last name	`txtLastName`
A button that, when clicked, adds a series of numbers	`btnAddNumbers`

Checkpoint

1.5 What is an algorithm?

1.6 Why were computer programming languages invented?

1.7 What are the two methods of programming used today?

1.8 What does event-driven mean?

1.9 Describe the difference between a property and a method.

1.10 Why should the programmer change the name of a control from its default name?

1.11 If a control has the programmer-defined name `txtRadius`, what type of control is it?

1.12 What is the default name given to the first TextBox control created in an application?

1.13 Is `txtFirst+LastName` an acceptable control name? Why or why not?

Programming an Application

Let's look at some source code from the *Wage Calculator* application you saw in Tutorial 1-3. The code shown here is part of the event procedure that executes when the user clicks the *Calculate* button. The line numbers that appear to the left of the statements are not part of the code, but they will help us provide a description of each statement.

```
1:   Dim dblHoursWorked As Double
2:   Dim dblPayRate As Double
3:   Dim dblGrossPay As Double
4:   dblHoursWorked = txtHoursWorked.Text
5:   dblPayRate = txtPayRate.Text
6:   dblGrossPay = dblHoursWorked * dblPayRate
7:   lblGrossPay.Text = dblGrossPay.ToString("c")
```

- Lines 1, 2, and 3 declare variables to hold the hours worked, pay rate, and gross pay.
- Line 4 copies the contents of the TextBox control named `txtHoursWorked` into the `dblHoursWorked` variable. Each TextBox control has a Text property that holds the contents of the TextBox at runtime. By *contents*, we mean the text typed by the user.
- Line 5 copies the contents of the TextBox named `txtPayRate` into the `dblPayRate` variable.
- Line 6 calculates the employee's gross pay by multiplying the hours worked by the hourly pay rate. The calculated value is stored in the variable named `dblGrossPay`.
- Line 7 converts the number in the `dblGrossPay` variable into a string with currency format, such as $500.00 and copies the string to the Label control named `lblGrossPay`.

Table 1-5 refers to the fundamental elements in a Visual Basic program.

Table 1-5 Visual Basic language elements

Language Element	Description
Keywords	Words that have a special meaning in a programming language. Keywords may only be used for their intended purpose. Some examples in Visual Basic are `Private`, `Sub`, `Dim`, and `End`.
Programmer-defined names	Words or names defined by the programmer.
Operators	Operators perform operations on one or more **operands**. An operand is usually a piece of data, such as a number. Examples of operators include +, –, *, and /.
Remarks	Also known as **comments**, remarks are notes that explain the purpose of statements or sections of code. Although remarks are part of an application's code, they are ignored by the compiler. They are intended for the programmer or others who might read the application's code.
Syntax	Rules that must be followed when constructing a method. Syntax dictates how keywords, operators, and programmer-defined names may be used.

Keywords

There are two keywords in line 1: `Dim` and `Double`. They are repeated in lines 2 and 3. Each of these words has a special meaning in Visual Basic and can only be used for its intended purpose. As you will see, a programmer is allowed to make up his or her own names for certain things in a program. Keywords, however, are reserved and cannot be used for anything other than their designated purpose. Part of learning a programming language is learning what the keywords are, what they mean, and how to use them.

Programmer-Defined Names (Identifiers)

The word `dblGrossPay`, which appears in lines 6 and 7, is a programmer-defined name, or **identifier**. It is not part of the Visual Basic language, but rather a name made up by the programmer to identify a variable. Variables are named memory locations that hold data while a program is running.

Operators

In line 6 the following statement appears:

```
dblGrossPay = dblHoursWorked * dblPayRate
```

The = and * symbols are operators, which perform operations on pieces of data known as operands. The * operator multiplies two operands, and the = operator stores a value in a variable or a property.

Comments (Remarks)

Comments, or remarks, help the reader of a program understand the purpose of program statements. Sometimes you (the programmer) will have to reread and understand your own code. Comments are a great way to remind you of what you were thinking when you created the program. The following are examples of comments:

```
'Convert the values in the text box to numbers,
'and calculate the gross pay.
```

A comment must begin with either an apostrophe (`'`) or the `REM` keyword. When a program runs, the computer ignores comments.

You should always add descriptive comments to your code. The extra time it takes is well spent. Someday you may have to modify or maintain code written by another programmer and you will appreciate the time spent to comment the code!

Syntax

Each statement shown in the sample code is written according to the rules of Visual Basic. The rules, known collectively as **language syntax**, define the correct way to use keywords, operators, and programmer-defined names. If a programming statement violates the Visual Basic syntax, the application will not run until it is corrected.

1.4 The Programming Process

CONCEPT: **The programming process consists of several steps, which include designing, creating, testing, and debugging activities.**

Imagine building a bridge without a plan. How could it be any easier to create a complex computer program without designing its appearance and behavior? In this section, we introduce some of the most important knowledge you will gain from this book—how to begin creating a computer application. Regardless of which programming language you use in the future, good program design principles always apply.

Steps for Developing a Visual Basic Application

1. Clearly define what the application is to do.
2. Visualize the application running on the computer and design its user interface.
3. Make a list of the controls needed.
4. Define the values of each control's relevant properties.
5. Make a list of methods needed for each control.
6. Create a flowchart or pseudocode version of each method.
7. Check the flowchart or pseudocode for errors.
8. Start Visual Basic and create the forms and other controls identified in Step 3.
9. Write the code for the event procedures and other methods created in Step 6.
10. Attempt to run the application. Correct any syntax errors found and repeat this step as many times as necessary.
11. Once all syntax errors are corrected, run the program with test data for input. Correct any runtime errors. Repeat this step as many times as necessary.

These steps emphasize the importance of planning. Just as there are good ways and bad ways to paint a house, there are good ways and bad ways to write a program. A good program always begins with planning.

With the *Wage Calculator* application as our example, let's look at each of these steps in greater detail.

1. Clearly define what the application is to do.

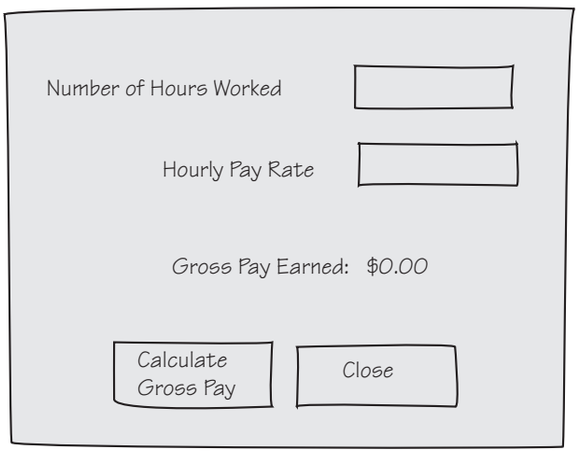
This step requires that you identify the purpose of the application, the information to be input, the processing to take place, and the desired output. For example, the requirements for the *Wage Calculator* application are as follows:

Purpose:	To calculate the user's gross pay
Input:	Number of hours worked, hourly pay rate
Process:	Multiply number of hours worked by hourly pay rate. The result is the user's gross pay
Output:	Display a message indicating the user's gross pay

2. Visualize the application running on the computer and design its user interface.

Before you create an application on the computer, first you should create it in your mind. Step 2 is the visualization of the program. Try to imagine what the computer screen will look like while the application is running. Then, sketch the form or forms in the application. For instance, Figure 1-10 shows a sketch of the form presented by the *Wage Calculator* application.

Figure 1-10 Sketch of the *Wage Calculator* form



3. Make a list of the controls needed.

The next step is to list all the controls needed. You should assign names to all controls that will be accessed or manipulated in the application code, and provide a brief description of each control. Table 1-6 lists the controls in the *Wage Calculator* application.

4. Define the values of each control's relevant properties.

Other than Name, Text is the only control property modified in the *Wage Calculator* application. Table 1-7 lists the value of each control's Text property.

Table 1-6 *Wage Calculator* controls

Control Type	Control Name	Description
Form	(Default)	A small form that will serve as the window onto which the other controls will be placed
Label	(Default)	Displays the message *Number of Hours Worked*
Label	(Default)	Displays the message *Hourly Pay Rate*
Label	(Default)	Displays the message *Gross Pay Earned*
TextBox	`txtHoursWorked`	Allows the user to enter the number of hours worked
TextBox	`txtPayRate`	Allows the user to enter the hourly pay rate
Label	`lblGrossPay`	Displays the gross pay, after the `btnCalcGrossPay` button has been clicked
Button	`btnCalcGrossPay`	When clicked, multiplies the number of hours worked by the hourly pay rate; stores the result in a variable and displays it in the `lblGrossPay` label
Button	`btnClose`	When clicked, terminates the application

Table 1-7 *Wage Calculator* control values

Control Type	Control Name	Text
Form	(Default)	`"Wage Calculator"`
Label	(Default)	`"Number of Hours Worked"`
Label	(Default)	`"Hourly Pay Rate"`
Label	(Default)	`"Gross Pay Earned"`
Label	`lblGrossPay`	`"$0.00"`
TextBox	`txtHoursWorked`	`""`
TextBox	`txtPayRate`	`""`
Button	`btnCalcGrossPay`	`"Calculate Gross Pay"`
Button	`btnClose`	`"Close"`

5. Make a list of methods needed for each control.

Next, you should list the event procedures and other methods you will write. There are only two event procedures in the *Wage Calculator* application. Table 1-8 lists and describes them. Notice the Visual Basic names for the event procedures. `btnCalcGrossPay_Click` is the name of the procedure invoked when the `btnCalcGrossPay` button is clicked and `btnClose_Click` is the event procedure that executes when the `btnClose` button is clicked.

Table 1-8 *Wage Calculator* event procedures

Method	Description
`btnCalcGrossPay_Click`	Multiplies the number of hours worked by the hourly pay rate; these values are entered into the `txtHoursWorked` and `txtPayRate` TextBox controls and the result is stored in the `lblGrossPay.Text` property
`btnClose_Click`	Terminates the application

6. Create a flowchart or pseudocode version of each method.

A **flowchart** is a diagram that graphically depicts the flow of a method. It uses boxes and other symbols to represent each step. Figure 1-11 shows a flowchart for the `btnCalcGrossPay_Click` event procedure.

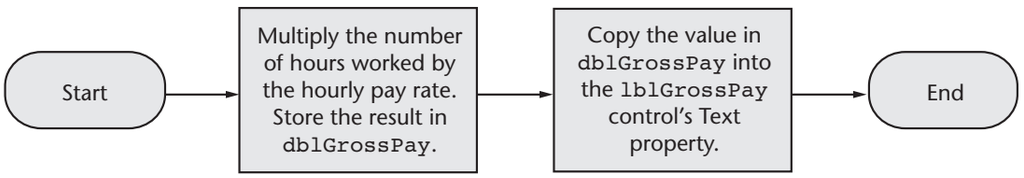
There are two types of boxes in the flowchart shown in Figure 1-11: ovals and rectangles. The flowchart begins with an oval labeled *Start* and ends with an oval labeled *End*. The rectangles represent a computational process or other operation. Notice that the symbols are connected with arrows that indicate the direction of the program flow.

Many programmers prefer to use pseudocode instead of flowcharts. **Pseudocode** is a cross between human language and a programming language. Although the computer can't understand pseudocode, programmers often find it helpful to plan an algorithm in

a language that's almost a programming language but still very readable by humans. The following is a pseudocode version of the `btnCalcGrossPay_Click` event procedure:

Store Number of Hours Worked × Hourly Pay Rate in dblGrossPay.
Copy dblGrossPay in lblGrossPay.Text.

Figure 1-11 Flowchart for `btnCalcGrossPay_Click` event procedure



7. Check the code for errors.

In this phase the programmer reads the flowcharts and/or pseudocode from the beginning and steps through each operation, pretending that he or she is the computer. The programmer jots down the current contents of variables and properties that change and sketches what the screen looks like after each output operation. By checking each step, a programmer can locate and correct many errors.

8. Start Visual Studio and create the forms and other controls identified in Step 3.

This step is the first actual work done on the computer. Here, the programmer uses Visual Studio to create the application's user interface and arrange the controls on each form.

9. Write the code for the event procedures and other methods created in Step 6.

This is the second step performed on the computer. The event procedures and other methods may be converted into code and entered into the computer using Visual Studio.

10. Attempt to run the application. Correct any syntax errors found and repeat this step as many times as necessary.

If you have entered code with syntax errors or typing mistakes, this step will uncover them. A **syntax error** is the incorrect use of a programming language element, such as a keyword, operator, or programmer-defined name. Correct your mistakes and repeat this step until the program runs.

11. Once all syntax errors are corrected, run the program with test data for input. Correct any runtime errors. Repeat this step as many times as necessary.

Runtime errors (errors found while running the program) are mistakes that do not prevent an application from executing but cause it to produce incorrect results. For example, a mistake in a mathematical formula is a common type of **runtime error**. When runtime errors are found in a program, they must be corrected and the program retested. This step must be repeated until the program reliably produces satisfactory results.

Checkpoint

1.14 What four items should be identified when defining what a program is to do?

1.15 Describe the importance of good planning in the process of creating a Visual Basic application.

1.16 What does it mean to visualize a program running? What is the value of such an activity?

1.17 What is a flowchart?

1.18 What is pseudocode?

1.19 What is a runtime error?

1.20 What is the purpose of testing a program with sample data or input?

1.21 How much testing should you perform on a new program?

1.5 Visual Studio and Visual Basic Express (the Visual Basic Environment)

CONCEPT: **Visual Studio and Visual Basic Express consist of tools that you use to build Visual Basic applications. The first step in using Visual Basic is learning about these tools.**

NOTE: The programs in this book can be written using either Microsoft Visual Studio or Microsoft Visual Basic Express. There are only minor differences between the two products. In cases where they work identically, we will refer to them as **Visual Studio**.

In Chapter 2 you will build your first Visual Basic application. First, you need to know how to start Visual Studio and understand its major components. Visual Studio is an **integrated development environment** (IDE) that provides the necessary tools for creating, testing, and debugging software. Visual Studio can be used to create applications not only with Visual Basic, but also with other languages such as Visual C++ and C#. Tutorial 1-4 guides you through the Visual Studio startup process and gives you a hands-on tour of its tools for creating Visual Basic applications.

Tutorial 1-4:
Starting Visual Studio

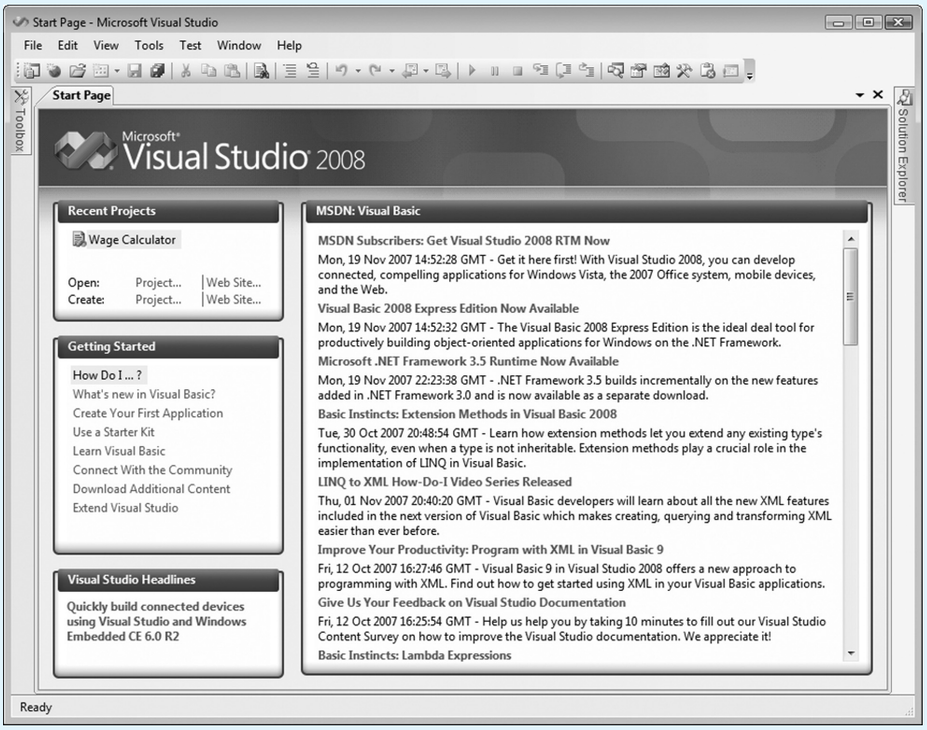
The following steps guide you through the Visual Studio startup process.

Step 1: Click the *Start* button and open the *All Programs* menu (or *Programs* menu for earlier versions of MS Windows). Ask your classroom instructor for the location in the menu of either Visual Studio 2008 or Visual Basic 2008 Express—whichever one your class will be using.

TIP: If you are using Visual Studio rather than Visual Basic Express, the first time you run the software, you may see a window entitled *Choose Default Environment Settings*. Select *Visual Basic Development Settings* from the list and click the *Start Visual Studio* button. (You can always change the settings later as the window explains.)

Step 2: Next, you should see the *Start Page*, as shown in Figure 1-12. Your screen may not appear exactly as shown in the figure, because some of the information shown here comes from the Web. Later in this chapter, we will show you how to control the appearance of Visual Studio.

Figure 1-12 Visual Studio *Start Page*



TIP: If you do not see the *Start Page* shown in Figure 1-12, click *View* in the menu bar, click *Other Windows*, and then click *Start Page*.

Step 3: Visual Studio allows you to set certain defaults for Visual Basic programming. In the menu bar, click *Tools* and then click *Options* (Figure 1-13). If you are using Visual Basic Express, you will not see the *Device Tools* option.

Step 4: In the *Options* dialog, under *Projects and Solutions*, select *General* (Figure 1-14). To the right of the Visual Studio projects location edit box, click the *Browse* button [...]. Select a new location to save your programming projects (you may be assigned a location by your instructor).

Figure 1-13 Visual Studio *Options* dialog

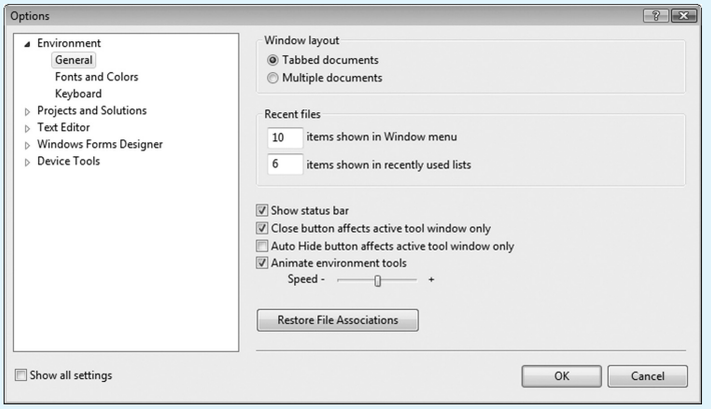
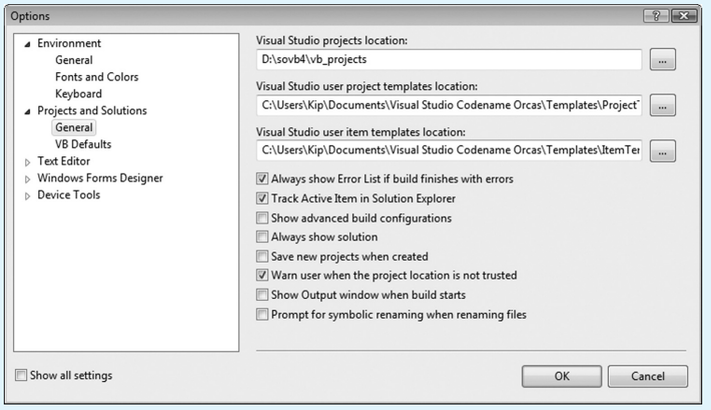
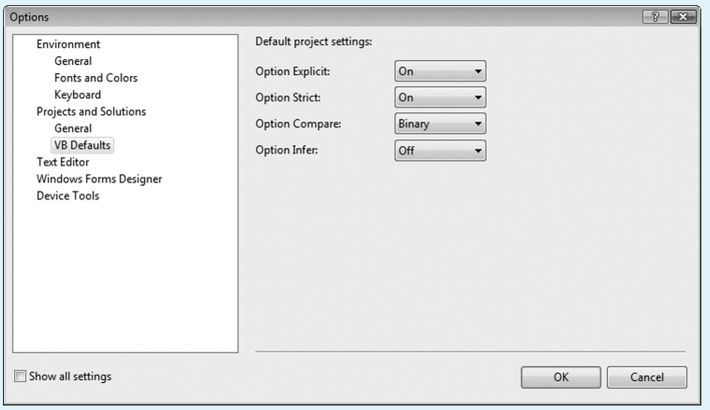


Figure 1-14 Setting the Visual Studio projects location



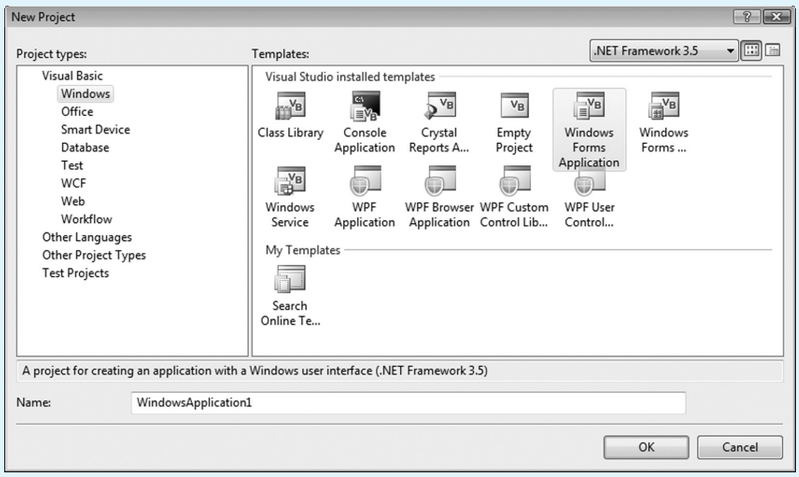
Step 5: Still in the *Options* dialog, under *Projects and Solutions*, select *VB Defaults* (Figure 1-15). Set *Option Strict* to *On* and set *Option Infer* to *Off*. (We will explain what these options mean in Chapter 3.) Click *OK* to close the dialog.

Figure 1-15 Setting *VB Defaults* in the *Options* dialog



Step 6: Each application you create with Visual Studio is called a **project**. Now you will start a new project. In the menu bar, click *File*, then click *New Project*. The ***New Project* dialog box** shown in Figure 1-16 appears. (Visual Basic Express shows fewer templates than Visual Studio.)

Figure 1-16 *New Project* dialog box



Step 7: The left pane, labeled *Project types*, lists the different programming languages available in Visual Studio. (Visual Basic Express shows only the right pane, with a smaller number of templates.) The right pane, labeled *Templates*, lists the types of applications you can create in the selected language. If you are using Visual Studio, select *Windows* under *Visual Basic* in the *Project types* pane. For both Visual Studio and Visual Basic Express, select *Windows Forms Application* in the *Templates* pane.

Step 8: The *Name* text box is where you enter the name of your project. Visual Studio automatically fills this box with a default name. In Figure 1-16 the default name is *WindowsApplication1*. Change the project name to **`Tutorial 1-4`** and click the *OK* button.

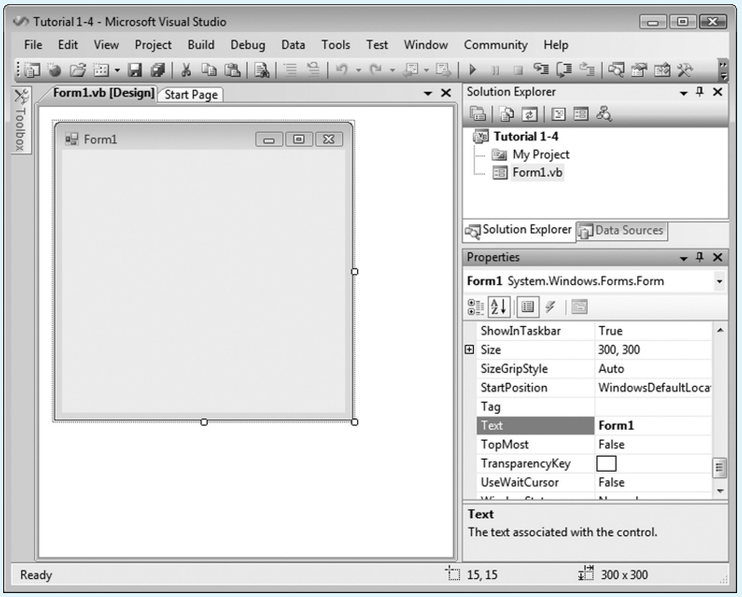
NOTE: A project consists of numerous files. When you begin a new project, Visual Studio stores the files in a folder with the same name as the project. As you create additional projects, you will find that default names such as *WindowsApplication1* do not help you remember what each project does. Therefore, you should always change the name of a new project to something that describes the project's purpose.

Step 9: Select *Save All* from the *File* menu. The *Location* text box shows where the project folder will be created on your system. If you wish to change the location, click the *Browse* button and select the desired drive and folder.

Step 10: Click the *Save* button. You should now see the Visual Studio window similar to the one shown in Figure 1-17.

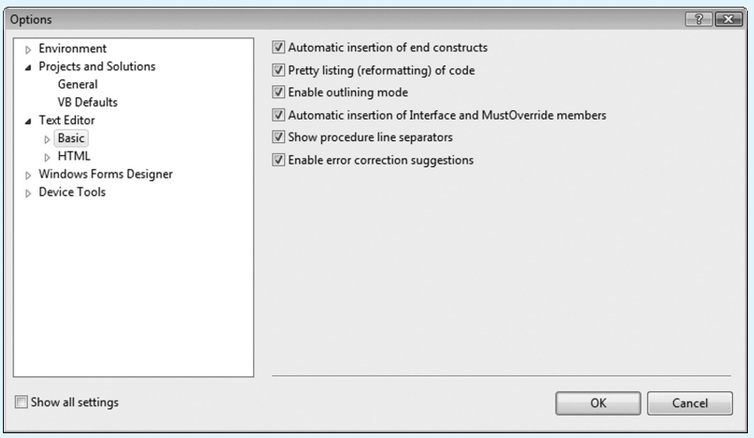
NOTE: Visual Studio is customizable. Your screen might not appear exactly as shown in Figure 1-17. As you continue through this chapter, you will learn how to arrange the screen elements in different ways.

Figure 1-17 Visual Studio environment with a new project open



Step 11: Now you will set some of the Visual Studio options so your screens and code will appear as the examples shown in this book. Click *Tools* on the menu bar. On the *Tools* menu, click *Options . . .* The ***Options* dialog box** appears. In the left pane click *Text Editor*, and then click *Basic*, as shown in Figure 1-18. (If you are using Visual Basic 2008 Express, simply click *Text Editor Basic*.) Be sure all options are checked.

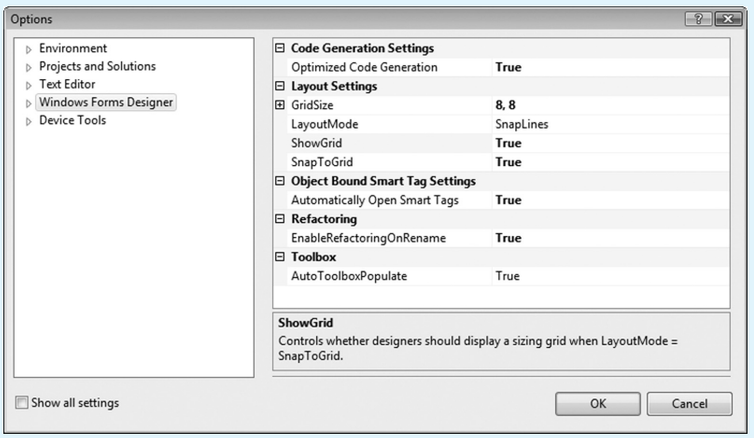
Figure 1-18 *VB Specific* text editor options



Step 12: Scroll down the left pane and select *Windows Forms Designer*. Make sure your settings match those shown in Figure 1-19. Specifically, *GridSize* should be set to *8, 8*, *ShowGrid* should be set to *True*, and *SnapToGrid* should be set to *True*. These settings control the grid you use to design forms.

TIP: To change the *GridSize* setting, click the area where the current setting is displayed, erase it, and enter `8, 8` as the new setting. To change either the *ShowGrid* or *SnapToGrid* settings, click the area where the current setting is displayed, then click the down arrow button () that appears. Select *True* from the menu that drops down.

Figure 1-19 *Windows Forms Designer* settings



NOTE: The options you set in Steps 10 through 13 will remain set until you or someone else changes them. If you are working in a shared computer lab and you find that your screens and/or the appearance of your code does not match the examples shown in this book, you will probably need to reset these options.

Step 13: Click *OK* to close the dialog box.

The Visual Studio Environment

The Visual Studio environment (for Visual Basic) consists of a number of windows and other components. Figure 1-20 shows the locations of the following components: the ***Design* window**, the ***Solution Explorer* window**, and the ***Properties* window**.

You can move the windows around, so they may not appear in the exact locations shown in Figure 1-20. You can also close the windows so they do not appear at all. If you do not see one or more of them, follow the steps in Tutorial 1-5 to make them visible.

Dynamic Help Window (Visual Studio Only)

The ***Dynamic Help* window**, as shown in Figure 1-21 gives you a list of help topics that provide excellent tutorial and reference information about Visual Basic. It is available only in Visual Studio. Display the *Dynamic Help* window by selecting *Dynamic Help* from the menu bar's *Help* menu.

Figure 1-20 The *Design* window, *Solution Explorer* window, and *Properties* window

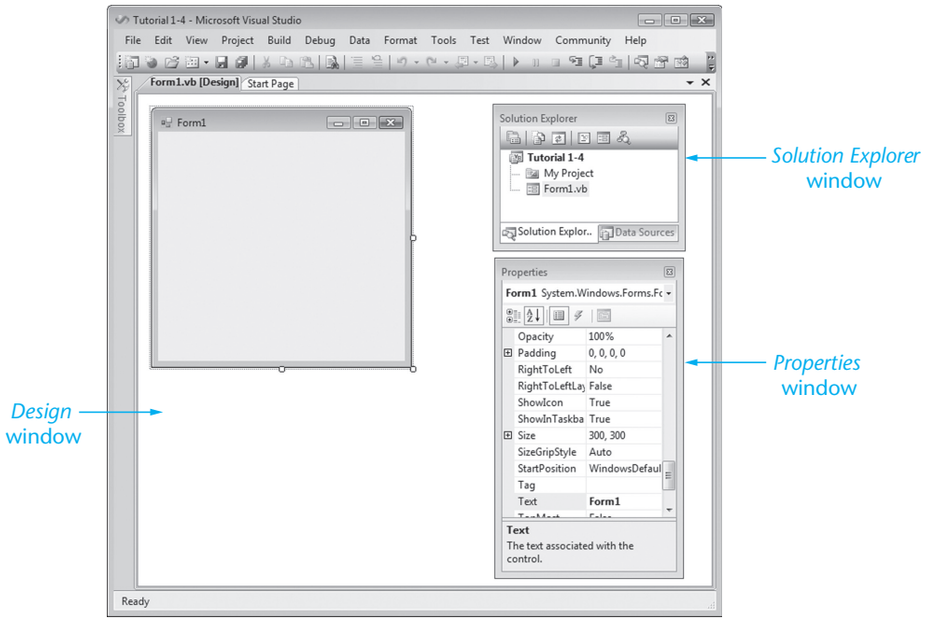


Figure 1-21 The *Dynamic Help* window

Tutorial 1-5:

Opening the *Design*, *Solution Explorer*, *Dynamic Help*, and *Properties* windows in Visual Studio

Step 1: (This tutorial is for Visual Studio users only.) If you do not see the *Design* window, click *View* on the menu bar. On the *View* menu, click *Designer*. You can also press Shift+F7 on the keyboard.

Step 2: If you do not see the *Solution Explorer* window, click *View* on the menu bar. On the *View* menu, click *Solution Explorer*. You can also press Ctrl+Alt+L on the keyboard.

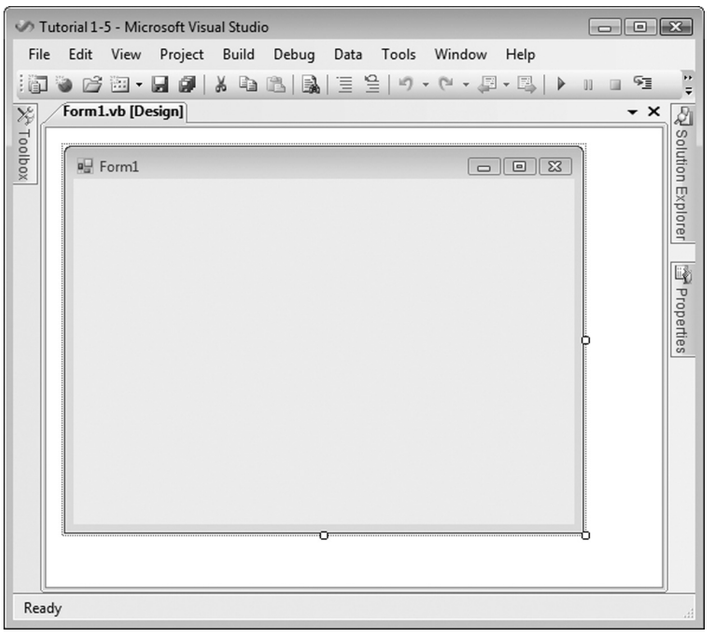
Step 3: If you do not see the *Dynamic Help* window or the *Dynamic Help* window tab, click *Help* on the menu bar. On the *Help* menu, click *Dynamic Help*.

Step 4: If you do not see the *Properties* window, click *View* on the menu bar. On the *View* menu, click *Properties*. You can also press F4 on the keyboard.

Hidden Windows

Many windows in Visual Studio have a feature known as *Auto Hide*. When *Auto Hide* is turned on, the window is displayed only as a tab along one of the edges of the *Visual Studio* window. This feature gives you more room to view your application's forms and code. Figure 1-22 shows how the *Solution Explorer* and *Properties* windows appear when their *Auto Hide* feature is turned on. Notice the tabs that read *Solution Explorer* and *Properties* along the right edge of the screen.

Figure 1-22 The *Solution Explorer* and *Properties* windows hidden



To display a hidden window, hover the mouse pointer over its tab, which pulls the window back into view. If you want the window to remain in view for a time, click its tab. The window will remain displayed until you click outside of it.

- To set a window to *Auto Hide*, right-click its caption bar and select *Auto Hide*.
- To remove the *Auto Hide* feature from a window, click its tab to display it; then right-click its caption bar and deselect *Auto Hide*.

(Alternatively, you can click the little pushpin icon on the window's caption bar to turn *Auto Hide* on and off.)

Docked and Floating Windows

Figure 1-17 shows the *Solution Explorer* and *Properties* windows when they are **docked**, which means they are attached to each other or to one of the edges of the *Visual Studio* window. Alternatively, the windows can be **floating**. You can control whether a window is **docked** (**dockable**) or floating as follows:

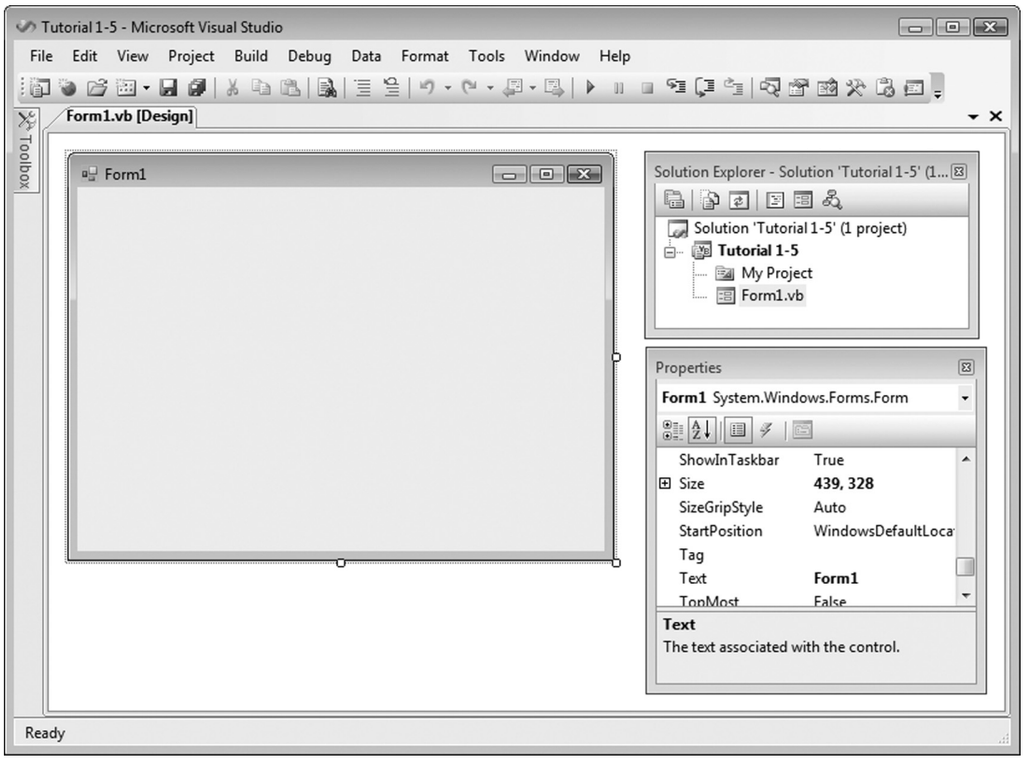
- To change a window from dockable to floating, right-click its caption bar and select *Floating*.

- To change a window from floating to dockable, right-click its caption bar and select *Dockable*.

When you click and drag one of these windows by its title bar, you move it out of the docked position, and the window becomes floating. Double-clicking the window's title bar produces the same effect. Figure 1-23 shows these windows floating.

To dock a floating window, double-click its title bar or drag it to one of the edges of the main window. You may use whichever style you prefer—docked or floating. When windows are floating, they behave as normal windows. You may move or resize them to suit your preference.

Figure 1-23 *Solution Explorer* and *Properties* windows floating



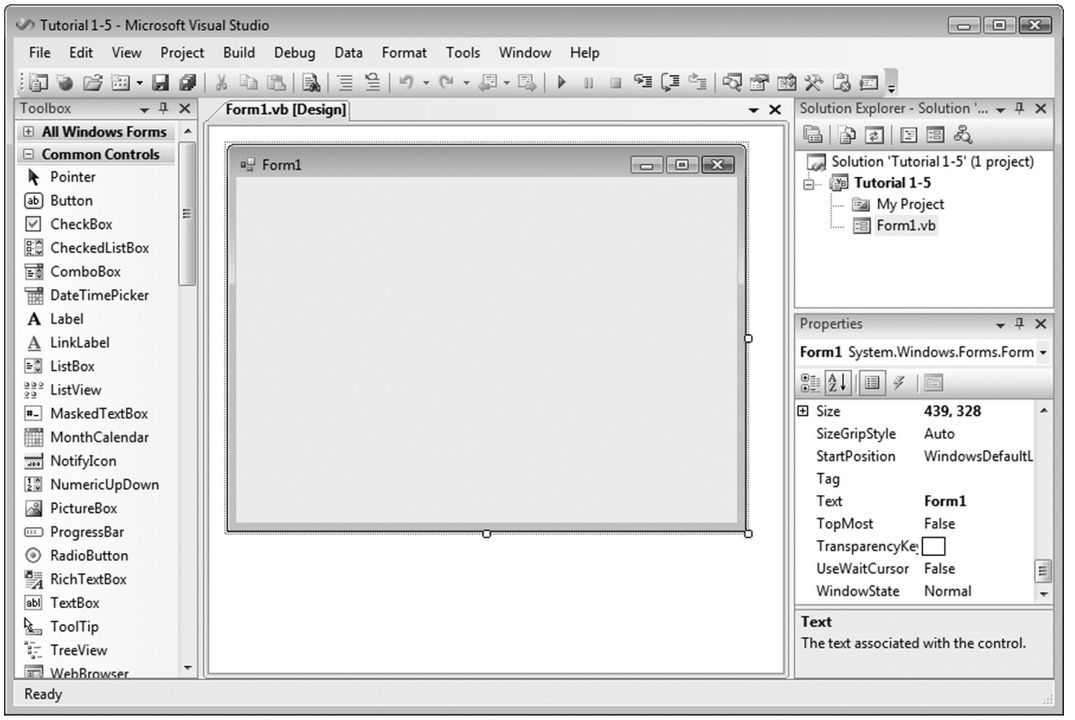
TIP: A window cannot float if its *Auto Hide* feature is turned on.

Now you have the Visual Basic environment set up properly to work with the projects in this book. Next, we will look at the individual elements.

The Title Bar

The title bar indicates the name of the project you are currently working on. The title bar shown in Figure 1-24 shows the name Tutorial 1-5. Visual Studio is currently in **Design mode**, the mode in which you design and build an application. Chapter 2 covers **Run mode** (runtime), the mode in which you run and test an application. **Break mode** is when an application is suspended for debugging purposes.

Figure 1-24 Visual Studio in Design mode



The Menu Bar

Below the title bar is the menu bar, from which you access menus when building an application.

The Standard Toolbar

Below the menu bar is the standard toolbar. The **standard toolbar** contains buttons that execute frequently used commands. All commands executed by the toolbar may also be executed from a menu, but the standard toolbar gives you quicker access to them. The standard toolbar buttons open windows with a single click. Figure 1-25 identifies the standard toolbar buttons and Table 1-9 gives a brief description of each.

Figure 1-25 Visual Basic standard toolbar buttons

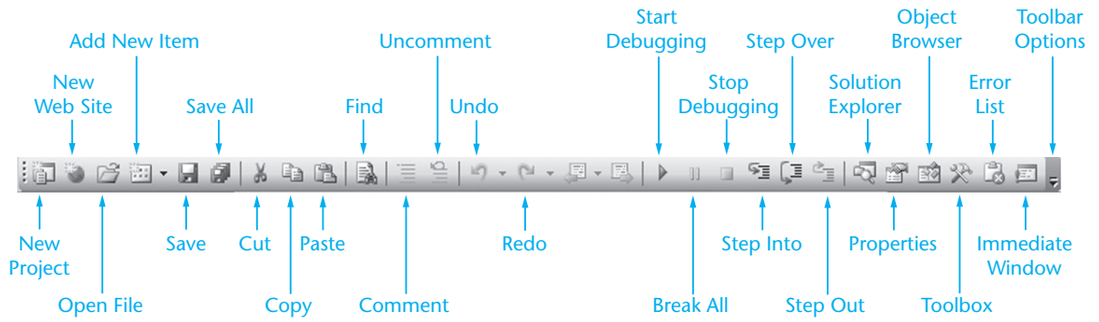


Table 1-9 Visual Basic toolbar buttons

Toolbar Button	Description
New Project	Starts a new project
New Web Site	Creates a new Web site (used in Web Forms applications)
Open File	Opens an existing file
Add New Item	Adds a new item such as a form to the current project. This button has a drop-down list (shown by the downward pointing arrow) that lets you select from several types of items.
Save *filename*	Saves the file named by *filename*
Save All	Saves all of the files in the current project
Cut	Cuts the selected item to the clipboard
Copy	Copies the selected item to the clipboard
Paste	Pastes the contents of the clipboard
Find	Searches for text in your application code
Comment	Comments out the selected lines
Uncomment	Uncomments the selected lines
Undo	Undoes the most recent operation
Redo	Redoes the most recently undone operation
Start Debugging	Starts debugging (running) your program
Break All	Pauses execution of your program
Stop Debugging	Stops debugging (running) your program
Step Into	Traces (steps) into the code in a procedure
Step Over	Executes the next statement without tracing into procedure calls
Step Out	Exits the current procedure while still debugging
Solution Explorer	Opens the *Solution Explorer* window
Properties	Opens the *Properties* window
Object Browser	Opens the *Object Browser* window
Toolbox	Opens the *Toolbox* window, displaying, for example, visual controls you can place on Windows forms
Error List	Displays a list of most recent errors generated by the Visual Basic compiler
Immediate Window	Opens the *Immediate* window, which is used for debugging
Toolbar Options	Lets you add buttons to or remove buttons from the toolbar

NOTE: As with most Windows applications, menu items and buttons cannot be used when they are grayed out.

The *Toolbox* Window

Figure 1-26 shows the ***Toolbox* window**, with all groups (tabs) closed. The *Toolbox* contains buttons and icons that let you build rich visual interfaces in Windows desktop applications. You have already seen some of the tools in tutorials in the earlier part of this chapter.

The *Toolbox* window is divided into sections, which are accessible by clicking on named tabs. Clicking a tab displays the entries in the group. Figure 1-27 shows the *Toolbox*

window after the user has clicked the *Common Controls* tab. The controls in this group include Buttons, Labels, TextBoxes, and other common controls. Not all of the *Toolbox* items can be displayed at once, so scroll arrows are provided.

TIP: The *Toolbox* window is only activated when a form is open in Design mode.

Using ToolTips

A **ToolTip** is a small rectangular box that pops up when you hover the mouse over a button on the toolbar or in the *Toolbox* for a few seconds. The box contains a short description of the button's purpose. Figure 1-28 shows the ToolTip that appears when the cursor is left sitting on the *Save All* button. Use a ToolTip whenever you cannot remember a particular button's function.

Figure 1-26 The *Toolbox* window, with all tabs closed

Figure 1-27 The *Toolbox* window, showing *Common Controls*

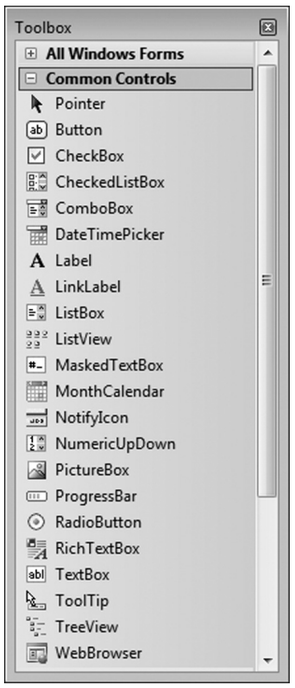


Figure 1-28 *Save All* ToolTip

Tutorial 1-6 introduces you to Visual Studio.

Tutorial 1-6:
Getting familiar with Visual Studio

If Visual Studio is not running on your computer, follow the steps in Tutorial 1-4. This exercise will give you practice working with elements of the Visual Basic environment.

Step 1: Make sure *Auto Hide* is turned off for the *Solution Explorer* and *Properties* windows. If your *Solution Explorer* and *Properties* windows are in the docked

position, double-click each of their title bars to undock them. (If they are already floating, skip to Step 2.)

Step 2: Practice moving the windows around on the screen by clicking and dragging their title bars.

Step 3: Double-click the title bars of each of the windows to move them back to their docked positions.

Step 4: The *Solution Explorer*, *Properties* window, *Dynamic Help* window, and *Toolbox* each have a *Close* button in their upper right corner. Close each of these windows by clicking their *Close* buttons (*Dynamic Help* does not appear in Visual Basic Express).

Step 5: Do you remember which buttons on the toolbar restore the *Solution Explorer*, *Properties* window, and *Toolbox*? If not, move your mouse cursor over any button on the toolbar, and leave it there until the ToolTip appears. Repeat this procedure on different buttons until you find the ones whose ToolTips read *Solution Explorer*, *Properties Window*, and *Toolbox*.

Step 6: Click the appropriate buttons on the toolbar to restore the *Solution Explorer*, *Properties*, and *Toolbox* windows.

Step 7: Exit Visual Studio by clicking *File* on the menu bar, then clicking *Exit* on the *File* menu. You may see a dialog box asking you if you wish to save changes to a number of items. Because we are just experimenting with the Visual Basic environment, click *No*.

In this section, you learned to start Visual Studio, interact with Visual Studio, and identify many on-screen tools and components. In Chapter 2, you will start building your first application.

Checkpoint

1.22 Briefly describe the purpose of the *Solution Explorer* window.

1.23 Briefly describe the purpose of the *Properties* window.

1.24 Briefly describe the purpose of the *Dynamic Help* window.

1.25 Briefly describe the purpose of the standard toolbar.

1.26 What is Design mode? What is Run mode? What is Break mode?

1.27 What is the difference between the toolbar and the *Toolbox*?

1.28 What is a ToolTip?

Summary

1.1 Computer Systems: Hardware and Software

- The major hardware components of a computer are the central processing unit (CPU), main memory, secondary storage devices, input devices, and output devices. Computer programs are stored in machine language, as a series of binary numbers.
- Main memory holds the instructions for programs that are running and data programs are working with. RAM is usually volatile, used only for temporary storage.
- The two general categories of software are operating systems and application software.

1.2 Programs and Programming Languages

- The two primary methods of programming in use today are procedural and object-oriented.
- The advent of graphical user interfaces (GUIs) has influenced the shift from procedural programming to object-oriented.
- There are several types of controls available in Visual Basic. Applications in this chapter contained forms, Labels, TextBoxes, Buttons, CheckBoxes, RadioButtons, ListBoxes, ComboBoxes, and scroll bars.
- The appearance of a screen object, such as a form or other control, is determined by the object's properties.
- An event-driven program is one that responds to events or actions that take place while the program is running.

1.3 More about Controls and Programming

- All controls have a name. Programmers manipulate or access a control in a programming statement by referring to the control by its name. When the programmer creates a control in Visual Basic, it automatically receives a default name.
- Any control whose name appears in a programming statement should have a descriptive, programmer-defined name. Although programmers have a great deal of flexibility in naming controls, they should follow some standard guidelines.
- The fundamental language elements of an event procedure or other method are keywords, programmer-defined names, operators, remarks, and syntax.

1.4 The Programming Process

- This section outlines the steps for designing and creating a Visual Basic application.

1.5 Visual Studio and Visual Basic Express (the Visual Basic Environment)

- The Visual Basic environment, which is part of Visual Studio, consists of tools used to build Visual Basic applications.
- Visual Basic can be used to create many different types of applications.

Key Terms

algorithm
application software
attributes
binary number
Break mode
button
central processing unit (CPU)
CheckBox

ComboBox
comments
compiler
controls
Design mode
Design window
disk drive
docked (dockable) windows
Dynamic Help window
event-driven
event procedure
floating window
flowchart
Form object
graphical user interface (GUI)
GroupBox
hardware
HScrollBar
identifier
input
input device
integrated development environment (IDE)
keywords
Label
language syntax
ListBox
machine language instructions
main memory
methods
Name property
New Project dialog box
object-oriented programming (OOP)
objects
operands
operating system (OS)
operators
Options dialog box
output
output device
PictureBox
procedural programming
procedure
program
programmer-defined name
programming languages
project
properties
Properties window
pseudocode
RadioButton
random-access memory (RAM)
remarks
Run mode
runtime error
secondary storage
software
Solution Explorer window
standard toolbar
syntax
syntax error
Text property
TextBox
Toolbox window
ToolTip
VScrollBar

Review Questions and Exercises

Fill-in-the-Blank

1. The job of the _____ is to fetch instructions, carry out the operations commanded by the instructions, and produce some outcome or resultant information.
2. A(n) _____ is an example of a secondary storage device.
3. The two general categories of software are _____ and _____.
4. A program is a set of _____.
5. Since computers can't be programmed in natural human language, algorithms must be written in a(n) _____ language.
6. _____ is the only language computers really process.

7. Words that have special meaning in a programming language are called _____.
8. Words or names defined by the programmer are called _____.
9. _____ are characters or symbols that perform operations on one or more operands.
10. A(n) _____ is part of an application's code but is ignored by the compiler. It is intended for documentation purposes only.
11. The rules that must be followed when constructing a program are called _____.
12. _____ is information a program gathers from the outside world.
13. _____ is information a program sends to the outside world.
14. A(n) _____ is a set of well-defined steps for performing a task or solving a problem.
15. A(n) _____ is a diagram that graphically illustrates the flow of a program.
16. _____ is a cross between human language and a programming language.
17. To set the Visual Basic environment options, click the *Options . . .* command, which is found on the _____ menu.
18. If you do not see the *Solution Explorer* or *Properties* windows in Visual Studio, you may use the _____ menu to bring them up.
19. A(n) _____ is a container for holding a project.
20. A(n) _____ is a group of files that make up a software application.
21. The _____ window allows you to navigate among the files in your project.
22. The _____ window shows most of the currently selected object's properties and those properties' values.
23. When windows are _____, it means they are attached to each other or to one of the edges of the Visual Studio main window.
24. To dock a floating window, _____ its title bar or drag it to one of the edges of the main window.
25. Visual Studio's _____ window indicates the name of the project you are working on while you are in Design mode.
26. All commands executed by the _____ may also be executed from a menu.
27. The _____ window contains your application's form. This is where you design your application's user interface by placing controls on the form that appears when your application executes.
28. You use the _____ to place controls on an application's form. It contains buttons for the commonly used Visual Basic controls.

29. The _____ window displays help topics that are relevant to the operation you are currently performing in Visual Basic.

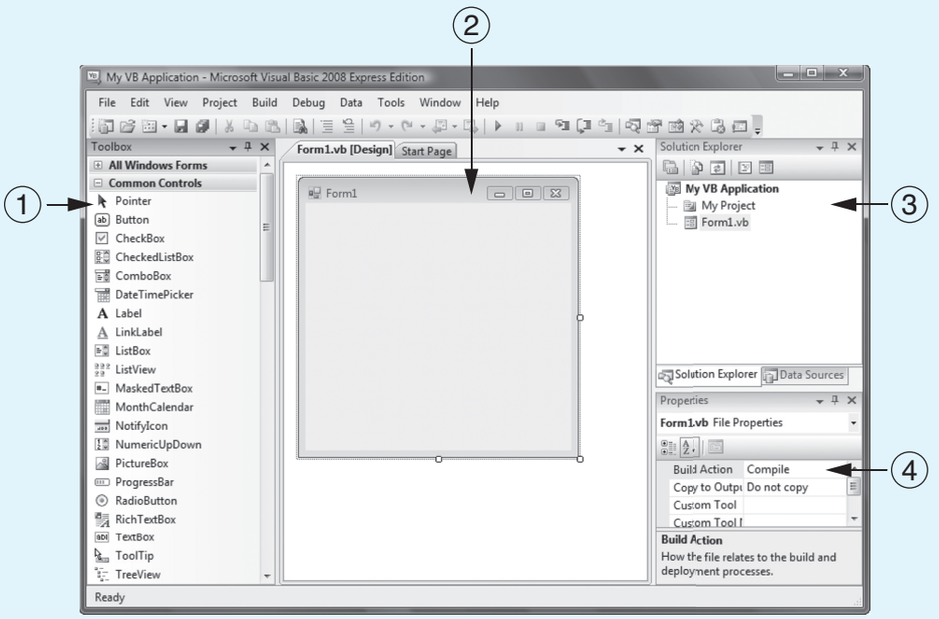
30. A(n) _____ is a small box that is displayed when you hold the mouse cursor over a button on the toolbar or in the *Toolbox* for a few seconds.

Short Answer

1. What is the difference between main memory and secondary storage?
2. What is the difference between operating system software and application software?
3. Briefly describe what procedural programming means.
4. Briefly describe what object-oriented programming means.
5. Briefly describe what an event-driven program is.
6. Why has the advent of graphical user interfaces (GUIs) influenced the shift from procedural programming to object-oriented/event-driven programming?
7. From what you have read in this chapter, describe the difference between a Label control and a TextBox control. When is it appropriate to use one or the other?
8. When creating a VB application, you will spend much of your time doing what three things?
9. What is a form?
10. Summarize the mandatory rules that you must follow when naming a control.
11. What is a keyword?
12. What is the purpose of inserting comments in a program?
13. What is language syntax?
14. What is a syntax error?
15. What is a runtime error?
16. What is an operator?
17. What is a flowchart?
18. What is pseudocode?
19. What default name will VB give to the first Label control that you place on a form? What default name will VB assign to the first TextBox control that you place on a form?
20. What property determines the text that is displayed by a Label control?
21. What is *Auto Hide*? How do you turn *Auto Hide* on or off?
22. What is the *Toolbox* window in Visual Studio?
23. What is the standard toolbar in Visual Studio?
24. What is a tooltip?
25. If you do not see the *Solution Explorer* window in Visual Studio, how do you display it?
26. If you do not see the *Properties* window in Visual Studio, how do you display it?

27. How do you display the *Dynamic Help* window in Visual Studio?
28. What mode is Visual Basic in while you are designing and building an application?
29. What mode is Visual Basic in while you are running an application?
30. What mode is Visual Basic in while an application is suspended for debugging?
31. Figure 1-29 shows the Visual Basic IDE. What are the names of the four areas that are indicated in the figure?

Figure 1-29 The Visual Basic IDE



What Do You Think?

1. Are each of the following control names legal or illegal? If a name is illegal, indicate why.
   a. `txtUserName`
   b. `2001sales`
   c. `lblUser Age`
   d. `txtName/Address`
   e. `btnCalcSubtotal`
2. What type of control does each of the following prefixes usually indicate?
   a. `btn`
   b. `lbl`
   c. `txt`
3. For each of the following controls, make up a legal name that conforms to the standard control name convention described in this chapter.
   a. A TextBox control in which the user enters his or her last name
   b. A Button control that, when clicked, calculates an annual interest rate
   c. A Label control used to display the total of an order
   d. A Button control that clears all the input fields on a form

4. The following control names appear in a Visual Basic application used in a retail store. Indicate what type of control each is and guess its purpose.

   a. `txtPriceEach`
   b. `txtQuantity`
   c. `txtTaxRate`
   d. `btnCalcSale`
   e. `lblSubTotal`
   f. `lblTotal`

Programming Challenges

1. **Carpet Size**

   You have been asked to create an application for a carpet sales and installation business. The application should allow the user to enter the length and width of a room and calculate the room's area in square feet. The formula for this calculation is

   $$Area = Length \times Width$$

   In this exercise, you will gain practice using Steps 1 through 6 of the programming process described in Section 1.5:

   1. Clearly define what the application is to do.
   2. Visualize the application running on the computer and design its user interface.
   3. Make a list of the controls needed.
   4. Define the values of each control's properties.
   5. Make a list of methods needed for each control.
   6. Create a flowchart or pseudocode version of each method.

   Step 1: Describe the following characteristics of this application:

   Purpose
   Input
   Process
   Output

   Step 2: Draw a sketch of the application's form and place all the controls that are needed.

   Step 3: Make a list of the controls you included in your sketch. List the control type and the name of each control.

   Step 4: List the value of the Text property for each control, as needed. (Remember, some controls do not have a Text property.)

   Step 5: List each method needed. Give the name of each method and describe what each method does.

   Step 6: For each method you listed in Step 5, draw a flowchart or write pseudocode.

2. **Available Credit**

   A retail store gives each of its customers a maximum amount of credit. A customer's available credit is determined by subtracting the amount of credit used by the customer from the customer's maximum amount of credit. As you did in Programming Challenge 1, perform Steps 1 through 6 of the programming process to design an application that determines a customer's available credit.

VideoNote
Solving the Sales Tax Problem

3. **Sales Tax**

   Perform Steps 1 through 6 of the programming process to design an application that gets from the user the amount of a retail sale and the sales tax rate. The application should calculate the amount of the sales tax and the total of the sale.

4. Account Balance

Perform Steps 1 through 6 of the programming process to design an application that gets from the user the starting balance of a savings account, the total dollar amount of the deposits made to the account, and the total dollar amount of withdrawals made from the account. The application should calculate the account balance.