2

Introduction to the Visual Studio .NET IDE

Objectives

- To be introduced to the Visual Studio .NET Integrated Development Environment (IDE).
- To become familiar with the types of commands contained in the IDE's menus and toolbars.
- To understand the use of various kinds of windows in the Visual Studio .NET IDE.
- To understand Visual Studio .NET's help features.
- To be able to create, compile and execute a simple Visual Basic program.

Seeing is believing.
Proverb

Form ever follows function.
Louis Henri Sullivan

Intelligence… is the faculty of making artificial objects, especially tools to make tools.
Henri-Louis Bergson

Outline

2.1 Introduction

Visual Studio .NET is Microsoft's Integrated Development Environment (IDE) for creating, running and debugging programs (also called *applications*) written in a variety of .NET programming languages. This IDE is a powerful and sophisticated tool that is used to create business-critical and mission-critical applications. In this chapter, we provide an overview of the Visual Studio .NET IDE and demonstrate how to create a simple Visual Basic program by dragging and dropping predefined building blocks into place—a technique called "visual programming." We introduce additional features of the IDE and discuss the more advanced "visual programming" techniques throughout the book.

2.2 Overview of the Visual Studio .NET IDE

When Visual Studio .NET is executed, the ***Start Page*** is displayed (Fig. 2.1). The left-hand side of the **Start Page** contains a list of helpful links, such as **Get Started**. Clicking a link displays its contents. We refer to single-clicking with the left mouse button as *selecting*, or *clicking*, whereas we refer to double-clicking with the left mouse button as *double-clicking*. [*Note*: Your Start Page may be slightly different depending on your version of Visual Studio.]

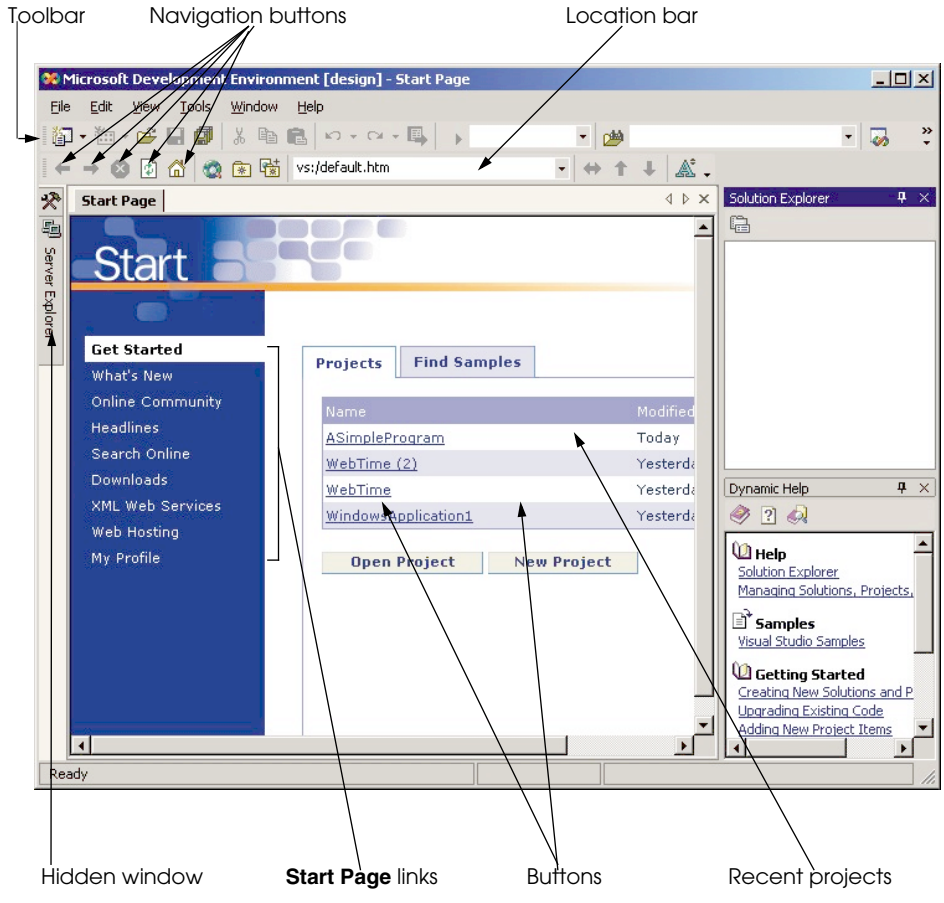


Fig. 2.1 **Start Page** in Visual Studio .NET.

When clicked, ***Get Started*** loads a page that contains a table of the names of recent *project* (such as **ASimpleProgram** in Fig. 2.1), along with the dates on which these projects were last modified. A project is a group of related files, such as the Visual Basic code and images that make up a program. When you load Visual Studio .NET for the first time, the list of recent projects will be empty. There are two *buttons* on the page—***Open Project*** and ***New Project***, which are used to open an existing project (such as the ones in the table of recent projects) or to create a new project, respectively. We discuss the process of creating new projects momentarily.

Other links on the **Start Page** offer information and resources related to Visual Studio .NET. Clicking ***What's New*** displays a page that lists new features and updates for Visual Studio .NET, including downloads for code samples and programming tools. ***Online Community*** links to on-line resources for contacting other software developers through *newsgroups* (organized message boards on the Internet) and Web sites. ***Headlines*** pro-

vides a page for browsing news, articles and how-to guides. To access more extensive information, users can select ***Search Online*** and begin browsing through the *MSDN* (*Microsoft Developer Network*) on-line library, which contains numerous articles, downloads and tutorials on various technologies of interest to Visual Studio .NET users. When clicked, ***Downloads*** displays a page that provides programmers access to product updates, code samples and reference materials. The ***Web Hosting*** page allows programmers to post their software (such as Web services, which we discuss in Chapter 21, and ASP.NET) on-line for public use. The ***My Profile*** link loads a page where users can adjust and customize various Visual Studio .NET settings, such as keyboard schemes and window layout preferences. (The programmer can also access **Tools > Options...** and **Tools > Customize...** to customize the Visual Studio .NET IDE.) [*Note:* The **Tools > Options...** notation indicates that **Options...** is a command in the **Tools** menu.]

Programmers can browse the Web from the IDE using Internet Explorer (also called the internal Web browser in Visual Studio). To request a Web page, type its address into the location bar (Fig. 2.1) and press the *Enter* key. [*Note*: The computer must, of course, be connected to the Internet.] Several other windows appear in the IDE besides the **Start Page**; we discuss them in subsequent sections.

To create a new Visual Basic program, click the **New Project** button (Fig. 2.1), which displays the **New Project** *dialog* (Fig. 2.2). Dialogs are windows that facilitate user–computer communication.

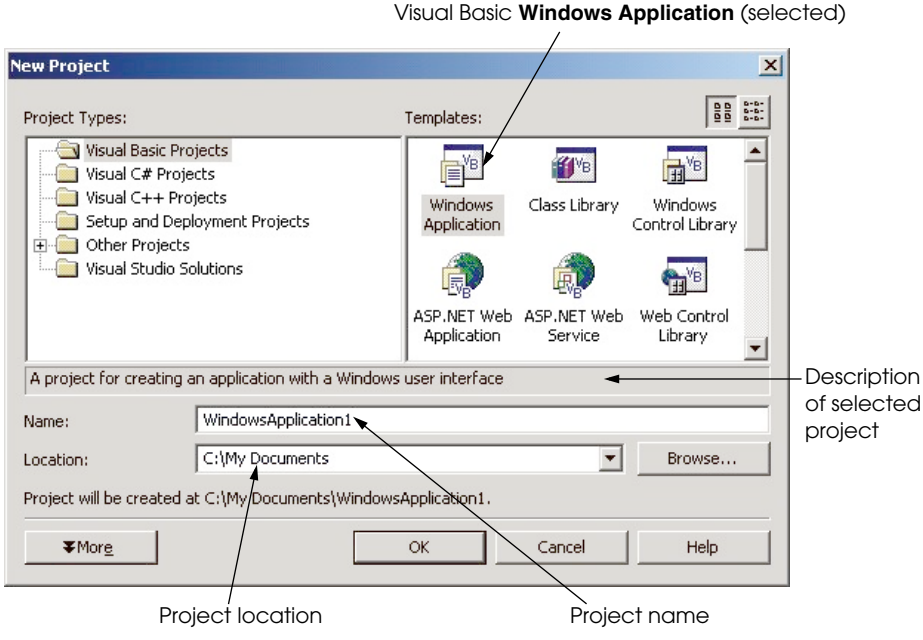


Fig. 2.2 **New Project** dialog.

The Visual Studio .NET IDE organizes programs into *projects* and *solutions*. Projects are groups of related files that form a Visual Basic program; solutions contain one or more

projects. Multiple-project solutions are used to create large-scale applications in which each project performs a single, well-defined task.

The Visual Studio .NET IDE provides project types for a variety of programming languages. This book focuses on Visual Basic, so select the ***Visual Basic Projects*** folder from the ***Project Types*** *window* (Fig. 2.2). We use several Visual Basic project types in this book. A ***Windows Application***, which is a program that executes inside the Windows OS (e.g., Windows 2000 or Windows XP). Such programs include customized software that you create as well as software products like Microsoft Word, Internet Explorer and Visual Studio .NET.

By default, the Visual Studio .NET IDE assigns the name **WindowsApplication1** to the new project and solution (Fig. 2.2). The **Visual Studio Projects** folder in the **My Documents** folder is the default folder referenced when Visual Studio .NET is executed for the first time. Programmers can change both the name and the location where projects are created. After selecting a name and location for the project, click **OK** to display the IDE in *design view* (Fig. 2.3), which contains all the features necessary to begin creating Visual Basic programs.

Good Programming Practice 2.1

Developers should change the name and location of each project to describe the program's functionality.

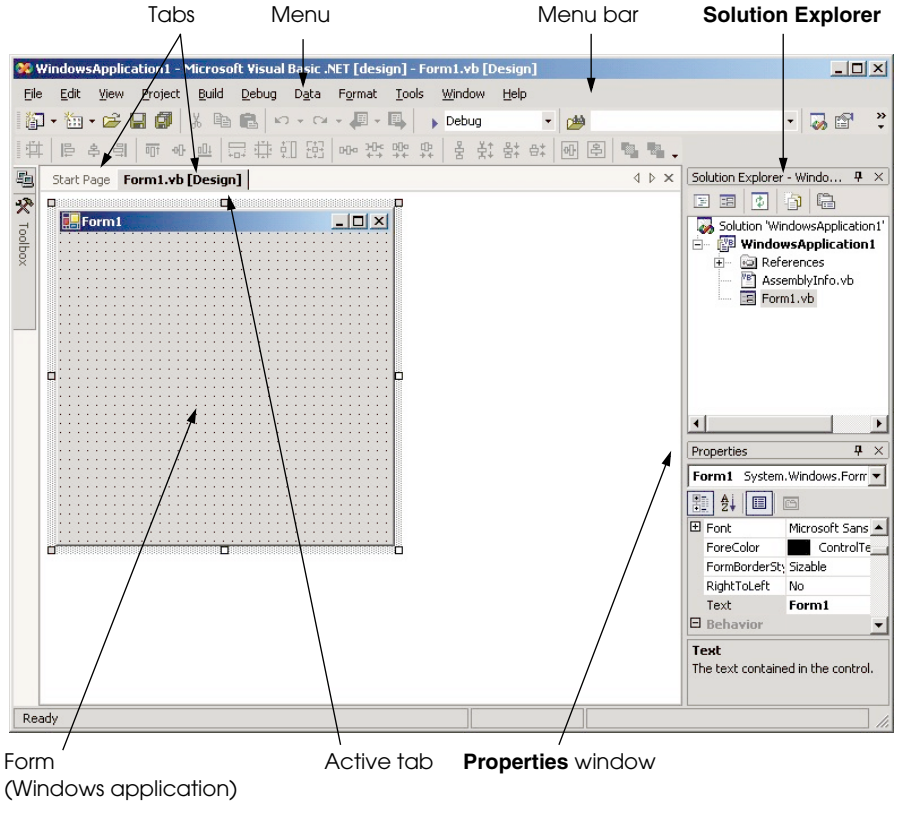


Fig. 2.3 **Design** view of Visual Studio .NET IDE.

The gray rectangle (called a *form*) titled **Form1** represents the Windows application that the programmer is creating. Later in this chapter, we discuss how to customize this form by adding *controls* (i.e., reusable components, such as buttons). Collectively, the form and controls constitute the program's *Graphical User Interface* (*GUI*), which is the visual part of the program with which the user interacts. Users enter data (*inputs*) into the program by typing at the keyboard, by clicking the mouse buttons and in a variety of other ways. Programs display instructions and other information (*outputs*) for users to read in the GUI. For example, the **New Project** dialog in Fig. 2.2 presents a GUI where the user clicks with the mouse button to select a project type and types a project name and location from the keyboard.

The name of each open document is listed on a tab. In our case, the documents are the **Start Page** and **Form1.vb [Design]** (upper left portion of Fig. 2.3). To view a document, click its tab. Tabs save space and facilitate easy access to multiple documents. The *active tab*, or the tab of the document currently displayed in the IDE, is displayed in bold text (e.g., **Form1.vb [Design]**) and is positioned in front of all the other tabs.

2.3 Menu Bar and Toolbar

Commands for managing the IDE and for developing, maintaining and executing programs are contained in the menus, which are located on the menu bar (Fig. 2.4). Menus contain groups of related commands (also called *menu items*) that, when selected, cause the IDE to perform specific actions (e.g., open a window, save a file, print a file and execute a program). For example, new projects are created by selecting **File > New > Project...**. The menus depicted in Fig. 2.4 are summarized in Fig. 2.5. In Chapter 13, Graphical User Interfaces: Part 2, we discuss how programmers can create and add their own menus and menu items to their projects.

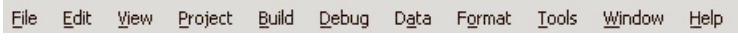


Fig. 2.4 Visual Studio .NET IDE menu bar.

Menu	Description
File	Contains commands for opening projects, closing projects, printing project data, etc.
Edit	Contains commands such as cut, paste, find, undo, etc.
View	Contains commands for displaying IDE windows and toolbars.
Project	Contains commands for managing a project and its files.
Build	Contains commands for compiling a program.
Debug	Contains commands for *debugging* (i.e., identifying and correcting problems in a program) and running a program.
Data	Contains commands for interacting with *databases* (i.e., files that store data, which we discuss in Chapter 19, Databases, SQL and ADO .NET).
Format	Contains commands for arranging and changing the appearance of a form's controls.
Tools	Contains commands for accessing additional IDE tools and options that enable customization of the IDE.
Windows	Contains commands for arranging and displaying windows.
Help	Contains commands for accessing the IDE's help features.

Fig. 2.5 Summary of Visual Studio .NET IDE menus .

Rather than having to navigate the menus for certain commonly used commands, the programmer can access them from the *toolbar* (Fig. 2.6), which contains pictures, called *icons*, that graphically represent commands. To execute a command via the toolbar, click its icon. Some icons contain a down arrow that, when clicked, displays additional options.

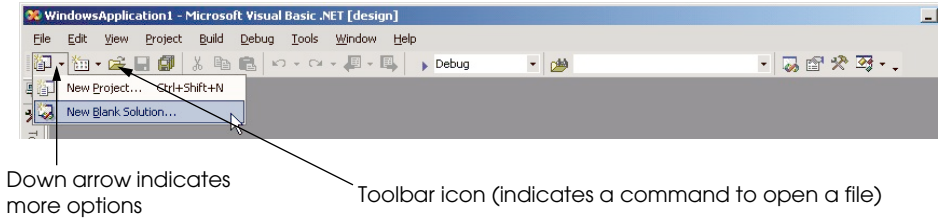


Fig. 2.6 Toolbar demonstration.

Positioning the mouse pointer over an icon highlights the icon and, after a few seconds, displays a description called a *tool tip* (Fig. 2.7). Tool tips help novice programmers become familiar with the IDE's features.

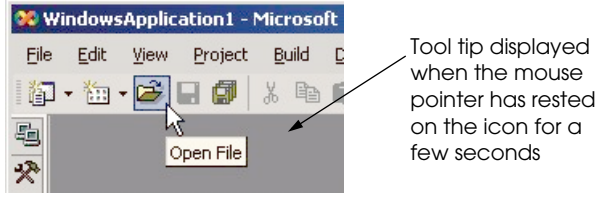


Fig. 2.7 Tool tip demonstration.

2.4 Visual Studio .NET IDE Windows

The IDE provides windows for accessing project files and customizing controls. In this section, we introduce several windows that are essential in the development of Visual Basic applications. These windows can be accessed via the toolbar icons (Fig. 2.8) or by selecting the name of the desired window in the **View** menu.

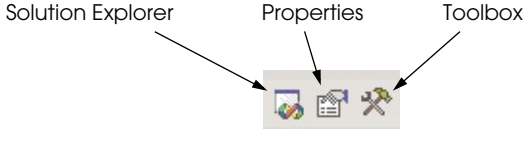


Fig. 2.8 Toolbar icons for three Visual Studio .NET IDE windows.

Visual Studio .NET provides a space-saving feature called *auto-hide*, which can be activated by clicking the *pin icon* in the upper right corner of a window (Fig. 2.9). When auto-hide is enabled, a toolbar appears along one of the edges of the IDE. This toolbar contains one or more icons, each of which identifies a hidden window. Placing the mouse pointer over one of these icons displays that window, but the window is hidden once the

mouse pointer is moved outside the window's area. To "pin down" a window (i.e., to disable auto-hide and keep the window open), click the pin icon. Notice that when a window is "pinned down," the pin icon has a vertical orientation, whereas when auto-hide is enabled, the pin icon has a horizontal orientation (Fig. 2.9).

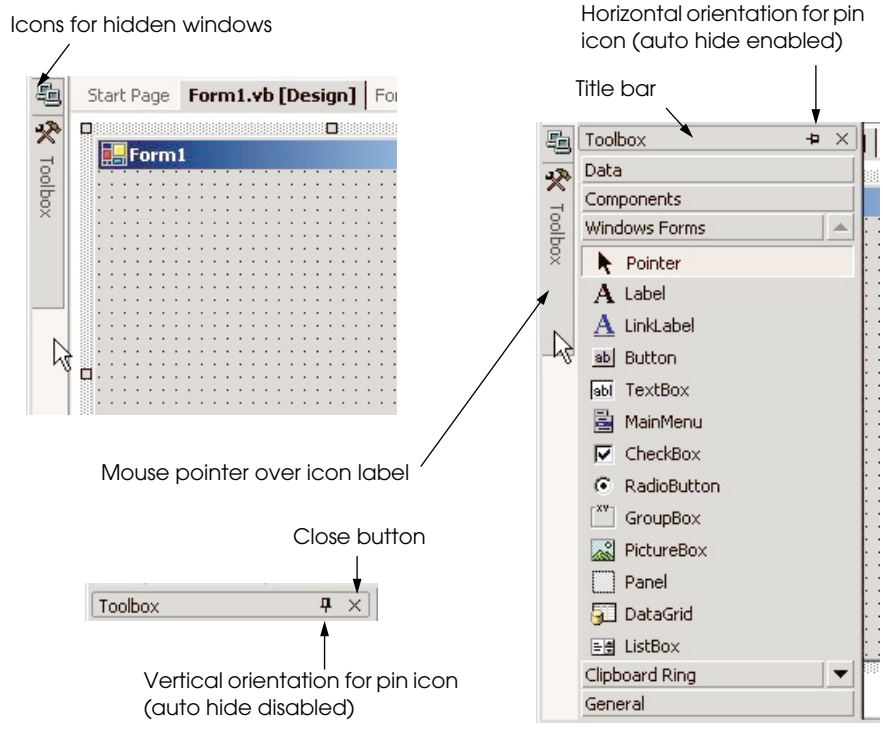


Fig. 2.9 Demonstrating the auto-hide feature.

2.4.1 Solution Explorer

The ***Solution Explorer*** window (Fig. 2.10) provides access to all the files in the solution. When the Visual Studio .NET IDE is first loaded, the **Solution Explorer** is empty; there

are no files to display. Once a solution is open, the **Solution Explorer** displays that solution's contents.

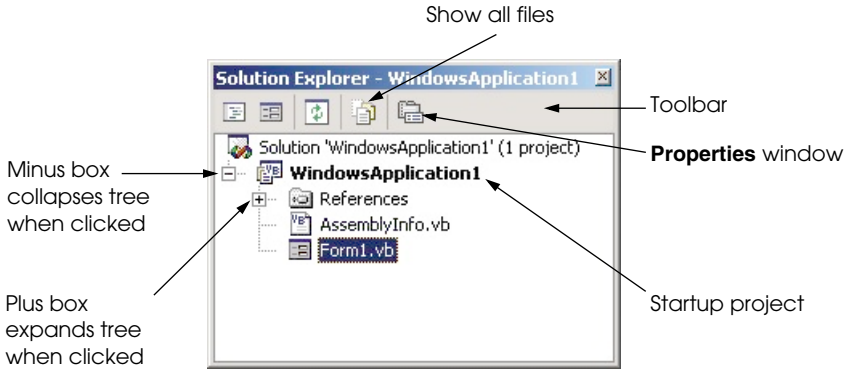


Fig. 2.10 Solution Explorer with an open solution.

The solution's *startup project* is the project that runs when the program is executed and appears in bold text in the **Solution Explorer**. For our single-project solution, the startup project is the only project (**WindowsApplication1**). The Visual Basic file, which corresponds to the form shown in Fig. 2.3, is named **`Form1.vb`**. (Visual Basic files use the **`.vb`** filename extension, which is short for "Visual Basic.") The other files and folders are discussed later in the book.

[*Note*: We use fonts to distinguish between IDE features (such as menu names and menu items) and other elements that appear in the IDE. Our convention is to emphasize IDE features in a **sans-serif bold helvetica** font and to emphasize other elements, such as filenames (e.g., **`Form1.vb`**) and property names (discussed in Section 2.4.3), in a **`serif bold courier`** font.]

The plus and minus boxes to the left of the name of the project and the **References** folder expand and collapse the tree, respectively. Click a plus box to display items grouped under the heading to the right of the plus box; click the minus box to collapse a tree already in its expanded state. Other Visual Studio .NET windows also use this plus-/minus-box convention.

The **Solution Explorer** window includes a toolbar that contains several icons. When clicked, the *show all files icon* displays all the files in the solution. The number of icons present in the toolbar is dependent on the type of file selected. We discuss additional toolbar icons later in the book.

2.4.2 Toolbox

The **Toolbox** (Fig. 2.11) contains controls used to customize forms. Using *visual programming*, programmers can "drag and drop" controls onto the form instead of writing code to build them. Just as people do not need to know how to build an engine in order to drive a car, programmers do not need to know how to build a control in order to use the control. The use of preexisting controls enables developers to concentrate on the big picture, rather than the minute and complex details of every control. The wide variety of con-

trols that are contained in the **Toolbox** is a powerful feature of the Visual Studio .NET IDE. We will use the **Toolbox** when we create our own program later in the chapter.

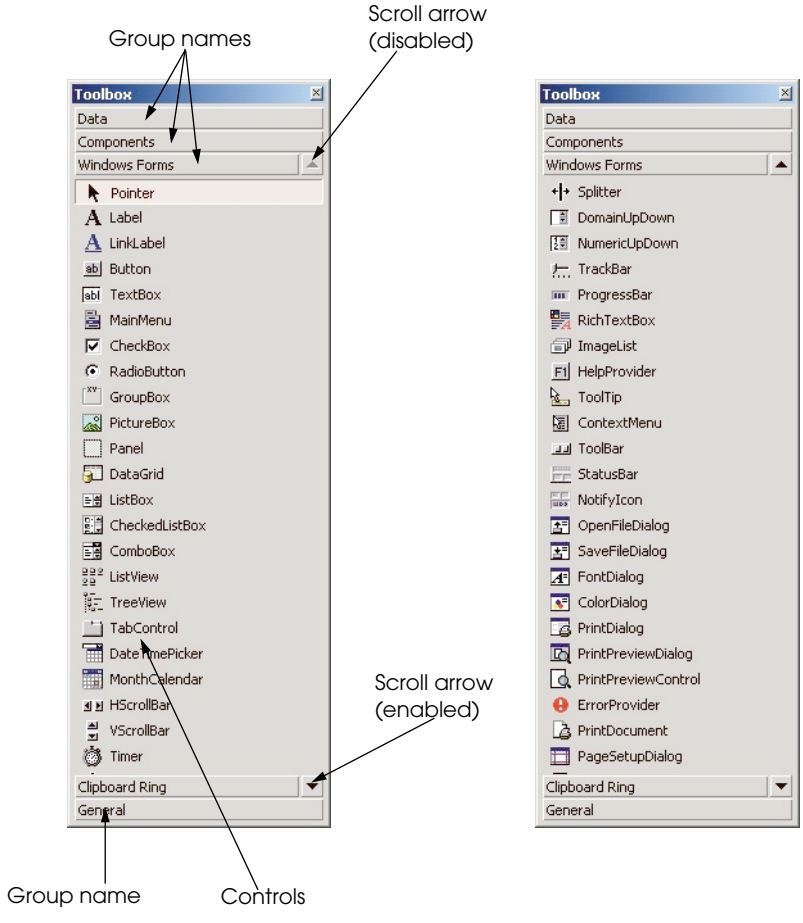


Fig. 2.11 **Toolbox** window.

The **Toolbox** contains groups of related controls (e.g., **Data**, **Components** in Fig. 2.11). When the name of a group is clicked, the list expands to display the various controls contained in the group. Users can scroll through the individual items by using the black *scroll arrows* to the right of the name of the group. If there are no more members in a group to reveal by scrolling, the scroll arrow appears in gray, meaning that it is *disabled*, i.e., it will not perform its normal function if clicked. The first item in the group is not a control—it is the *mouse pointer*. The mouse pointer is used to navigate the IDE and to manipulate a form and its controls. In later chapters, we discuss many of the **Toolbox**'s controls.

2.4.3 Properties Window

The **Properties** window (Fig. 2.12) displays the *properties* for a form or control. Properties specify information such as size, color and position. Each form or control has its own set of properties; a property's description is displayed at the bottom of the **Properties** window whenever that property is selected. If the **Properties** window is not visible, accessing **View > Properties Window**, or pressing *F4*, displays the **Properties** window.

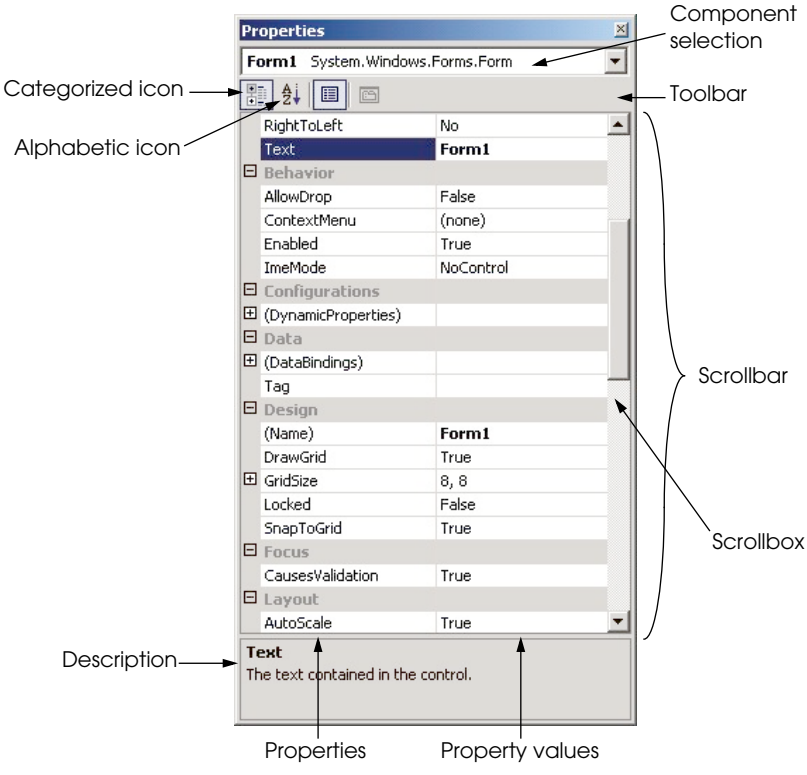


Fig. 2.12 Properties window.

In Fig. 2.12, the form's **Properties** window is shown. The left column of the **Properties** window lists the form's properties; the right column displays the current value of each property. Icons on the toolbar sort the properties either alphabetically (by clicking the ***Alphabetic*** *icon*) or categorically (by clicking the ***Categorized*** *icon*). Users can scroll through the list of properties by dragging the scrollbar's scrollbox up or down. We show how to set individual properties later in this chapter and throughout the book.

The **Properties** window is crucial to visual programming; it allows programmers to modify controls visually, without writing code. This capability provides a number of benefits. First, programmers can see which properties are available for modification and, in many cases, can learn the range of acceptable values for a given property. Second, the pro-

grammer does not have to remember or search the Visual Studio .NET documentation for the possible settings of a particular property. Third, this window also displays a brief description of the selected property, helping programmers understand the property's purpose. Fourth, a property can be set quickly using this window—usually, only a single click is required, and no code needs to be written. All of these features are designed to help programmers avoid repetitive tasks while ensuring that settings are correct and consistent throughout the project.

At the top of the **Properties** window is the *component selection* drop-down list, which allows programmers to select the form or control whose properties are displayed in the **Properties** window. When a form or control in the list is clicked, the properties of that form or control appear in the **Properties** window.

2.5 Using Help

The Visual Studio .NET IDE provides extensive help features. The ***Help*** *menu* contains a variety of commands, which are summarized in Fig. 2.13.

Command	Description
Contents	Displays a categorized table of contents in which help articles are organized by topic.
Index	Displays an alphabetized list of topics through which the programmer can browse.
Search	Allows programmers to find help articles based on search keywords.

Fig. 2.13 **Help** menu commands.

Dynamic help (Fig. 2.14) is an excellent way to get information about the IDE and its features, as it provides a list of articles based on the current content (i.e., the items around the location of the mouse cursor). To open the ***Dynamic Help*** window (if it is not already open), select **Help > Dynamic Help**. Then when you click a word or component (such as a form or a control) in the IDE, links to relevant help articles appear in the **Dynamic Help** window. The window lists relevant help topics, samples and "Getting Started" information. There is also a toolbar that provides access to the **Contents**, **Index** and **Search** help features.

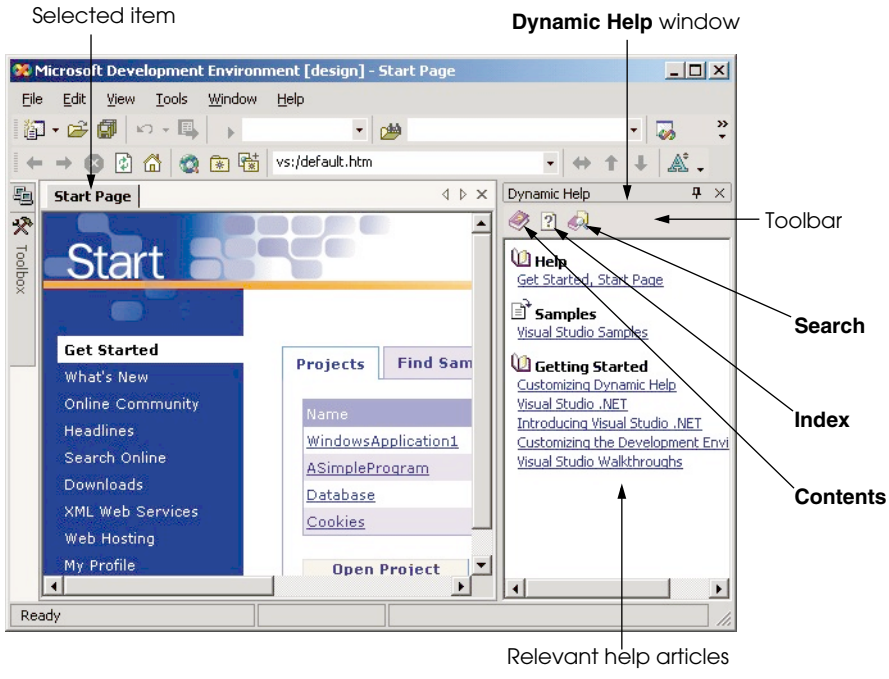


Fig. 2.14 Dynamic Help window.

Visual Studio .NET also provides *context-sensitive help*, which is similar to dynamic help, except that it immediately displays a relevant help article, rather than presenting a list of articles. To use context-sensitive help, click an item and press *F1*. Help can appear either *internally* or *externally*. When external help is selected, a relevant article immediately pops up in a separate window outside the IDE. When internal help is selected, a help article appears as a tabbed window inside the IDE. The help options can be set from the **My Profile** section of the **Start Page**.

2.6 Simple Program: Displaying Text and an Image

In this section, we create a program that displays the text "**`Welcome to Visual Basic!`**" and an image of the Deitel & Associates bug mascot. The program consists of a single form that uses a *label control* (i.e., a control that displays text which the user cannot modify) to display the text, and uses a picture box to display the image. Figure 2.15 shows the program as it executes. The example here (as well as the image file used in the example)

is available on our Web site (**www.deitel.com**) under the **Downloads/Resources** link.

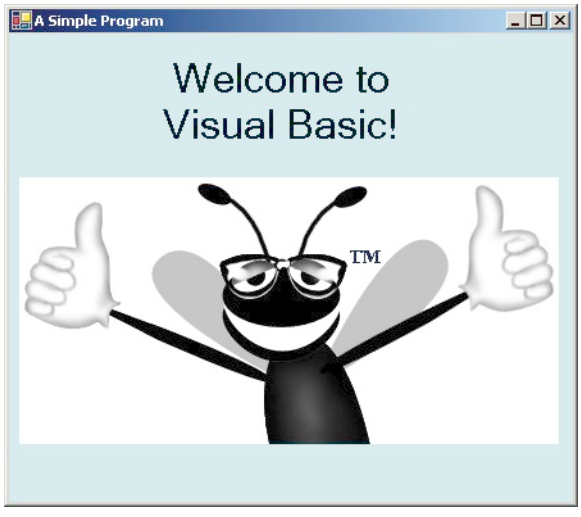


Fig. 2.15 Simple program executing.

To create the program whose outputs are shown in Fig. 2.15, we did not write a single line of program code. Instead, we use the techniques of visual programming. Various programmer *gestures* (such as using the mouse for pointing, clicking, dragging and dropping) provide Visual Studio .NET with the information necessary to generate all of this simple program's code. In the next chapter, we begin our discussion of how to write program code. Throughout the book, we produce increasingly substantial and powerful programs. Visual Basic programs usually include a combination of code written by a programmer and code generated by Visual Studio .NET.

Visual programming is useful for building GUI-intensive programs that require a large amount of user interaction. Some programs are designed not to interact with users and therefore do not have GUIs. Programmers must write the code for the latter type of program directly.

To create, run and terminate this first program, perform the following steps:

1. *Create the new project*. If a project is already open, close it by selecting **File > Close Solution**. A dialog asking whether to save the current solution may appear. To keep the unsaved changes, save the solution. Saving a solution saves all files that are part of that solution, including projects and forms. To create a new Windows application for our program, select **File > New > Project...** to display the **New Project** dialog (Fig. 2.16). Click the **Visual Basic Projects** folder to display a list of project types. From this list, select **Windows Application**. Name the project **ASimpleProgram**, and select the directory to which the project will be saved. To select a directory, click the **Browse...** button, which opens a **Project Location** dialog (Fig. 2.17). Navigate through the directories, find one in which to place the project and click **OK** to close the dialog. The select-

ed folder will now appear in the **Location** text box. Click **OK** to close the **New Project** dialog. The IDE will then load the new single-project solution, which contains a form named **Form1**.

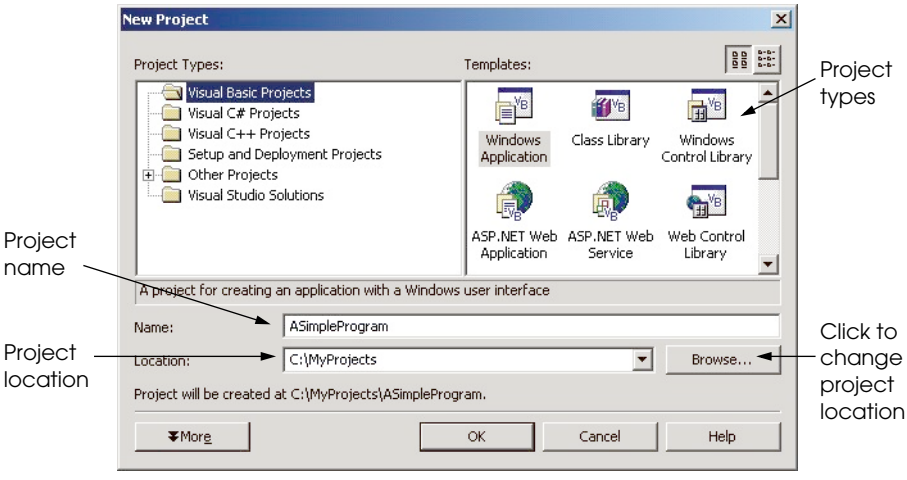


Fig. 2.16 Creating a new **Windows Application**.

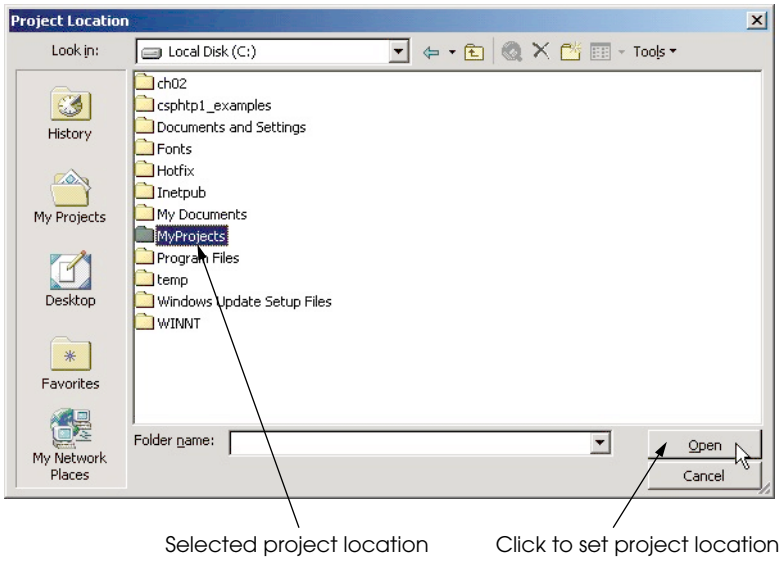


Fig. 2.17 Setting the project location in the **Project Location** dialog.

2. *Set the form's title bar*. The text in the form's title bar is determined by the form's ***Text*** property (Fig. 2.18). If the form's **Properties** window is not open, click

the properties icon in the toolbar, or select **View > Properties Window**. Click the form to display the form's properties in the **Properties** window. Click in the textbox to the right of the **`Text`** property's box, and type **A Simple Program**, as in Fig. 2.18. Press the *Enter* key (*Return* key) when finished; the form's title bar will be updated immediately.

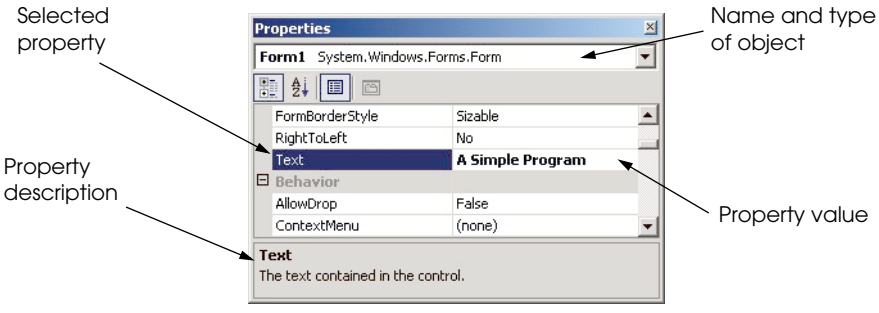


Fig. 2.18 Setting the form's **`Text`** property.

3. *Resize the form*. Click and drag one of the form's *enabled sizing handles* (the small white squares that appear around the form shown in Fig. 2.19). The mouse pointer changes its appearance (i.e., it changes to a pointer with one or more arrows) when it is over an enabled sizing handle. The new pointer indicates the direction(s) in which resizing is permitted. *Disabled sizing handles* appear in gray and cannot be used to resize the form. The *grid* on the background of the form is used by programmers to align controls and is not present when the program is running.

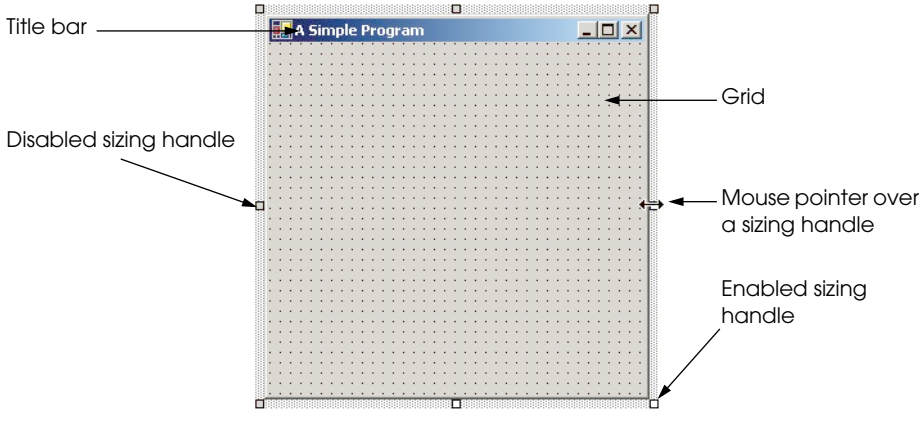


Fig. 2.19 Form with sizing handles.

4. *Change the form's background color*. The **`BackColor`** property specifies a form's or control's background color. Clicking **`BackColor`** in the **Properties** window causes a down-arrow button to appear next to the value of the property

(Fig. 2.20). When clicked, the down-arrow button displays a set of other options, which varies depending on the property. In this case, the arrow displays tabs for ***System*** (the default), ***Web*** and ***Custom***. Click the **Custom** tab to display the *palette* (a series of colors). Select the box that represents light blue. Once you select the color, the palette will close, and the form's background color will change to light blue (Fig. 2.21).

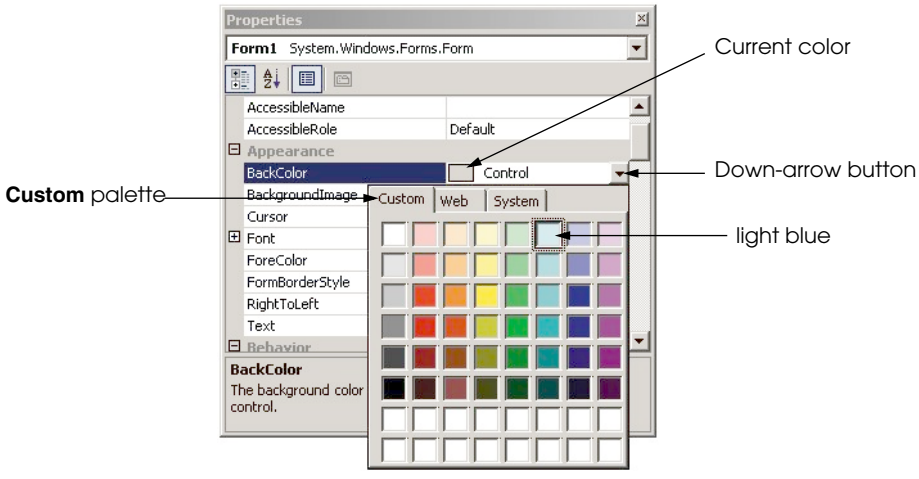


Fig. 2.20 Changing the form's **`BackColor`** property.

5. *Add a label control to the form*. Click the **Windows Forms** button in the **Toolbox**. Next, double-click the **Label** control in the **Toolbox**. This action causes a label with sizing handles to appear in the upper left corner of the form. (Fig. 2.21). Although double-clicking any **Toolbox** control places the control on the form, programmers also can "drag" controls from the **Toolbox** to the form. Labels are used to display text; our label displays the text **Label1** by default. Notice that our label's background color is the same as the form's background color. When a con-

trol is added to the form, its **`BackColor`** property is set to the form's **`BackColor`**.

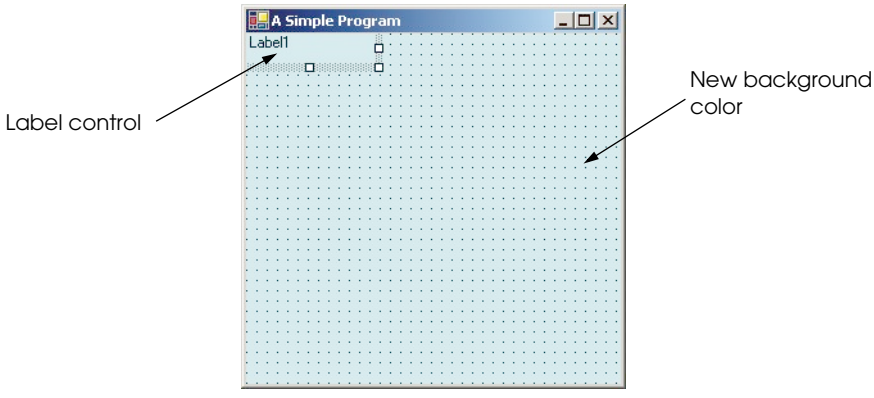


Fig. 2.21 Adding a label to the form.

6. *Set the label's appearance*. Select the label, so that its properties appear in the **Properties** window. The label's **`Text`** property determines the text (if any) that the label displays. The form and label each have their own **`Text`** property. Forms and controls can have the same types of properties (e.g, **`BackColor`**, **`Text`**, etc.) without conflict. Set the **`Text`** property of the label to **Welcome to Visual Basic!** (Fig. 2.22). Resize the label (using the sizing handles) if the text does not fit. Move the label to the top center of the form by dragging it or by using the keyboard's left and right arrow keys to adjust its position. Alternatively, you can center the label control horizontally by selecting **Format > Center In Form > Horizontally**.

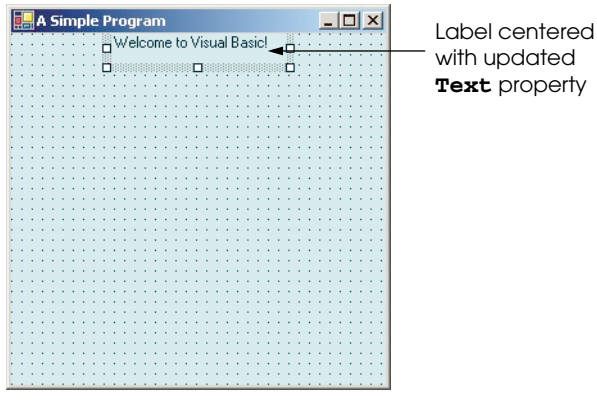


Fig. 2.22 Label in position with its **`Text`** property set.

7. *Set the label's font size, and align the label's text*. Clicking the value of the ***Font*** property causes an *ellipsis* button (**...**) to appear next to the value, as shown in Fig. 2.23. When the ellipsis button is clicked, a dialog that provides additional values—in this case, the **Font** dialog (Fig. 2.24)—is displayed. Programmers can select the font name (**MS Sans Serif**, **Arial**, etc.), font style (**Regular**, **Bold**, etc.) and font size (**8**, **10**, etc.) in this dialog. The text in the **Sample** area displays the selected font. Under the **Size** category, select **24** points, and click **OK**. If the text does not fit on a single line, it will wrap to the next line. Resize the label if it is not large enough to hold the text. Next, select the label's ***TextAlign*** property, which determines how the text is aligned within the label. A three-by-three grid of buttons representing alignment choices will be displayed. The position of each button corresponds to where the text appears in the label (Fig. 2.25). Click the top–center button in the three-by-three grid; this selection will cause the text to appear at the top–center position in the label.

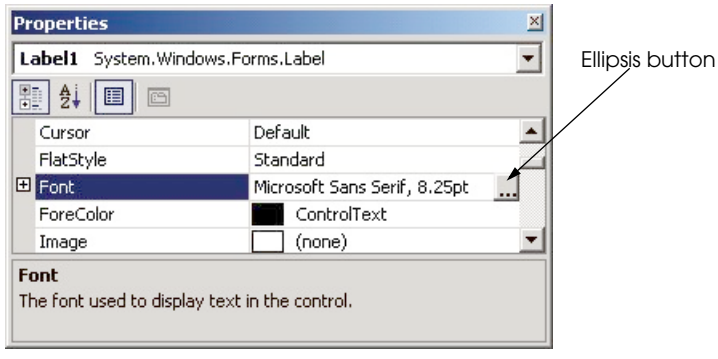


Fig. 2.23 **Properties** window displaying the label's properties.

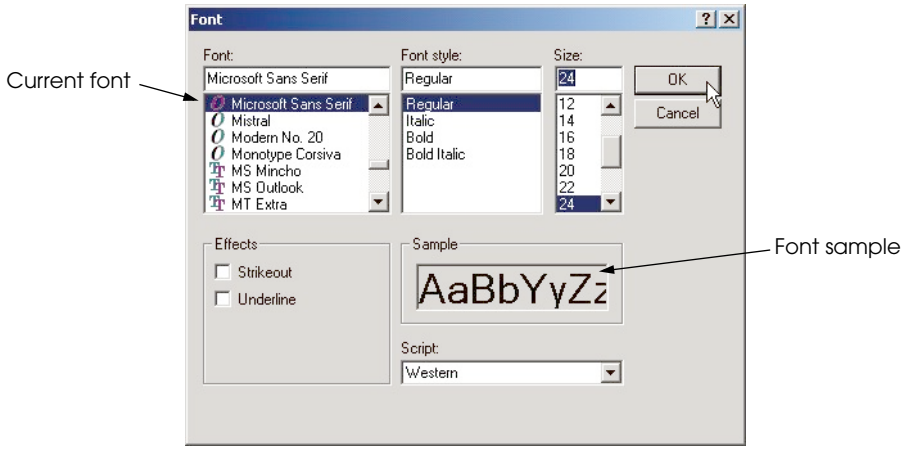


Fig. 2.24 **Font** dialog for selecting fonts, styles and sizes.

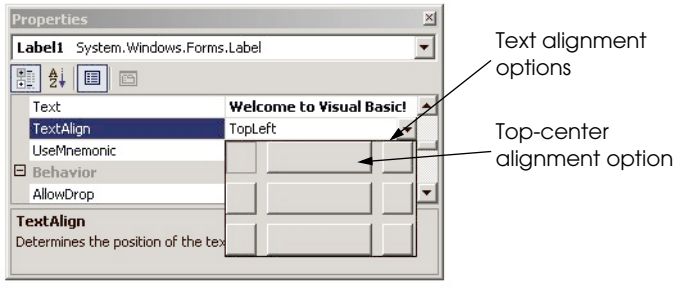


Fig. 2.25 Centering the label's text.

8. *Add a picture box to the form.* The picture-box control is used to display images. The process involved in this step is similar to that of Step 5, in which we added a label to the form. Locate the picture box in the **Toolbox**, and double click it to add it to the form. When the picture box appears, move it underneath the label, either by dragging it or using the arrow keys (Fig. 2.26).

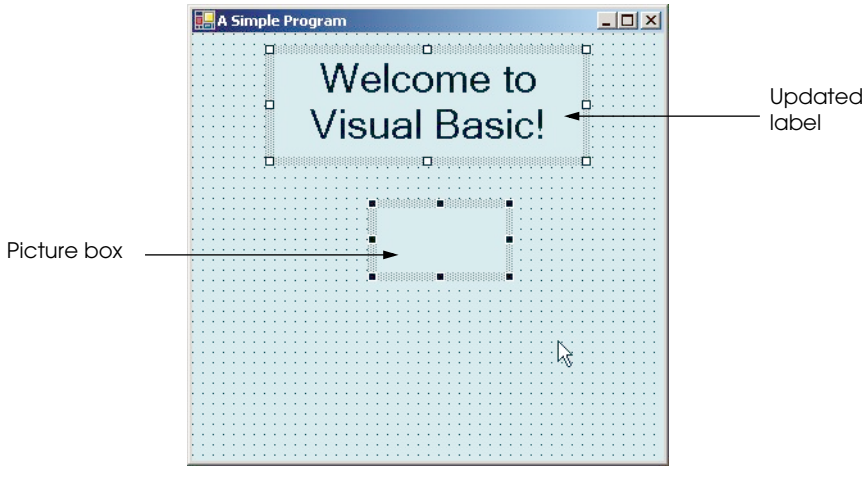


Fig. 2.26 Inserting and aligning the picture box.

9. *Insert an image*. Click the picture box to display its properties in the **Properties** window, and find the ***Image*** *property*, which displays a preview of the image (if one exists). No picture has been assigned (Fig. 2.27), so the value of the ***Image*** property displays **(none)**. Click the ellipsis button to display the **Open** dialog (Fig. 2.28). Browse for an image to insert, select it with the mouse and press the *Enter* key. Supported image formats include *PNG* (*Portable Networks Graphic*), *GIF* (*Graphic Interchange Format*), *JPEG* (*Joint Photographic Experts Group*) and *BMP* (*Windows bitmap*). The creation of a new image requires image-editing software, such as Jasc Paint Shop Pro (**`www.jasc.com`**), Adobe Photoshop Elements (**`www.adobe.com`**) and Microsoft Paint (provided with Windows). In

our case, the picture is **bug.png** and is available on our Web site (**www.deitel.com**) along with this example. Once the image is selected, the picture box displays the image, and the **Image** property displays a small preview. To fit the image in the picture box, change the ***SizeMode*** *property* to ***StretchImage***, which stretches or shrinks the image to the size of the picture box. Resize the picture box by making it larger (Fig. 2.29).

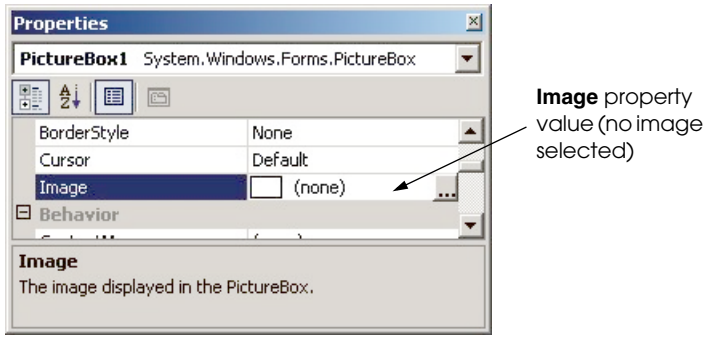


Fig. 2.27 **Image** property of the picture box.

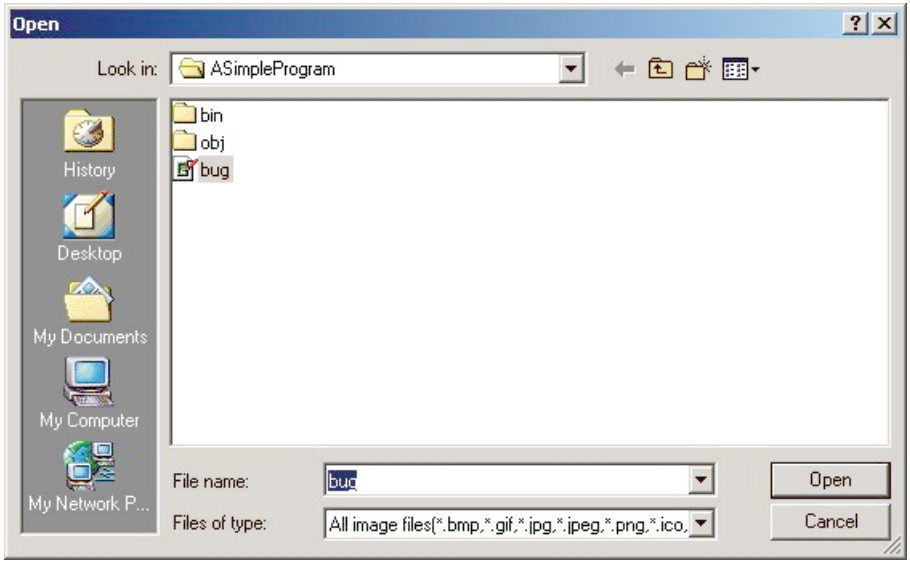


Fig. 2.28 Selecting an image for the picture box.

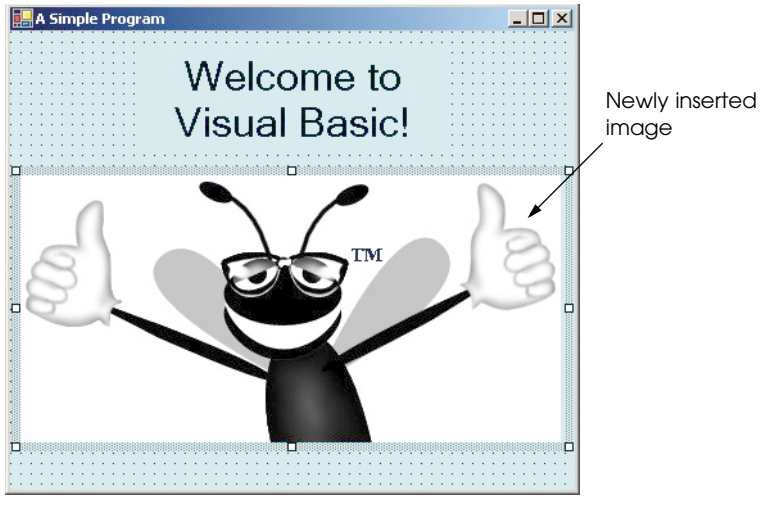


Fig. 2.29 Picture box displaying an image.

10. *Save the project*. Select **File > Save All** to save the entire solution. To save an individual file, select it in the **Solution Explorer**, and select **File > Save**. The solution file contains the name(s) and location(s) of its project(s), and the project file contains the names and locations of all the files in the project.

11. *Run the project*. Up to this point, we have been working in the IDE *design mode* (i.e., the program being created is not executing). This mode is indicated by the text **Microsoft Visual Basic.NET [design]** in the title bar. While in design mode, programmers have access to all the environment windows (e.g., **Toolbox**, **Properties**, etc.), menus, toolbars and so forth. While in *run mode*, the program is executing, and programmers can interact with only a few IDE features. Features that are not available are disabled or grayed out. The text **Form1.vb [Design]** in the title bar means that we are designing the form visually, rather than programmatically. If we had been writing code, the title bar would have contained only the text **Form1.vb**. The program can then be executed by clicking the ***Start*** button (the blue triangle) or by selecting **Debug > Start Without Debugging**. Figure 2.30 shows the IDE in run mode. Note that the IDE title bar displays **[run]** and that many toolbar icons and menus are disabled.

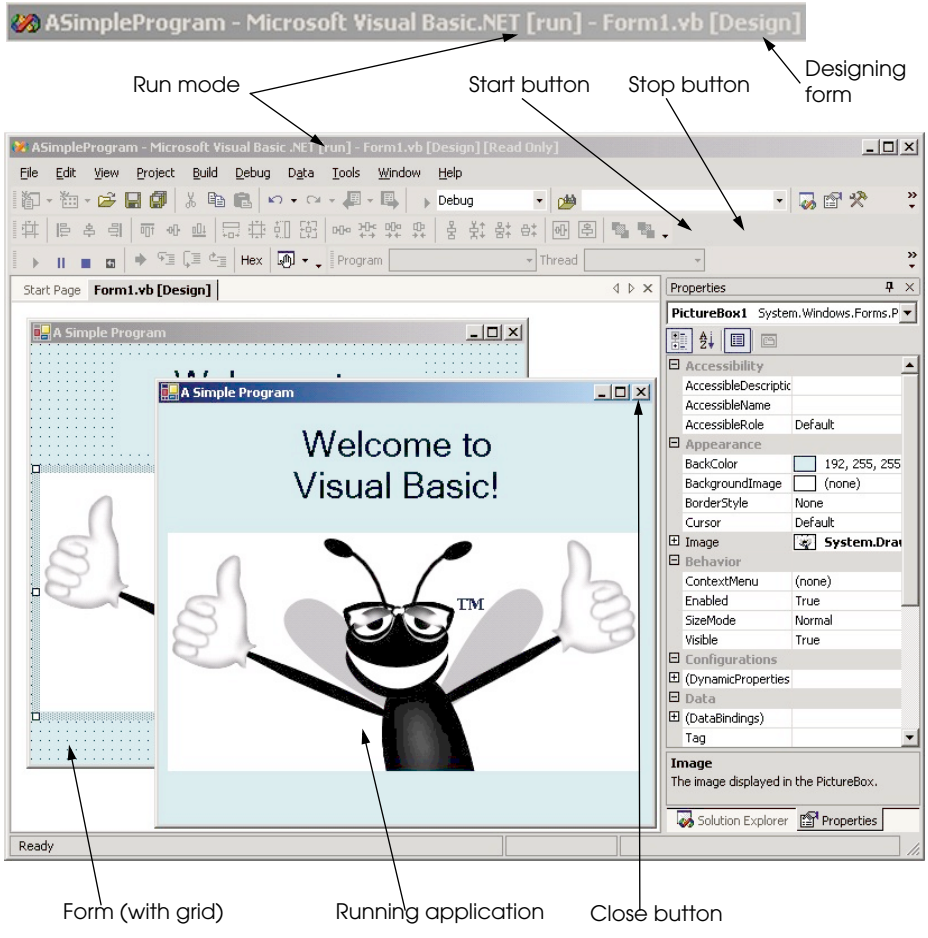


Fig. 2.30 IDE in run mode, with the running application in the foreground.

12. *Terminate execution*. To terminate the program, click the running application's *close* button (the **x** in the top right corner). Alternatively, click the *Stop* button in the toolbar (the blue square). Either action stops program execution and returns the IDE to design mode.

Software Engineering Observation 2.1

Visual programming can be simpler and faster than writing code when developing GUIs.

Software Engineering Observation 2.2

Most programs require more than visual programming. For these programs, at least some, and often all, code must be written by the programmer. Examples of such programs include programs that use event handlers (used to respond to the user's actions), as well as database, security, networking, text-editing, graphics, multimedia and other types of applications.

We have just used visual programming to create a working Visual Basic program without writing a single line of code! In the next chapter, we discuss "nonvisual," or "conventional," programming—we create programs using only code. Visual Basic programming is a mixture of the two styles: Visual programming allows us to develop GUIs easily and avoid tedious GUI programming; conventional programming is employed to specify the behavior of our program. The most important part of a program is its behavior, the programming of which we begin to explain in the next chapter.

SUMMARY

- Visual Studio .NET is Microsoft's Integrated Development Environment (IDE) used by Visual Basic and the languages to create, run and debug programs.
- When Visual Studio .NET is executed, the **Start Page** is displayed. This page contains helpful links, such as recent projects, on-line newsgroups and downloads.
- The **Get Started** page contains links to recent projects.
- The **My Profile** page allows programmers to customize the Visual Studio .NET IDE.
- In the Visual Studio .NET IDE, programmers can browse the Web, using the internal Web browser.
- Dialogs are windows used to communicate with users.
- Programs in the Visual Studio .NET IDE are organized into projects and solutions. A project is a group of related files that form a program, and a solution is a group of projects.
- Windows applications are programs that execute inside the Windows OS, such as Microsoft Word, Internet Explorer and Visual Studio .NET. They contain reusable, graphical components, such as buttons and labels, with which the user interacts.
- The form and its controls constitute the graphical user interface (GUI) of the program and are what users interact with when the program is run. Controls are the graphical components with which the user interacts. Users enter data (inputs) into the program by entering information from the keyboard and by clicking the mouse buttons. The program displays instructions and other information (outputs) for users to read in the GUI.
- The IDE's title bar displays the name of the project, the programming language, the mode of the IDE, the name of the file being viewed and the mode of the file being viewed.
- To view a tabbed document, click the tab displaying the document's name.
- Menus contain groups of related commands that, when selected, cause the IDE to perform some action. They are located on the menu bar.
- The toolbar contains icons that represent menu commands. To execute a command, click the corresponding icon. Click the down-arrow button beside an icon to display additional options.
- Moving the mouse pointer over an icon highlights the icon and displays a description called a tool tip.
- The **Solution Explorer** window lists all the files in the solution.
- The solution's startup project is the project that runs when the program is executed.

- The plus and minus boxes to the left of the name of the project and the **References** folder expand and collapse the tree, respectively.
- The **Toolbox** contains controls for customizing forms.
- By using visual programming, programmers can place predefined controls onto the form instead of writing the code themselves.
- Moving the mouse pointer over the icon of a hidden window opens the window. When the mouse pointer leaves the area of the window, the window is hidden. This feature is known as auto-hide. To "pin down" a window (i.e., to disable auto-hide), click the pin icon in the upper right corner.
- The **Properties** window displays the properties for a form or control. Properties are information about a form or control, such as size, color and position.
- Each control has its own set of properties. The left column of the **Properties** window shows the properties of the control, whereas the right column displays their current values. The toolbar sorts the properties either alphabetically (when the **Alphabetic** icon is clicked) or categorically (when the **Categorized** icon is clicked.)
- The **Properties** window allows programmers to modify controls visually, without writing code.
- The **Help** menu contains a variety of options: The **Contents** menu item displays a categorized table of contents; the **Index** menu item displays an alphabetical index that the programmer can browse; the **Search** feature allows programmers to find particular help articles, based on search keywords. For each of these help features, a filter can be used to narrow the search to articles relating to Visual Basic.
- **Dynamic Help** provides a list of articles based on the current content (i.e., the items around the location of the mouse cursor).
- Context-sensitive help is similar to dynamic help, except that it immediately brings up a relevant help article instead of a list of articles. To use context-sensitive help, click an item, and press *F1* key.
- Visual Basic programming usually involves a combination of writing a portion of the program code and having the Visual Studio .NET IDE generate the remaining code.
- The text that appears at the top of the form (the title bar) is specified in the form's **`Text`** property.
- To resize the form, click and drag one of the form's enabled sizing handles (the small squares around the form). Enabled sizing handles are white; disabled sizing handles are gray.
- The grid on the background of the form is used to align controls and is not displayed at runtime.
- The **`BackColor`** property specifies a form or control's background color. The form's background color is the default background color for any controls added to the form.
- Double-clicking any **Toolbox** control icon places a control of that type on the form. Alternatively, programmers can "drag and drop" controls from the **Toolbox** to the form.
- The label's **`Text`** property determines the text (if any) that the label displays. The form and label each have their own **`Text`** property.
- A property's ellipsis button, when clicked, displays a dialog containing additional options.
- In the **Font** window, programmers can select the font for a form or label's text.
- The **`TextAlign`** property determines how the text is aligned within the label's boundaries.
- The picture-box control is used to display images. The **`Image`** property specifies the image that is displayed.
- Select **File > Save All** to save the entire solution. To save an individual file, select the file in the **Solution Explorer**, and select **File > Save**.

- IDE design mode is indicated by the text **Microsoft Visual Basic.NET [Design]** in the title bar. When in design mode, the program is not executing.
- While in run mode, the program is executing, and programmers can interact with only a few IDE features.
- When designing a program visually, the name of the Visual Basic file appears in the title bar, followed by **[Design]**.
- To execute or run a program, click the **Start** button (the blue triangle), or select **Debug > Start**. The IDE title bar displays **[run]**, and many toolbar icons are disabled.
- Terminate execution by clicking the **Close** button. Alternatively, click the **End** button in the toolbar (the blue square).

TERMINOLOGY

active tab
`Alignment` property
Alphabetic icon
Appearance category in the **Properties** window
application
auto-hide
`BackColor` property
background color
Build menu
button
Categorized icon
clicking
close a project
close button icon
collapse a tree
compile a program
component selection
context-sensitive help
control
control the layout
customize a form
customize Visual Studio .NET
Data menu
debug a program
Debug menu
design mode
dialog
double-clicking
down arrow
dynamic help
Dynamic Help window
Edit menu
expand a tree
external help
F1 help key
File menu
find

`Font` property
font size
font style
Font window
form
Format menu
form's background color
form's title bar
GUI (graphical user interface)
Help menu
icon
IDE (integrated development environment)
input
internal help
internal Web browser
Internet Explorer
label
menu
menu item
menu bar in Visual Studio .NET
mouse pointer
new project in Visual Studio .NET
opening a project
output
palette
paste
picture box
pin a window
print a project
project
Project menu
Properties window
property for a form or control
recent project
Run menu
run mode
selecting
single-clicking with left the mouse button
sizing handle
solution
Solution Explorer in Visual Studio .NET
Start button
Start Page
startup project
`StretchImage` property
tabbed window
`Text` property
title bar
tool tip
toolbar

toolbar icon
Tools menu
undo
View menu
Visual Studio .NET
window layout
Windows application
Windows menu

SELF-REVIEW EXERCISES

2.1 Fill in the blanks in each of the following statements:

a) The technique of _____ allows programmers to create GUIs without writing any code.
b) A _____ is a group of one or more projects that collectively form a Visual Basic program.
c) The _____ feature hides a window when the mouse pointer is moved outside the window's area.
d) A _____ appears when the mouse pointer hovers over an icon.
e) The _____ window allows programmers to browse solution files.
f) A plus box indicates that the tree in the **Solution Explorer** can _____.
g) The **Properties** window's properties can be sorted _____ or _____.
h) A form's _____ property specifies the text displayed in the form's title bar.
i) The _____ allows programmers to add controls to the form in a visual manner.
j) _____ displays relevant help articles, based on the current context.

2.2 State whether each of the following is *true* or *false*. If *false*, explain why.

a) The title bar displays the IDE's mode.
b) The programmer can customize the IDE by clicking **Get Started** on the **Start Page**.
c) The **x** button toggles auto hide.
d) The toolbar icons represent various menu commands.
e) The toolbar contains control icons.
f) A form's sizing handles are always enabled.
g) Both forms and labels have a title bar.
h) Controls can be modified only by writing code.
i) Buttons typically perform actions when clicked.
j) A form's grid is visible only at design time.

ANSWERS TO SELF-REVIEW EXERCISES

2.1 a) visual programming. b) solution. c) auto-hide. d) tool tip. e) **Solution Explorer**. f) expand. g) alphabetically, categorically. h) **`Text`**. i) **Toolbox**. j) **Dynamic help**.

2.2 a) True. b) False. The programmer can customize the IDE by clicking the **My Profile** link on the **Start Page**. c) False. The pin icon toggles auto-hide. d) True. e) False. The **Toolbox** contains control icons. f) False. Some of a form's sizing handles are disabled. g) False. Forms have a title bar, but labels do not. h) False. Control properties can be set using the **Properties** window. i) True. j) True.

EXERCISES

2.3 Fill in the blanks in each of the following statements:

a) The _____ button in the **Properties** window displays a dialog when clicked.

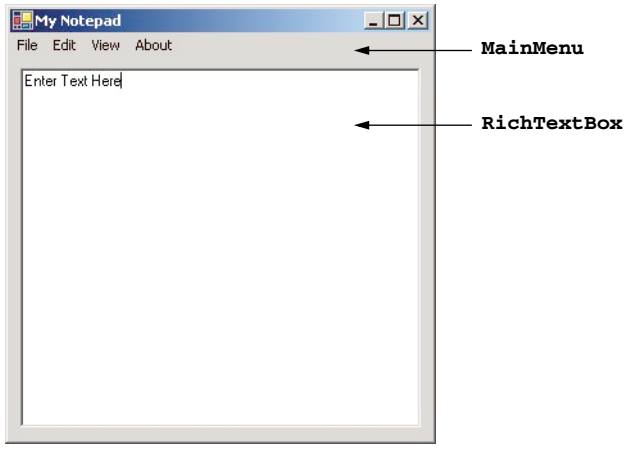
b) To save every file in a solution, select _____.
c) _____ help immediately displays a relevant help article. It can be accessed using the _____ key.
d) "GUI" is an acronym for _____.

2.4 State whether each of the following is *true* or *false*. If *false*, explain why.
a) A control can be added to a form by double-clicking its control icon in the **Toolbox**.
b) The form, label and picture box have identical properties.
c) Programmers can browse the Internet from the Visual Studio .NET IDE.
d) Visual Basic programmers often create complex applications without writing any code.
e) Sizing handles are visible during execution.

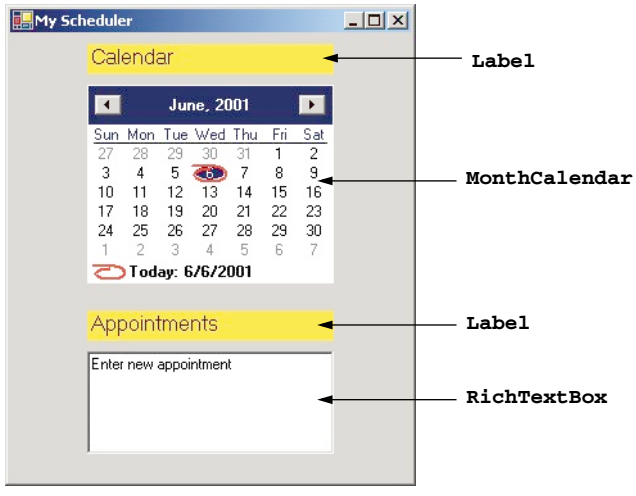
2.5 Some features that appear throughout Visual Studio .NET perform similar actions in different contexts. Explain and give examples of how plus and minus boxes, ellipsis buttons, down-arrow buttons and tool tips act in this manner. Why do you think the Visual Studio .NET IDE was designed this way?

2.6 Build the GUIs given in each part of this exercise. (You need not provide any functionality.) Execute each program, and determine what happens when a control is clicked with the mouse. Drag controls from the **Toolbox** onto the form and resize them as necessary.

a) This GUI consists of a **`MainMenu`** and a **`RichTextBox`**. After inserting the **`MainMenu`**, add items by clicking in the **Type Here** section, typing a menu name and pressing *Enter*. Resize the **`RichTextBox`** to fill the form.



b) This GUI consists of two **`Label`**s (12-point font size, yellow background), a **`MonthCalendar`** and a **`RichTextBox`**. The calendar is displayed when the **`MonthCalendar`** is dragged onto the form. [*Hint*: Use the **`BackColor`** property to change the background color of the labels.]



2.7 Fill in the blanks in each of the following statements:

a) A _____ contains a series of colors.
b) The _____ is part of the **Toolbox**, but is not a control.
c) The _____ contains commands for arranging and displaying windows.
d) Property _____ determines a form or control's background color.

2.8 Briefly describe each of the following terms:

a) toolbar
b) menu bar
c) Toolbox
d) control
e) form
f) project
g) title bar
h) solution