The Integrated Development Environment

Much of the popularity of Visual Basic comes from its Integrated Development Environment, or IDE for short. In theory, you can edit your Visual Basic programs using any editor, including the aged Notepad, but I never met a programmer insane enough to do that. In fact, the IDE gives you everything you need to create great applications, to write code for them, to test and fine-tune them, and, finally, to produce executable files. These files are independent of the environment and therefore can be delivered to customers for execution on their machines, even if they haven't installed Visual Basic.

Running the IDE

You can choose from several ways to launch the Visual Basic IDE, as is true for any Windows executable:

- You can run the Visual Basic 6 environment from the Start Menu; the exact path to the menu command depends on whether you have installed Visual Basic as part of the Microsoft Visual Studio suite.
- You can create a shortcut to the IDE on your desktop and run it by simply double-clicking on it.
- When Visual Basic is installed, it registers the .vbp, .frm, .bas, and a few other extensions with the operating system. Therefore, you can run the environment by double-clicking on any Visual Basic file.
- If you have installed Microsoft Active Desktop, you can create a shortcut to the Visual Basic IDE on the system taskbar. This is probably the fastest way to run the IDE: it's similar to a desktop shortcut, but you don't have to minimize other windows to uncover it.

Don't underestimate the convenience of running the Visual Basic IDE in the fastest way possible. When you develop COM components or add-ins, you might need to follow the commonplace practice of opening multiple instances of the environment at the same time. You might need to repeat this operation several times during your working day.

Selecting the Project Type

The first time you run the Visual Basic IDE, you're asked to select the type of project you want to create, as you can see in Figure 1-1. In this chapter, as well as in many chapters in the first part of this book, we're going to create Standard EXE projects only, so you can click on the Open button—or just press the Enter key—to start working with a regular project that, once compiled, will deliver a stand-alone EXE application. You can also decide to tick the "Don't show this dialog in future" check box if you want to avoid this operation the next time you launch the IDE.

Figure 1-1. *The New Project dialog box that appears when you launch the Visual Basic 6 environment.*

IDE Windows

If you have worked with Visual Basic 5, the Visual Basic 6 IDE will look very familiar to you, as you can see in Figure 1-2. In fact, the only indication that you're not interacting with Visual Basic 5 is a couple of new top-level menus—Query and Diagram—and two new icons on the standard toolbar. When you begin to explore the IDE's menus, you might find a few other commands (in the Edit, View, Project, and Tools menus) that were missing in Visual Basic 5. But overall changes are minimal, and if you're familiar with the Visual Basic 5 environment you can start working with Visual Basic 6 right away.

On the other hand, if you have worked only with versions of Visual Basic earlier than 5, you're going to be surprised by the many changes in the working environment. For one thing, the IDE is now an MDI (Multiple Document Interface) application, and you can reduce it and its dependent window with a single operation. You can restore the SDI (Single Document Interface) working mode, if you prefer, by choosing Options from the Tools menu, clicking the Advanced tab, and ticking the SDI Development Environment check box.

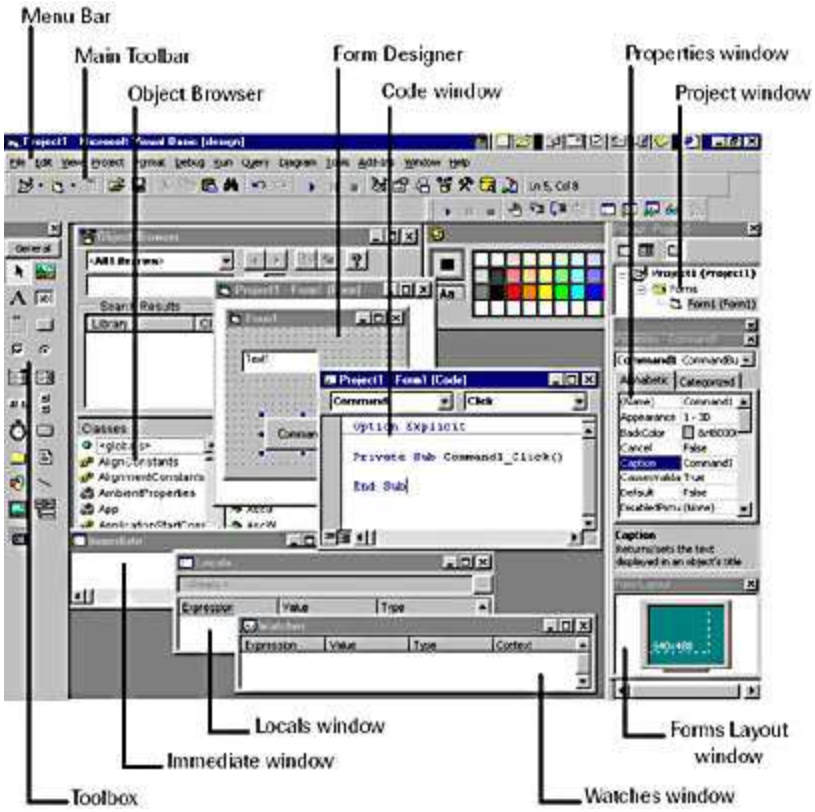

Figure 1-2. *The Visual Basic 6 environment with most windows opened.*

Finally, if this is your first exposure to Visual Basic, you'll surely be confused by the many menu commands, toolbars, and windows that the IDE hosts. Let's quickly review the purpose of each item. You can display any of the windows using an appropriate command in the View menu. Many of them can also be opened using a keyboard shortcut, as described in the following paragraphs, or by clicking on an icon in the main toolbar.

- The Project window gives you an overview of all the modules that are contained in your application. You can display such modules grouped by their types or in alphabetical order by clicking on the rightmost icon on the Project window's toolbar. You can then view the code in each module or the object associated with each module (for example, a form) by clicking the first or the second icon, respectively. You can quickly display the Project window or put it in front of other windows by pressing the Ctrl+R key combination or by clicking on the Project Explorer icon on the standard toolbar.

- You use the Form Designer window to design your application's user interface. Any application can contain multiple forms, and you can open a number of form designers at the same time. Moreover, both Visual Basic 5 and 6 support additional designers, such as the UserControl and UserDocument designers.

- The Toolbox window includes a set of objects that you can place on a form or on another designer. Visual Basic comes with a fixed set of controls—the so-called *intrinsic controls*—but you can add other Microsoft ActiveX controls to this window. To avoid filling this window with too many controls, you can create multiple *tabs* on it: just right-click on the window, select the Add Tab command, and at a prompt assign a name to the new tab. Then you can place additional ActiveX controls on this new tab or drag one or more controls from the General tab. Similarly, you can delete or rename any tab by right-clicking on it and selecting the Delete Tab or Rename Tab commands, respectively. You can't delete or rename the General tab, though.

- You use the Code window to write code that determines the behavior of your forms and other objects in your application. You can keep a number of code windows visible at one time, each one displaying the code related to a form or, more generally, to a module in your application. You can't open two code windows for the same module; however, you can split a code window into two distinct and independent portions by dragging the small gray rectangle located immediately above the vertical scrollbar.

TIP

You can quickly show the code window associated with a form or another designer by pressing the F7 function key while the focus is on the designer. Similarly, if you have opened the code window related to a designer, press the Shift-F7 key combination to display the associated designer.

- The Properties window lists all the properties of the object that's currently selected and gives you the opportunity to modify them. For instance, you can change the foreground and background colors of the form or the control that's currently selected. You can list properties in alphabetical order or group them in categories, and you can find a short description of the currently selected property near the bottom of this window. When you select an object, its properties are automatically displayed in the Properties window. If the window isn't visible, you can quickly display it by pressing the F4 key or by clicking on the Toolbox icon on the toolbar.

- The Color Palette window is handy for quickly assigning a color to an object, such as the control that's currently selected on a form designer. You can select the foreground color of an object by left-clicking on the desired color, and if you click on an empty item in the bottom row of cells you can also define a custom color. Finally, you can select the background color of an object if you click on the larger square located near the upper left corner of the Color Palette window.

- The Form Layout window shows how a given form will be displayed when the program runs. You can drag a form to the place on the screen where you want it to appear during execution, and you can also compare the relative sizes and positions of two or more forms. By right-clicking on this window, you can show its *resolution guides,* which enable you to check how your forms display at screen resolutions different from the current one.

- The Immediate window lets you enter a Visual Basic command or expression and see its result using the *Print* command (which can also be shortened to *?*). In *break mode*—that is, when you have temporarily suspended a running program—you can use these commands to display the current value of a variable or an expression. You can also write diagnostic messages from your application's code to this window using the *Debug.Print* statement. Because the Immediate window isn't visible when the application is compiled and executed outside the environment, such diagnostic statements aren't included in the executable file. You can quickly open the Immediate window by pressing the Ctrl+G key combination.

TIP

No menu command or toolbar icon lets you delete the current contents of the Immediate window. The quickest way to do it is by pressing the Ctrl+A key combination to select the Immediate window's entire contents, after which you press the Delete key to delete it or simply begin typing to replace the contents with whatever you want to type in instead.

- The Object Browser is one of the most important tools available to the Visual Basic developer. It lets you explore external libraries so that you can learn about the objects they expose and their properties, methods, and events. The Object Browser also helps you quickly locate and jump to any procedure in any module of your application. You can open the Object Browser by pressing the F2 key or by clicking its icon on the standard toolbar.

- The Locals window is active only when a program is executing. It lists the values of all the variables that are local to a module or to a procedure. If the variable is an object itself (a form or a control, for example) a plus (+) sign appears to the left of the variable name, which means that you can expand it and look at its properties.

- The Watches window has a dual purpose: it lets you continuously monitor the value of a variable or an expression in your program—including a global variable, which is outside the capabilities of the Locals window—and it also gives you the ability to stop the execution of a program when a given expression becomes True or whenever it changes its value. You can add one or more watch expressions using the Add Watch command from the Debug menu, or you can select the Add Watch command from the pop-up menu that appears when you right-click the Watches window itself.

- The Call Stack window (not visible in Figure 1-2) appears only when you break the execution of a running program and press Ctrl+L. It shows all the procedures that are waiting for the current procedure to complete. It's a useful debugging tool in that it lets you understand the execution path that led to the current situation. Note that this is a modal window, so you must close it to resume regular execution.

- The Data View window is new to Visual Basic. (See Figure 1-3 below.) In a nutshell, the Data View window offers an integrated tool to administer your databases and to explore their structures and the attributes of their tables and fields. The Data View window is particularly versatile when connecting to a Microsoft SQL Server or Oracle database because you can also add and delete its tables, views, and fields and edit stored procedures. Regardless of the database you're using, you can often drag tables and fields from the Data View window onto other windows in the IDE. You display this window using the Data View command from the View menu or by clicking on its icon on the standard toolbar.

Most of the windows I've just described can be *docked*: in other words, they can stick to the external frame of the main window of the IDE and are always on top of all other windows. By default, the majority of IDE windows are docked, although only the Toolbox, Project, Properties, and Form Layout windows are visible when the environment is launched. You can switch the *Docked* attribute on and off for a single window by right-clicking in it and then selecting the Dockable menu command. Alternatively, you can modify the *Docked* attribute individually for all the windows in the IDE by choosing Options from the Tools menu and then clicking the Docking tab. Just tick the check box associated with any window you want to be docked.

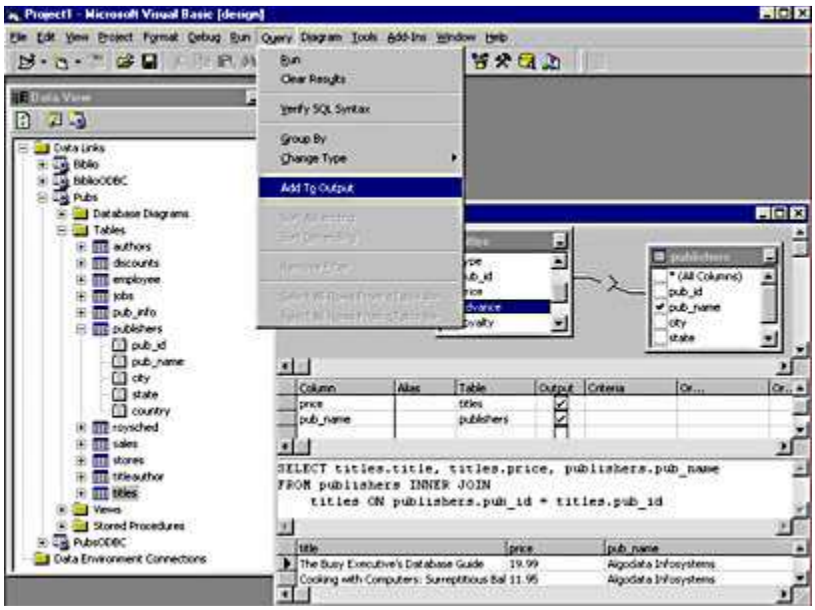

Figure 1-3. *The Data View window lets you interactively create a new View object in an SQL Server database.*

Menus

It's unnecessary to describe in detail the purpose of each menu command at this point in the book because most of the commands are related to advanced features of Visual Basic. But I think that an overview of all the top-level menus is useful, in that it gives you an idea of where to look for a given function when you need it.

- The File menu includes the commands to load and save a Visual Basic project or a group of projects (Visual Basic 6 can open multiple projects in the environment), to save the current module, to print the entire project or selected portions of it, and to build the executable file.

- The Edit menu lets you perform the typical editing commands, including Cut, Copy, Paste, Find, Replace, Undo, and Redo. It also includes commands that act on database tables, but they are active only when you're viewing the structure of a database or a database diagram. This menu also includes a bunch of commands related to Microsoft IntelliSense, a feature of the Visual Basic IDE that lets you automatically complete commands, list the syntax of functions and expected arguments, and so forth.

- The View menu is the primary means of displaying any of the environment's windows described previously. It also includes some database-related commands that are enabled only if you have activated a tool for database maintenance.

- The Project menu lets you add modules to the current project, including forms, standard (BAS) modules, class modules, UserControl modules, and so on. It also lets you add *designer* modules, which are the key to many new Visual Basic 6 features. The last three commands in this menu are particularly useful because they give you access to the References, the Components, and the Project Properties dialog boxes, respectively.

- The Format menu is used to align and resize one or more controls on a form or on a designer of any type. You can also center a control on its form and increase or decrease the distance among a group of controls. When you're satisfied with the appearance of your form, you should select the Lock Controls option so that you can't accidentally move or resize the controls using the mouse.

- The Debug menu contains the commands that you usually issue when you're testing an application within the IDE. You can execute your code step-by-step, display the value of a variable or an expression, and set one or more *breakpoints* in code. Breakpoints are special points in the code that, when reached during the program's execution, cause the Visual Basic environment to interrupt execution, thus entering break mode. You can also create *conditional breakpoints,* which are expressions that are monitored as each statement is executed. When the value of a conditional breakpoint changes or when the condition you specified becomes True, the program enters break mode and you can debug it easily.

- The Run menu is probably the simplest of the group. It contains the commands to start the execution of the application being developed, to stop it and enter break mode, and to definitively end it.

- The Query menu is new to the Visual Basic environment. It's available only in Visual Basic Enterprise and Professional Editions and only when you're interactively creating an SQL query using the Microsoft Query Builder utility, which you can see in Figure 1-3.

- The Diagram menu, shown in Figure 1-4, is also new to Visual Basic. As with the Query menu, the Diagram menu is available only in the Enterprise and Professional Editions and only when you're interacting with SQL Server or Oracle databases to create or edit a database diagram.

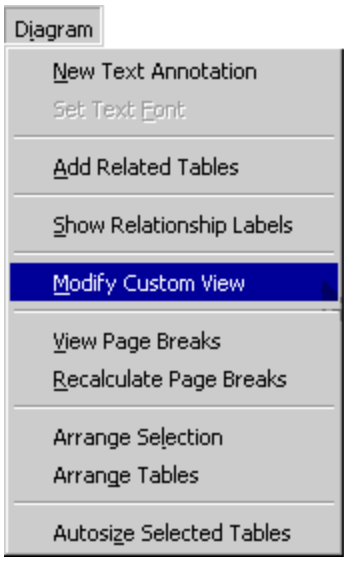

Figure 1-4. *The Diagram menu becomes active only when you're building a query or editing an SQL Server View object.*

- The Tools menu contains several miscellaneous commands, the most important of which is the Options command. This command allows you to access a dialog box that lets you customize the IDE.
- The Add-In menu lists a collection of commands related to external modules that integrate into the environment. Visual Basic 6 itself comes with a number of such external add-ins, and you can also write your own.
- The Window menu is the typical menu that you find in most MDI applications; it lets you arrange and tile your windows.
- The Help menu is also standard for Microsoft Windows applications. Visual Basic 6 doesn't use standard HLP files any longer, and its help subsystem requires that you install Microsoft Developer Network (MSDN) to access its documentation.

Toolbars

Visual Basic comes with a standard toolbar that includes many common commands, such as those for loading and saving the project, running the program, and opening the most frequently used windows. Three more toolbars, Debug, Edit, and Form Editor, are visible only after you right-click on the standard toolbar and select one toolbar at a time from the submenu that appears. You can also make these toolbars visible by selecting the Toolbars option from the View menu.

All the toolbars can be docked in the upper portion of the main IDE window, or they can freely float in the environment, as you can see in Figure 1-5. You can quickly dock a floating toolbar by double-clicking on its title bar, and you can make a docked toolbar float by double-clicking on its left-most vertical stripes. If you want to know what a particular toolbar icon represents, place the mouse cursor over it and a yellow ToolTip showing a short explanation will appear after about a second.

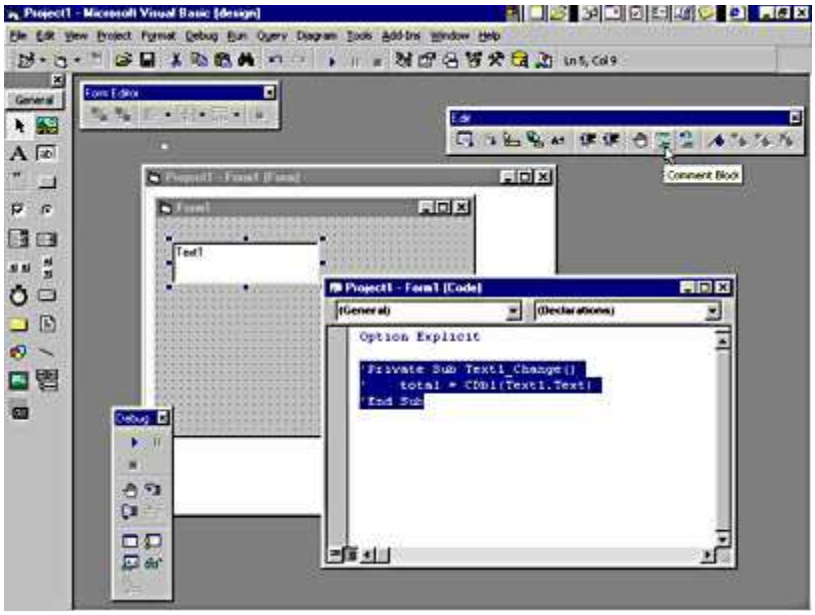

Figure 1-5. *Visual Basic 6 comes with four toolbars, which can float or be docked.*

The Debug toolbar hosts most of the commands that are found in the Debug menu. The Edit toolbar is useful when you're editing code and setting breakpoints and bookmarks. The Form Editor toolbar includes most of the commands in the Format menu and is useful only when you're arranging controls on a form's surface.

Making these additional toolbars visible or not is largely a matter of personal taste. I usually prefer not to waste valuable desktop space with toolbars other than the standard one. If you work with a higher screen resolution, this might not be an issue for you.

> **TIP**
>
> The Edit toolbar is unusual because it contains two commands that aren't available through menu commands—the Comment Block and Uncomment Block commands, which are useful when you're testing an application. (See Figure 1-5 for an example of a routine that has been commented using the Comment Block command.) For this reason, you might want to make the Edit toolbar visible.

You can customize the appearance of all the Visual Basic toolbars and even create new ones, as you can see in Figure 1-6. The procedure for creating a new toolbar is simple:

1. Right-click on any toolbar, and select the Customize menu command; this brings up the Customize dialog box.
2. Click the New button, and type a name for the new custom toolbar (for example, *Custom Toolbar*). The name of the new toolbar appears in the list of toolbars, and its check box is ticked. The empty toolbar appears on the screen. You're now ready to add commands to it.
3. Click the Commands tab, and then click a menu name in the leftmost list box. Click on an item in the list box on the right, and drag it over the custom toolbar to the spot where you want to insert it.
4. Right-click on the icon you have just added, and select a command from the pop-up menu that appears. The commands in this menu let you replace the icon with a different one, associate it with a caption, make it the beginning of a group, and so on.
5. Repeat steps 3 and 4 for all the commands you want to add to the custom toolbar, and then click on the Close button to make your additions permanent.

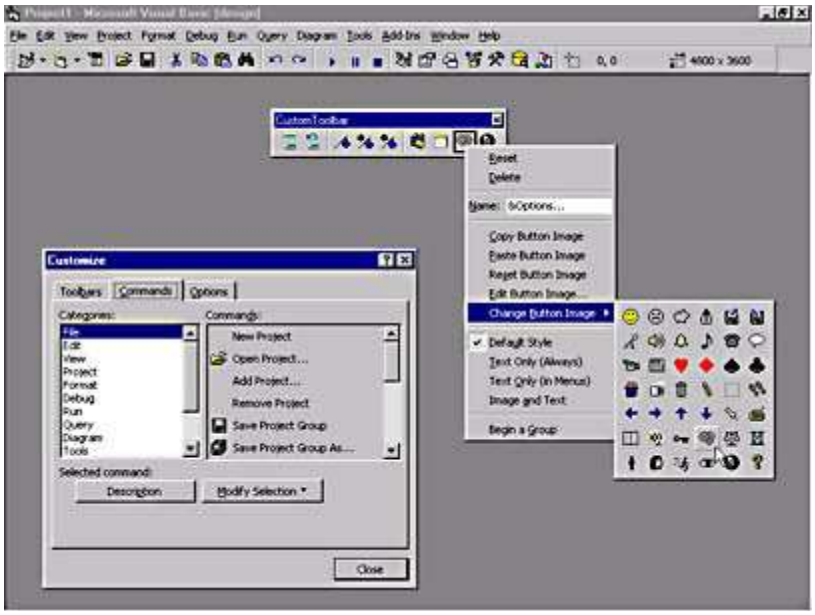

Figure 1-6. *Creating a custom toolbar.*

Here are a few commands that you should consider for inclusion in a custom toolbar because they're frequently used but don't have any associated hot keys:

- The References and Properties commands from the Project menu
- The Comment Block and Uncomment Block commands from the Edit toolbar (not displayed on the menu)
- All the Bookmark submenu commands from the Edit menu
- The Options command from the Tools menu

The Toolbox

The Toolbox window is probably the first window you'll become familiar with because it lets you visually create the user interface for your applications. More specifically, the Toolbox contains the icons of all the intrinsic controls—that is, all the controls that are included in the Visual Basic runtime.

If you have already programmed with a previous version of Visual Basic, you surely know the characteristics of all the controls that are present in the Toolbox. If you haven't, refer to Figure 1-7 while you read the following condensed descriptions.

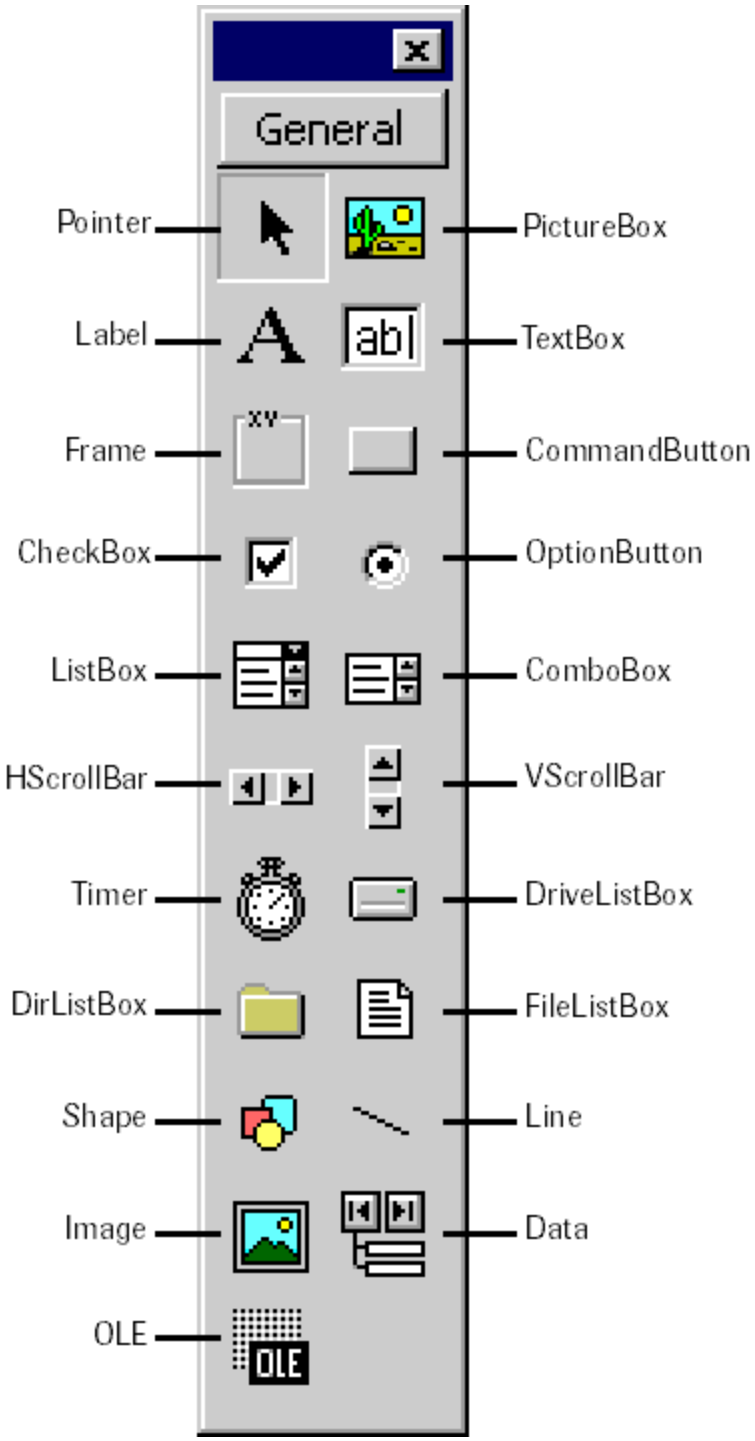

Figure 1-7. *The Visual Basic 6 Toolbox with all the intrinsic controls.*

- The Pointer isn't a control; click this icon when you want to select controls already on the form rather than create new ones.
- The PictureBox control is used to display images in any of the following formats: BMP, DIB (bitmap), ICO (icon), CUR (cursor), WMF (metafile), EMF (enhanced metafile), GIF, and JPEG.
- The Label control serves to display static text or text that shouldn't be edited by the user; it's often used to label other controls, such as TextBox controls.

- The TextBox control is a field that contains a string of characters that can be edited by the user. It can be single-line (for entering simple values) or multiline (for memos and longer notes). This is probably the most widely used control of any Windows application and is also one of the richest controls in terms of properties and events.

- The Frame control is typically used as a container for other controls. You rarely write code that reacts to events raised by this control.

- The CommandButton control is present in almost every form, often in the guise of the OK and Cancel buttons. You usually write code in the *Click* event procedure of this control.

- The CheckBox control is used when the user has to make a yes/no, true/false selection.

- OptionButton controls are always used in groups, and you can select only one control in the group at a time. When the user selects a control in the group, all other controls in the group are automatically deselected. OptionButton controls are useful for offering to the user a number of mutually exclusive selections. If you want to create two or more groups of OptionButton controls on a form, you must place each group inside another container control (most often a Frame control). Otherwise, Visual Basic can't understand which control belongs to which group.

- The ListBox control contains a number of items, and the user can select one or more of them (depending on the value of the control's *MultiSelect* property).

- The ComboBox control is a combination of a TextBox and a ListBox control, with the difference that the list portion is visible only if the user clicks on the down arrow to the right of the edit area. ComboBox controls don't support multiple selections.

- The HScrollBar and VScrollBar controls let you create stand-alone scroll bars. These controls are used infrequently because the majority of other controls display their own scroll bars if necessary. Stand-alone scroll bars are sometimes used as sliders, but in this case you'd better use other, more eye-catching controls, such as the Slider control, which is covered in Chapter 10.

- The Timer control is peculiar in that it isn't visible at run time. Its only purpose is to regularly raise an event in its parent form. By writing code in the corresponding event procedure, you can perform a task in the background—for instance, updating a clock or checking the status of a peripheral device.

- The DriveListBox, DirListBox, and FileListBox controls are often used together to create file-oriented dialog boxes. DriveListBox is a ComboBox-like control filled automatically with the names of all the drives in the system. DirListBox is a variant of the ListBox control; it shows all the subdirectories of a given directory. FileListBox is another special ListBox control; this control fills automatically with names of the files in a specified directory. While these three controls offer a lot of functionality, in a sense they have been superseded by the Common Dialog control, which displays a more modern user interface (to be covered in Chapter 12). If you want to write applications that closely conform to the Windows 9x look, you should avoid using these controls.

- The Shape and Line controls are mostly cosmetic controls that never raise any events and are used only to display lines, rectangles, circles, and ovals on forms or on other designers.

- The Image control is similar to the PictureBox control, but it can't act as a container for other controls and has other limitations as well. Nevertheless, you should use an Image control in place of a PictureBox control whenever possible because Image controls consume fewer system resources.

- The Data control is the key to *data binding,* a Visual Basic feature that lets you connect one or more controls on a form to fields in a database table. The Data control works with Jet databases even though you can also use attached tables to connect to data stored in databases stored in other formats. But it can't work with ActiveX Data Objects (ADO) sources and is therefore not suitable for exploiting the most interesting database-oriented Visual Basic 6 features.

- The OLE control can host windows belonging to external programs, such as a spreadsheet window generated by Microsoft Excel. In other words, you can make a window provided by another program appear as if it belongs to your Visual Basic application.

From this short description, you can see that not all the intrinsic controls are equally important. Some controls, such as the TextBox, Label, and CommandButton controls, are used in virtually every Visual Basic application, while other controls, such as the DriveListBox, DirListBox, and FileListBox controls, have been replaced, in practice, by newer controls. Similarly, you shouldn't use the Data control in any application that uses the ADO data sources.