One of a Kind

Selected Material from

Programming with Microsoft Visual Basic 5.0 *for windows.* with Microsoft Internet Explorer 4

Diane Zak
Gary B. Shelly
Thomas J. Cashman
Kurt A. Jordan

ITP Custom Courseware

Selected Material from

Programming with Microsoft **Visual Basic 5.0** *for Windows.* with Microsoft **Internet Explorer 4**

Diane Zak
Gary B. Shelly
Thomas J. Cashman
Kurt A. Jordan

SHELLY CASHMAN SERIES®

COURSE TECHNOLOGY
ONE MAIN STREET
CAMBRIDGE MA 02142
an International Thomson Publishing company I(T)P®

CAMBRIDGE • ALBANY • BONN • CINCINNATI • LONDON • MADRID • MELBOURNE
MEXICO CITY • NEW YORK • PARIS • SAN FRANCISCO • TOKYO • TORONTO • WASHINGTON

Printed in the United States of America

For more information, contact Course Technology, One Main Street, Cambridge MA 02142, or electronically at http://www.thomson.com/cti.html

International Thomson Publishing Europe
Berkshire House 168-173
High Holborn
London, WC1V 7AA, England

International Thomson Editores
Campos Eliseos 385, Piso 7
Col. Polanco
11560 México D.F. México

Thomas Nelson Australia
102 Dodds Street
South Melbourne 3205
Victoria, Australia

International Thomson Publishing Asia
221 Henderson Road
#05-10 Henderson Building
Singapore 0315

Nelson Canada
1120 Birchmount Road
Scarborough, Ontario
Canada M1K 5G4

International Thomson Publishing Japan
Hirakawacho Kyowa Building, 3F
2-2-1 Hirakawacho
Chiyoda-ku, Tokyo 102, Japan

International Thomson Publishing GmbH
Königswinterer Strasse 418
53227 Bonn, Germany

International Thomson Publishing Southern Africa
Building 18, Constantia Park
240 Old Pretoria Road
Halfway House, 1685 South Africa

ISBN 0-7600-1120-6

Custom Contents

8. Command buttons in an interface should be ________________.
   a. centered along the bottom of the screen
   b. stacked in either the upper-left or lower-left corner of the screen
   c. stacked in either the upper-right or lower-right corner of the screen
   d. either a or b
   e. either a or c
9. Use no more than ________________ command buttons on a screen.
   a. five
   b. four
   c. seven
   d. six
   e. two
10. If more than one command button is used in an interface, the most commonly used command button should be placed ________________.
    a. first
    b. in the middle
    c. last
    d. either a or c
11. Which of the following statements is false?
    a. A command button's caption should appear on one line.
    b. A command button's caption should be from one to three words only.
    c. A command button's caption should be entered using book title capitalization.
    d. A command button's caption should end with a colon (:).
12. The labels for controls (other than command buttons) should be entered using ________________.
    a. book title capitalization
    b. sentence capitalization
    c. either a or b
13. Which of the following statements is false?
    a. Labels for controls (other than command buttons) should be aligned on the left.
    b. Labels for controls (other than command buttons) should be positioned either above or to the left of the control.
    c. Labels for controls (other than command buttons) should be entered using book title capitalization.
    d. Labels for controls (other than command buttons) should end with a colon (:).
14. Use ________________ for labels, which means you capitalize only the first word and any words that are customarily capitalized.
    a. book title capitalization
    b. sentence capitalization
15. Use ________________ for command button captions, which means you capitalize the first letter in each word, except for articles, conjunctions, and prepositions that do not occur at either the beginning or the end of the caption.
    a. book title capitalization
    b. sentence capitalization
16. Listed below are the four steps you should follow when planning an OOED application. Put them in the proper order by placing a number (1 to 4) on the line to the left of the step.

    ______ Identify the objects to which you will assign those tasks.
    ______ Draw a sketch of the user interface.
    ______ Identify the tasks the application needs to perform.
    ______ Identify the events required to trigger an object into performing its assigned tasks.

17. Listed below are the five steps you should follow when creating an OOED application. Put them in the proper order by placing a number (1 to 5) on the line to the left of the step.
______ Test and debug the application.
______ Build the user interface.
______ Code the application.
______ Assemble the documentation.
______ Plan the application.

E X E R C I S E S

1. In this exercise, you will prepare a TOE chart and create two sketches of the application's user interface.

   Scenario: Sarah Brimley is the accountant at Paper Products. The salespeople at Paper Products are paid a commission, which is a percentage of the sales they make. The current commission rate is 10%. (In other words, if you have sales totaling $2,000, your commission is $200.) Sarah wants you to create an application that will compute the commission after she enters the salesperson's name, territory number, and sales. She also wants to print this information.
   a. Prepare a TOE chart ordered by task.
   b. Rearrange the TOE chart created in step a so that it is ordered by object.
   c. Draw two sketches of the user interface—one using a horizontal arrangement and the other using a vertical arrangement.
2. In this exercise, you will prepare a TOE chart and create two sketches of the application's user interface.

   Scenario: RM Sales divides its sales territory into four regions: North, South, East, and West. Robert Gonzales, the sales manager, wants an application in which he can enter the current year's sales for each region and the projected increase (expressed as a percentage) in sales for each region. He then wants the application to compute the following year's projected sales for each region. (For example, if Robert enters 10000 as the current sales for the South region, and then enters a 10% projected increase, the application should display 11000 as next year's projected sales.) He also wants to print a report showing the four current year's sales, the four projected sales percentages, and the four projected sales amounts.
   a. Prepare a TOE chart ordered by task.
   b. Rearrange the TOE chart created in step a so that it is ordered by object.
   c. Draw two sketches of the user interface—one using a horizontal arrangement and the other using a vertical arrangement.
3. In this exercise, you will modify an existing application's user interface so that the interface follows the GUI design tips covered in Tutorial 2's Lesson A.
   a. Open the La3 (La3.vbp) project, which is located in the Tut02 folder on your Student Disk.
   b. Save the form and the project as La3Done in the Tut02 folder on your Student Disk.
   c. Lay out and organize the interface so it follows all of the GUI design tips specified in Lesson A. (Refer to the *Layout and Organization of Your Interface* GUI design tips at the end of Lesson A.)
   d. Run the application, then click the Print Time Report button to print the interface. (The Print Time Report button contains the code to print the interface. If the labels in the printout do not appear the same as on the screen, you may need to select a different font for the labels. Exit the application, select the labels, then use the Font property to change the Font; try the Arial font.)
   e. Click the Exit button to end the application. (The Exit button contains the code to end the application.)
   f. Submit the printout from step d.

LESSON B

objectives

In this lesson you will learn how to:

- Build the user interface using your TOE chart and sketch
- Follow the Windows standards regarding the use of graphics, color, and fonts
- Apply the BackStyle, BorderStyle, and Appearance properties
- Add a text box control to a form
- Use the TabIndex property to control the focus
- Lock the controls on the form
- Assign access keys to the controls

Building the User Interface

Preparing to Create the User Interface

In Lesson A you completed the first of the five steps involved in creating an OOED application (plan the application). You are now ready to tackle the second step (build the user interface). You use the TOE chart and sketch you created in the planning step as guides when building the interface, which involves placing the appropriate controls on the form and setting the applicable properties of those controls. Recall that a property controls the appearance and behavior of the objects included in the interface, such as the object's font, size, and so on. Some programmers create the entire interface before setting the properties of each object; other programmers change the properties of each object as it is added to the form. Either way will work, so it's really just a matter of preference.

To save you time, your Student Disk contains a partially completed application for Skate-Away Sales. When you open the application, you will notice that most of the user interface has been created and most of the properties have been set. One control, however, is missing from the form: a text box control. You'll add the missing control later in this lesson. First open the project and save it under a different name so that the original files remain intact in case you want to practice this lesson again.

tips

Another way to make a form the active window is to click its name in the Project window, then click the Project window's View Object button.

To open the partially completed project, then save the files under a new name:

1. Start Visual Basic, if necessary, and make sure your Student Disk is in the appropriate drive.
2. Open the **T2case** (T2case.vbp) file, which is located in the Tut02 folder on your Student Disk. Click the **form's title bar** to make it the active window.
3. Use the **Save T2CASE.FRM As** option on the File menu to save the form as **lbOrder**, then use the Save Project As option on the File menu to save the project as **lbOrder**.

   Figure 2-14 identifies the controls already included in the application. (You won't see the names of the controls on your screen. The names are included in Figure 2-14 for your reference only.)

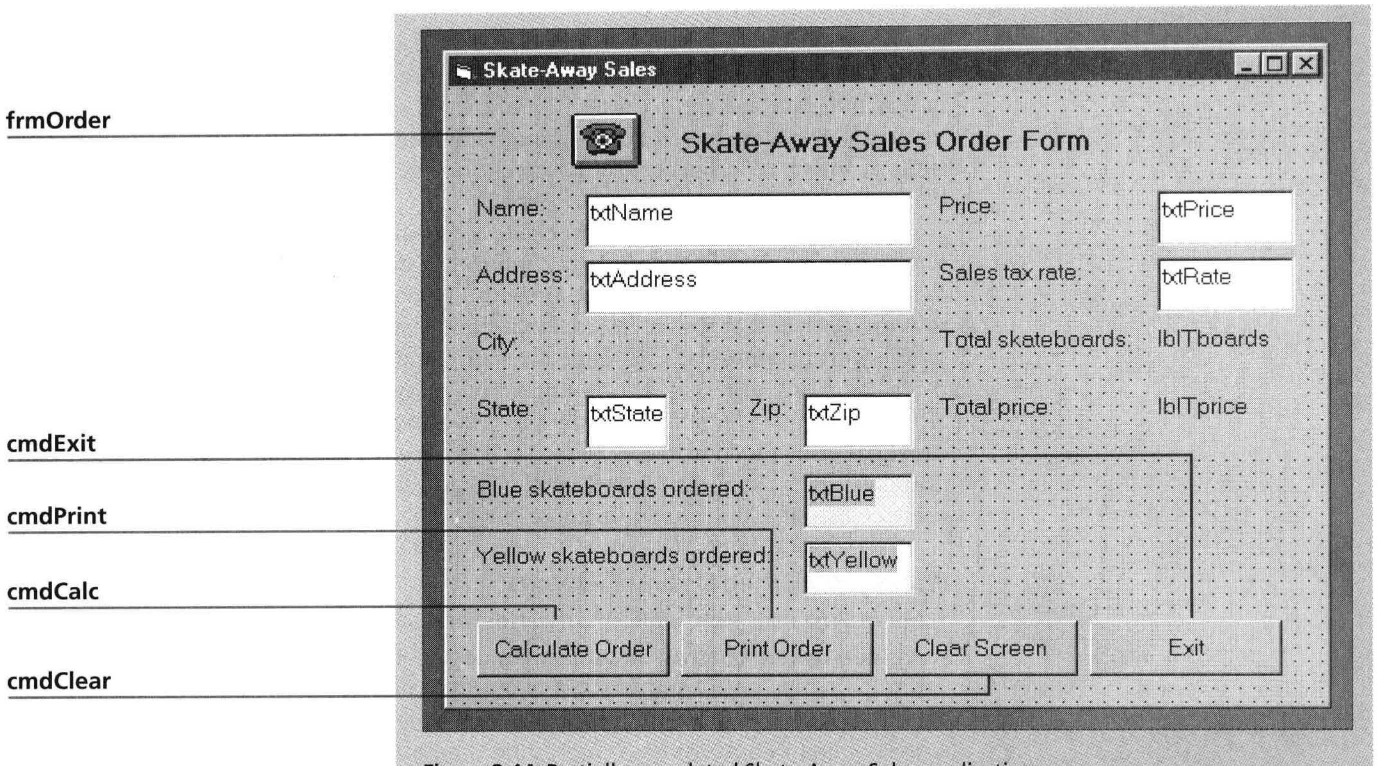


Figure 2-14: Partially completed Skate-Away Sales application

Notice that Figure 2-14 shows only the names of those controls whose names were changed from their default values. Only the form and objects that will contain code or objects that will be referred to in code need to have their names changed to more meaningful ones.

The user interface shown in Figure 2-14 resembles the sketch shown in Lesson A's Figure 2-12. Recall that that sketch was created using the guidelines you learned in Lesson A. For example, the information is arranged vertically, with the most important information located in the upper-left corner of the screen. The command buttons are centered along the bottom of the screen, with the most commonly used button positioned first. The command buttons contain meaningful captions, which are entered using book title capitalization. Each caption appears on one line, and no caption exceeds the three-word limit. The labels identifying the controls other than the command buttons are left-aligned and positioned to the left of their respective controls; each uses sentence capitalization and each ends with a colon.

Notice that the labels and controls are aligned wherever possible to minimize the number of different margins appearing in the user interface. You can use the dots that Visual Basic displays on the form during design time to help you align the various controls in the interface. When positioning the controls, be sure to maintain a consistent margin from the edge of the window; two or three dots is recommended. As illustrated in Figure 2-14, related controls are typically placed on succeeding dots. For example, notice that the top of the txtAddress control is placed on the horizontal line of dots found immediately below the txtName control. Also notice that the left edge of the Print Order command button is placed on the vertical line of dots found to the immediate right of the Calculate Order command button. (Controls that are not part of any logical grouping may be positioned from two to four dots away from other controls.)

Always size the command buttons in the interface relative to each other. When the command buttons are centered on the bottom of the screen, as they are in this interface, all the buttons should be the same height; their widths, however, may vary if necessary. If the command buttons are stacked in either the upper-right or lower-right corner of the screen, on the other hand, all the buttons should be the same height and the same width.

When building the user interface, keep in mind that you want to create a screen that no one notices. Snazzy interfaces may get "oohs" and "aahs" during their initial use, but they become tiresome after a while. The most important point to remember is that the interface should not distract the user from doing his or her work. Unfortunately, it's difficult for many application developers to refrain from using the many different colors, fonts, and graphics available in Visual Basic; actually, using these elements isn't the problem—overusing them is. So that you don't overload your user interfaces with too much color, too many fonts, and too many graphics, the next three sections will give you some guidelines to follow regarding these elements. Consider the graphics first.

GUI Design Tips

Placing and Sizing Design Elements

- Maintain a consistent margin of two or three dots from the edge of the window.
- Position related controls on succeeding dots. Controls that are not part of any logical grouping may be positioned from two to four dots away from other controls.
- Command buttons in the user interface should be sized relative to each other. If the command buttons are centered on the bottom of the screen, then each button should be the same height; their widths, however, may vary. If the command buttons are stacked in either the upper-right or lower-right corner of the screen, then each should be the same height and the same width.
- Try to create a user interface that no one notices.

Including Graphics in the User Interface

The human eye is attracted to pictures before text, so include a graphic only if it is necessary to do so. You can use a graphic, for example, to either emphasize or clarify a portion of the screen. You can also use a graphic for aesthetic purposes, as long as the graphic is small and as long as it is placed in a location that does not distract the user. The small graphic in the Skate-Away Sales interface, for example, is included for aesthetics only. The graphic is purposely located in the upper-left corner of the interface, which is where you want the user's eye to be drawn first anyway. The graphic adds a personal touch to the Skate-Away Sales order form without being distracting to the user.

In the next section you will learn some guidelines pertaining to the use of different fonts in the interface.

GUI Design Tips

Adding Graphics

- The human eye is attracted to pictures before text, so include a graphic only if it is necessary to do so. If the graphic is used solely for aesthetics, use a small graphic and place it in a location that will not distract the user.

Including Different Fonts in the User Interface

As you learned in Tutorial 1, you can change the type of font used to display the text in an object, as well as the style and size of the font. Recall that Courier and MS Sans Serif are examples of font types; regular, bold, and italic are examples of font styles; and 8, 10, and 18 points are examples of font sizes. The default font used for interface elements in Windows is MS Sans Serif 8-point. A point, you may remember, is 1/72 of an inch; so each character in an 8-point font is 1/9 of an inch. You can use either 8, 10, or 12 point fonts for the elements in the user interface, but be sure to limit the number of font sizes used to either one or two. The Skate-Away Sales application uses two font sizes: 10 point and 12 point. The heading at the top of the interface is in 12 point; all of the other labels and captions are in 10 point.

Some fonts are serif, and some are sans serif. A **serif** is a light cross stroke that appears at the top or bottom of a character. The characters in a serif font have the light strokes, whereas the characters in a sans serif font do not. ("Sans" is a French word meaning "without.") Books use serif fonts because those fonts are easier to read on the printed page. Sans serif fonts, on the other hand, are easier to read on the screen, so use a sans serif font for the text in the user interface. You should use only one font type for all of the text in the interface. Avoid using italics and underlining because both make text difficult to read. The Skate-Away Sales interface uses the default MS Sans Serif font.

GUI Design Tips

Selecting Appropriate Font Style and Size

- Use 8, 10, or 12 point fonts for the elements in the user interface.
- Limit the number of font sizes used to either one or two.
- Use a sans serif font for the text in the interface.
- Use only one font type for all of the text in the interface.
- Avoid using italics and underlining because both make text difficult to read.

In addition to overusing graphics and fonts, many application developers make the mistake of using either too much color or too many different colors in the user interface. In the next section you will learn some guidelines pertaining to the use of color.

Including Color in the User Interface

Just as the human eye is attracted to graphics before text, it is also attracted to color before black and white, so use color sparingly. It is a good practice to build the interface using black, white, and gray first, then add color only if you have a good reason to do so. Keep the following four points in mind when deciding whether to include color in the interface:

1. Some users will be working on monochrome monitors.
2. Many people have some form of either color-blindness or color confusion, so they will have trouble distinguishing colors.
3. Color is very subjective; what's a pretty color to you may be hideous to someone else.
4. A color may have a different meaning in a different culture.

It is usually best to follow the Windows standard of using black text on either a white, off-white, or light gray background. The Skate-Away Sales interface, for example, displays black text on a light gray background. If you want to add some color to the interface, you can also use black text on either a pale blue or a pale yellow background. Because dark text on a light background is the easiest to read, never use a dark color for the background or a light color for the text; a dark background is hard on the eyes, and light-colored text can appear blurry.

If you are going to include color in the interface, limit the number of colors to three, not including white, black, and gray. Be sure that the colors you choose complement each other.

Although color can be used to identify an important element in the interface, you should never use it as the only means of identification. In the Skate-Away Sales application, for example, the colors blue and yellow help the salesperson quickly identify where to enter the order for blue skateboards and where to enter the order for yellow skateboards. Notice, however, that color is not the only means of identifying those areas in the interface; the labels to the left of the controls also tell the user where to enter the orders for blue and yellow skateboards.

GUI Design Tips

Selecting Appropriate Colors

- The human eye is attracted to color before black and white. Build the interface using black, white, and gray first, then add color only if you have a good reason to do so.
- Use either white, off-white, or light gray for an application's background, and black for the text. You can also use black text on either a pale blue or a pale yellow background. Dark text on a light background is the easiest to read.
- Never use a dark color for the background or a light color for the text; a dark background is hard on the eyes, and light-colored text can appear blurry.
- Limit the number of colors to three, not including white, black, and gray. The colors you choose should complement each other.
- Never use color as the only means of identification for an element in the user interface.

Now complete the user interface for the Skate-Away Sales application. First see what the interface looks like with a white background instead of a gray one.

tips

Another way to display the Properties window for the form is to right-click an empty area of the form, then select Properties from the popup menu. You can also click the Properties Window button on the Standard toolbar.

To change the background color of the interface:

1 Click the **form's title bar** to select the form. Press the **F4** key to display the Properties window, then set the form's BackColor property to **white**. The form's background color changes to white.

HELP? To change the BackColor property, click BackColor in the Properties window, then click the Settings list arrow. Click the Palette tab, then click the desired color square.

Notice that the background color of the label controls, however, is still gray. You can fix this problem in two ways. You can either set each label control's BackColor property to white, just as you did for the form, or you can set each label control's BackStyle property to 0-Transparent. For reasons you will learn in the next section, setting the BackStyle property is the preferred way.

The BackStyle Property

Recall from Tutorial 1 that the BackColor property determines an object's background color. The default background color for forms and label controls is gray. Because each form and each label control can have its own BackColor setting, changing the background color of the form does not change the background color of the label controls on that form. You can match the color of a label control to the background color of the form by setting the label control's BackStyle property.

A label control's **BackStyle property** determines whether the label is transparent or opaque. The default setting is 1-Opaque, which means that the color value stored in the control's BackColor property fills the control and obscures any color behind it—in this case, the gray in the label controls obscures the white background of the form. When a control's BackStyle property is set to 0-Transparent, Visual Basic ignores the setting in the control's BackColor property; instead, Visual Basic allows you to see through the control. In most cases it is more efficient to change the BackStyle property of the label controls, instead of their BackColor property; then, if you want to experiment with the background color of the form, you won't have to change the BackColor property of the label controls each time to match the form. Observe how this works by changing the BackStyle property for the label controls to 0-Transparent.

To change the BackStyle property:

1 Select all of the label controls except the lblTboards and lblTprice controls. (There are 12 label controls, not counting the lblTboards and lblTprice controls.) Selection handles appear around each selected control.

HELP? Recall from Tutorial 1 that to select more than one control, you click the first control, then Ctrl-click the other controls.

2 Display the Properties window, then set the BackStyle property of the selected controls to **0-Transparent**. (Remember to press the **Enter** key after selecting the 0-Transparent setting.) You can now see the form's white background through the label controls.

3 Click the **form** to deselect the selected label controls. (Be sure to click the form, and not one of the selected controls on the form.)

Suppose you decide that the user interface looked better with a gray background.

4 Display the Properties window, then set the BackColor property of the form to a **light gray**.

5 Click the **form** to select it.

Notice that you can still see the form's background color, which is now gray, through the 12 label controls.

Before you set the BackStyle property for the lblTboards and lblTprice controls, you will change their BorderStyle property and their Appearance property.

The BorderStyle and Appearance Properties

As you learned in Tutorial 1, the BorderStyle property determines the style of the object's border. The **BorderStyle property** for a label control can be either 0-None, which is the default, or 1-Fixed Single. The 1-Fixed Single setting surrounds the label control with a thin line, so it looks similar in appearance to a text box. A control's **Appearance property**, on the other hand, determines if the control appears flat (0-Flat) or three-dimensional (1-3D) on the screen. Although the default setting of a label control's Appearance property is 1-3D, you will not see the three-dimensional effect unless you set the label control's BorderStyle property to 1-Fixed Single. You will change the BorderStyle property of the lblTboards and lblTprice controls to 1-Fixed Single, so that the label controls will have a border. You will then change their Appearance property to 0-Flat because, in Windows applications, controls that contain data that the user is not allowed to edit do not typically appear three-dimensional. The last change you will make to the lblTboards and lblTprice controls is to change their BackStyle property to 0-Transparent.

To change the properties of the lblTboards and lblTprice controls:

1 Select the **lblTboards** and **lblTprice** controls.

HELP? Refer back to Figure 2-14 for the location of these two controls.

2 Display the Properties window. Set the selected controls' BorderStyle property to **1-Fixed Single**, then click the **form** to both deselect the controls and remove the Properties window. Notice that both label controls appear three-dimensional.

3 Now select the **lblTboards** and **lblTprice** controls again and set their Appearance property to **0-Flat**, then set their BackStyle property to **0-Transparent**.

4 Click the **form** to deselect the controls. The two label controls appear as flat boxes on the screen.

Now continue building the interface.

Setting the Text Property

Recall that most of Skate-Away's customers are in Illinois. Instead of having the salesperson enter IL in the txtState control for each order, it would be more efficient to have IL appear, automatically, in the state text box when the application is run. If the user needs to change the state entry while the application is running, he or she can simply click the State text box, then delete the current entry and retype the new one. You can display IL in the txtState control by setting that control's Text property to IL. Do that now.

To set the txtState control's Text property:

1 Click the **txtState** control, then display the Properties window. Click **Text** in the Properties list, then type **IL** and press the **Enter** key.

2 Click the **form** to select it. IL appears in the txtState control.

Notice that the text box control in which the user enters the city is missing from the interface. You will add that control next.

Adding a Text Box Control to the Form

A **text box control** provides an area in the form where the user can enter data. Add the missing text box control to the form and then set its properties.

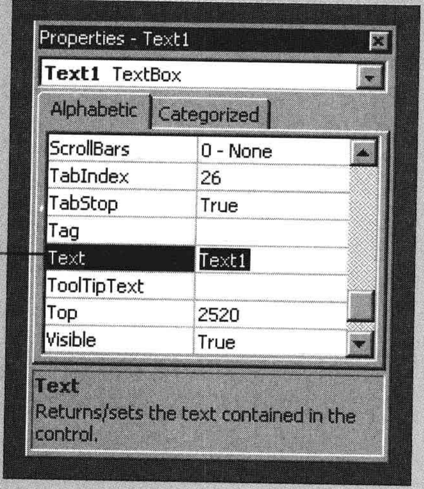
To add the text box control to the form, set its properties, and then save and run the application:

1 Double-click the **Text Box** tool in the toolbox. A default-sized text box control appears in the middle of the form.

The Text1 text you see inside the text box is the current setting of this control's Text property. The **Text property** for a text box is similar to the Caption property for a label control; both properties manage the text shown inside their respective controls. Because you don't want the Text1 text to appear when the application is run, you will delete the contents of this control's Text property.

2 Display the Properties window. Double-click **Text** in the Properties list. Visual Basic highlights the Text1 text in the Settings box, as shown in Figure 2-15.

double-click here to highlight Text1 text



Text1 TextBox

ScrollBars	0 - None
TabIndex	26
TabStop	True
Tag	
Text	Text1
ToolTipText	
Top	2520
Visible	True

Text
Returns/sets the text contained in the control.

Figure 2-15: Text property highlighted in Settings box

HELP? If the Text1 text is not highlighted in the Settings box, double-click the Text property again until the Text1 text is highlighted. You can also drag in the Settings box to highlight the Text1 text.

3 Press the **Delete** key, then press the **Enter** key to remove the highlighted text. The text box is now empty.

4 Set the following two properties for this text box control:

Name:	txtCity ("txt" stands for text box)
Font:	10 points

HELP? Recall that the Name property is listed first on the Alphabetic tab in the Properties window; it's listed in the Misc category on the Categorized tab.

5 Drag the **txtCity** control to a location immediately below the txtAddress control, then drag the txtCity control's right border until the control is as wide as the txtAddress control. The correct location and size are shown in Figure 2-16.

correct size and placement of the txtCity control

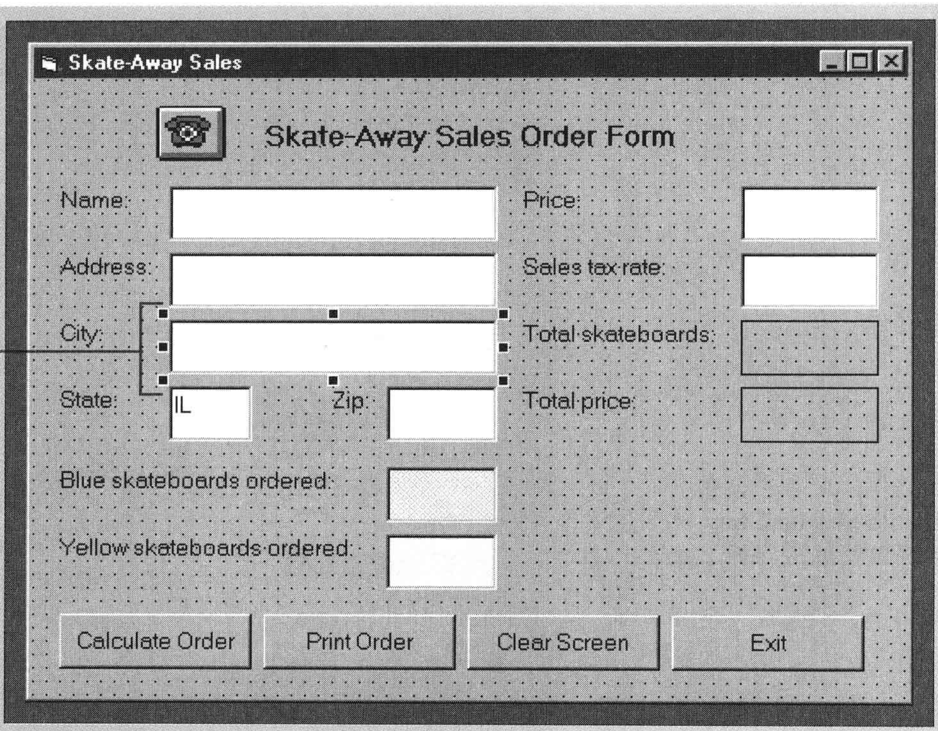


Figure 2-16: Correct size and placement of the txtCity control

Now save and run the project to see if it is working correctly.

6 **Save** and **run** the project. An insertion point appears in the txtName control.

The insertion point indicates that this text box is the active control. In Windows terminology, the txtName control has the **focus**, which means that the control is ready to receive input from you. Try this next.

7 Type **Sport Warehouse** as the customer name. Notice that a text box displays the information it receives from you. The information is recorded in the text box's Text property. In this case, for example, the information is recorded in the Text property of the txtName control.

In Windows applications, you can move the focus from one control to another by pressing the Tab key.

8 Press the **Tab** key to move the focus to the txtAddress control, then type **123 Main.**

The city entry is next.

9 Press the **Tab** key to move the focus to the txtCity control. Notice that the focus skips the txtCity control and moves to the txtState control.

Stop the application and think about why this happened. The `End` statement has already been entered in the Exit button's Click event procedure.

10 Click the **Exit** button to end the application. The application ends, and Visual Basic returns to the design screen.

tips

The easiest way to save a project using its current name is to click the Save Project button on the Standard toolbar. The Save Project button will save any file in the project that has been changed since the last time it was saved.

Controlling the Focus with the TabIndex Property

The **TabIndex property** determines the order in which a control receives the focus when the user is using the Tab key to tab through the application. Not all controls have a TabIndex property; Menu, Timer, CommonDialog, Data, Image, Line, and Shape controls do not. When you add a control that has a TabIndex property to a form, Visual Basic sets the control's TabIndex property to a number representing the order in which that control was added to the form. Keep in mind that when assigning these numbers Visual Basic starts counting at 0 (zero). In other words, the TabIndex property for the first control added to a form is 0 (zero), the TabIndex property for the second control is 1, and so on. Visual Basic uses the TabIndex property to determine the Tab order for the controls. To get a better idea of how this works, check the TabIndex property for the txtAddress, txtState, and txtCity controls.

To view the TabIndex property:

1 Click the **txtAddress** control, then click **TabIndex** in the Properties list. Notice that Visual Basic set the TabIndex property to 13. A TabIndex property of 13 means that this was the fourteenth control added to the form.

Now look at the TabIndex property for the txtState control.

HELP? If the TabIndex property is not 13, you might have selected the Address: label control instead of the txtAddress text box. Click the txtAddress text box, then repeat Step 1.

2 Click the **Object box** list arrow, which is located below the Properties window's title bar, then click **txtState** in the list. Notice that the txtState control's TabIndex property is set to 14. A control with a TabIndex of 14 will receive the focus immediately after the control with a TabIndex of 13. In this case, for example, the txtState control will receive the focus immediately after the txtAddress control.

Now look at the TabIndex property for the txtCity control, which was the last control added to the form.

3 Use the Properties window's Object box to select the **txtCity** control. Notice that the txtCity control's TabIndex property is set to 26, which means it's the 27th control added to the form.

To tell Visual Basic that you want the txtCity control to receive the focus immediately after the txtAddress control, you need to change the TabIndex property of the txtCity control to 14, which is one number greater than the TabIndex of the txtAddress control (13). Figure 2-17 shows the current TabIndex values for the txtAddress, txtCity, and txtState controls and what their TabIndex values should be.

Control Name	Current TabIndex Value	New TabIndex Value
txtAddress	13	13
txtCity	26	14
txtState	14	15

Figure 2-17: TabIndex values

The only TabIndex value you will need to change is the one for the txtCity control. Changing the txtCity control's TabIndex property to 14 will cause Visual Basic to renumber the controls on the form, beginning with the control that originally had a TabIndex of 14. For example, the TabIndex of the txtState control, which was originally 14, will be assigned a TabIndex of 15. The control that had a TabIndex of 15 will be assigned a TabIndex of 16, and so on. After you change the TabIndex for the txtCity control, you will verify the renumbering by selecting the txtState control and looking at its TabIndex value.

To change the txtCity control's TabIndex property, then save and run the project:

1 The TabIndex property for the txtCity control should already be selected in the Properties window, so just type **14** and press the **Enter** key.

Verify that the txtState control's TabIndex property is now 15.

2 Use the Properties window's Object box to select the **txtState** control. The value of its TabIndex property should be 15.

3 **Save** and **run** the project.

4 Type **Sport Warehouse** as the customer name, then press the **Tab** key to move the focus to the txtAddress control.

5 Type **123 Main** as the address, then press the **Tab** key. Notice that the focus now moves to the txtCity control.

You can check the focus order of the remaining controls by simply pressing the Tab key.

6 Press the **Tab** key. The focus moves from the txtCity control to the txtState control, which is correct.

7 Press the **Tab** key, slowly, ten times. The focus correctly moves to the following controls: txtZip, txtBlue, txtYellow, txtPrice, txtRate, cmdCalc, cmdPrint, cmdClear, cmdExit, and finally back to the txtName control.

In Windows applications, the Tab key moves the focus forward, and the Shift+Tab key combination moves the focus backward. Try the Shift + Tab key combination.

8 Press **Shift + Tab**. The focus moves to the Exit button, as indicated by its highlighted border and the dotted rectangle around its caption.

9 Because the Exit button has the focus, you can simply press the **Enter** key to end the application. The application ends and Visual Basic returns to the design screen.

tips

You can also move a locked control by selecting it in the form, and then pressing and holding down the Ctrl key as you press the up, down, right, or left arrow key on the keyboard.

Now that all of the controls are on the user interface, you can lock them in place.

Locking the Controls on a Form

Once you have placed all of the controls in the desired locations on the form, it is a good idea to lock the controls in their current positions so you don't inadvertently move them. Once the controls are locked, you will not be able to move them until you unlock them; you can, however, delete them.