Chapter 13

Multiple Document Interface and Advanced Graphics

GOALS

After working through this chapter, you will be able to:

- Use option buttons and check boxes in programs.
- Implement the Multiple Document Interface in programs that call for such an approach.
- Program graphing algorithms for various functions.
- Use the MouseDown, MouseUp, and MouseMove events.
- Use the **PSet**, **Line**, and **Circle** methods to draw various shapes and figures.
- Use static and global variables as flags and indicators.

OVERVIEW

This chapter looks at three types of tools:

- *Controls*
  - *Option button*
  - *Check box*
- *Mouse events*
- *Multiple Document Interface*
  - *Parent forms*
  - *Child forms*

The Truth in Lending program in the first section lets the user calculate the annual percentage rate of a loan given the monthly payment, the original principle of the loan, and the number of years over which the loan is to be repaid. As you build the program, you will use option buttons to indicate the format of the display and the number of payments made per year.

The Multiple Document Interface (MDI) is described in the second section. MDI creates a parent form as a container for all the child forms of a project. When a child form becomes active, its menu bar replaces that of the

parent form. This Multiple Document Interface is used in many Windows applications. You can use it to make your programs have the same "look and feel" as other Windows programs.

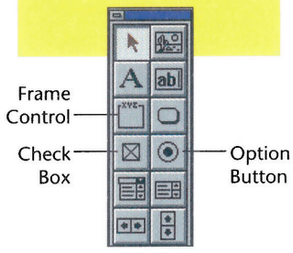
The Drawing project described in the third section lets the user customize a drawing utility. In this program you let the user express preferences, by giving him or her more choices than in any program you have written so far. You will learn how to provide those choices and write code that accommodates them. The program also illustrates the Mouse events recognized by forms and picture boxes. Mouse events occur when a mouse button is pressed or released or when the mouse is moved. Each of these events generates an event. Finally, the Drawing project provides a framework for additional work with graphics commands (fourth section).

Option Buttons, Check Boxes, and Frames

Figure 13-1 shows the positions of the frame control, check box, and option button in the Toolbox. You can use these controls to simplify the user interface, making it easier to use and understand. A frame groups related controls. With check and option boxes, you can let the user interact with a program with clicks instead of by adding text.

You will add option buttons to the Truth in Lending program. This program calculates the annual percentage rate of a loan, given information about the payments and duration of the loan. Whether you are thinking of a car loan or a college loan, you can see what the loan costs using this program. Knowing the true cost of a loan (including interest) helps you choose the right loan. In this program, a user chooses the number of payments per year and the format of the output by clicking on option buttons.

Figure 13-1
Position of controls in Toolbox



Looking at the Objects

Typically, you group options buttons and check boxes with frames. The frame serves to group the controls together, so the user will understand they stand as a group.

OPTION BUTTONS AND CHECK BOXES

Figure 13-2 shows two frames, each containing two option buttons. The two payments option buttons are related to each other, as are the two format option buttons. Related option buttons must appear in the same frame or on the same form.

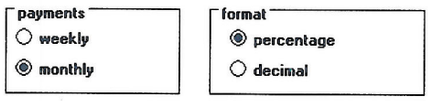


Figure 13-2
Two sets of related option buttons

Using the payments group, a user determines the number of payments made in a year. Using the format group, a user selects the display format of the program's output. The output of the Truth in Lending program is a percentage. The user may choose to view the percentage as a decimal with four decimal places, or as a percentage with two decimal places.

To select an option button, a user clicks on it. As you can see from the example, option buttons represent nonoverlapping choices. Payments are made either weekly or monthly, not both. Output is displayed as a percentage or decimal, not both. Users can select only a single option button from a group of related option buttons. If a user clicks on one button, then on another, the first button is turned off. This is an excellent reason to group option buttons with frames, so that it's clear which buttons are related to each other. You will sometimes find it convenient to make a control array out of all the option buttons in a group.

Option buttons within the same frame are grouped together. Clicking an option button in the payments group will not affect the buttons in the format group. When a button is clicked, the value of its Value property becomes **True**, and the values of all related buttons (buttons within the same frame), become **False**.

Option buttons are also known as radio buttons. This name is easy to remember because there is a similarity between option buttons and the buttons on your car radio. You can only select a single station on your radio. When you press a button to select a station, the previously pressed button is unpressed.

Check boxes are also set up in related groups, with related check boxes appearing in the same frame or on the same form. You use check boxes to provide a list of parameters, more than one of which can be set on at the same time. For instance, you may have a frame with check boxes designating the style for the text in a document. A user may select any number of style elements, or none at all. The text may be bold and underlined, one or the other, or neither. See Figure 13-3. When a user clicks on a check box, any other selected check boxes remain selected.

PUTTING THE CONTROLS IN A FRAME

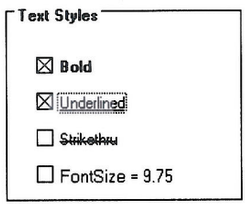


Figure 13-3
Selecting check boxes

To put option buttons into a frame, you must first place the frame on the form. Do not start by double-clicking on the option button icon or the check box icon. After one of these controls is placed directly on a form, you cannot move it into a frame.

Click on the frame icon, move the mouse pointer to where you want the upper-left corner of the frame, then press and drag to the frame's lower-right corner. As soon as you release the mouse button, the frame appears.

To change the heading, edit the Caption property of the frame. To place option buttons in the frame, click the option button icon in the Toolbox. Click and drag to position the option button within the frame. As soon as you release the mouse button, the option button appears.

Change the label for the option button by changing the Caption property of the control. Using the Name property, give the option button an appropriate name that begins with an "opt" prefix, such as optWeekly or optMonthly.

An important property of option buttons and check boxes is the Value property. This property is a toggle with two possible values: True or False. A value of True indicates that the button or check box is selected. In code, you can refer to the option button or check box by name, then set values within the procedures on the basis of whether the Value property of the control is set to **True** or **False**.

Option button values are read by the program to determine the user's preferences. In addition, the option button controls themselves generate event procedures. In the Truth in Lending program, for example, whenever any of the options is changed, the cmdCalc_Click() event is executed. Clicking the option button sets its Value to **True**. This event is attached to a command button that calculates all the output of the program.

```
Sub optDecimal_Click ()
   Call cmdCalc_Click
End Sub
```

Truth in Lending Program

The goal of the program is to calculate the annual percentage rate of a loan along with the total finance charge that would be paid by the time the loan is paid off. To make these calculations, the program must prompt the user for three pieces of necessary information: the size of the monthly payment, the original amount of the loan, and the duration of the loan.

As you have done in other programs, you use labeled textboxes for this purpose (Figure 13-4). Also as you have done in other programs, you should provide default values in these textboxes. In addition, default values should be entered for the option buttons. Then, the user only needs to click on the Calculate button to see the program work.

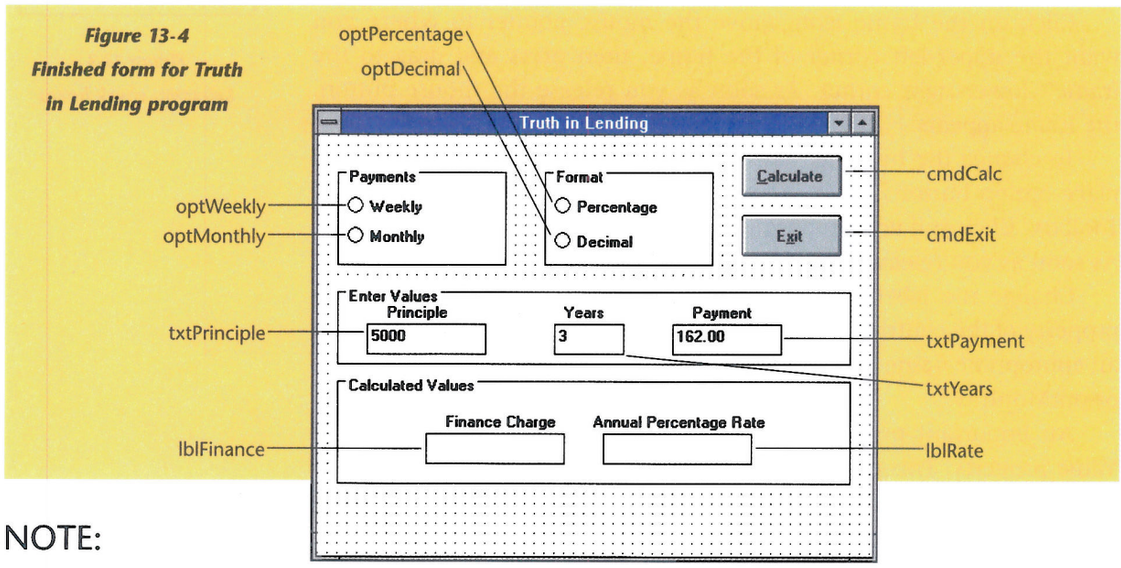


Figure 13-4
Finished form for Truth in Lending program

NOTE:

The organization of controls on the form is not set in stone. You need to follow Windows conventions, such as putting command buttons on the right or bottom, but you also need to use your judgment. Think of yourself as the user. What would be the easiest layout to understand? Change the layout shown in Figure 13-4 if you can think of a better arrangement.

Another piece of the puzzle is the number of payments made on the loan. The duration of the loan in years is part of the answer, but you also need another piece of information:

*# Payments = number of years * payments per year*

What are the advantages of using a set of option buttons to find out the number of payments per year? You could use a textbox, but users will appreciate just having to click on a choice. Also, with option buttons, you know exactly what the possible responses are as the program runs. You do not have to be concerned with having the program interpret responses such as "4x/mo" from a textbox.

The second set of option buttons on the form cover the format of the final output. This user input is not necessary to perform the program's calculations; instead, the program uses it to provide the data in the desired format.

If the user clicks any of the option buttons, the option button Click event handler will call the cmdCalc_Click routine, which calculates new values for the finance charge and the annual percentage rate (APR).

This formula is used to calculate the total finance charge:

Finance charge = monthly payment # payments - principal*

SETTING UP THE FORM

Set up the Truth in Lending form using Figure 13-4 as a guide.

CODING THE CALCULATE BUTTON

The cmdCalc_click() routine starts by collecting values from the three textboxes: textPrincipal, txtYears, and txtPayment. To determine the number of payments per year, the program reads the Value property assigned to one of the option buttons from the Payments frame. You can use either button to set up the **If-Then** statement: if one is true, the other must be false.

The condition portion of the **If-Then** statement may seem strange to you:

```
If optWeekly Then
   PayPerYear = 52
```

The program treats the name of the button (optWeekly) as either a **True** or **False** value. If the button is selected, optWeekly is **True**; if it is not selected, optWeekly is **False**. The name of the button, then, represents the button's Value property at the time the Calculate button is clicked.

After the number of payments has been determined, the appropriate value, 12 or 52, is assigned to the *PayPerYear* variable. This program uses this value to calculate the total finance charge and the annual percentage rate (APR). You will be using this statement to calculate the APR:

```
APR = 2 * PayPerYear * Finance / (Principal * (NumberPay + 1))
```

This statement is based on the following formula.
APR = 2 * # of payments per yr. * total finance charge / loan amount * (# of payments + 1)

As a final step after the annual percentage has been calculated, the option buttons in the Format frame are read to determine the display format.

To write this code:

1 Double-click on the Calculate button to open the Code window and enter this code between the first and last lines of the subroutine:

```
'—Local declarations
Dim NumberPay, Years, Principal
Dim PayPerYear, Payment, Finance, APR
```

continued

```
'—Read the textboxes
Principal = Val(txtPrincipal.Text)
Years = Val(txtYears.Text)
Payment = Val(txtPayment.Text)
'—Check the option button
If optWeekly Then   'equivalent to: If optWeekly = True
   PayPerYear = 52
Else
   PayPerYear = 12
End If
'—Calculate the number of payments
NumberPay = PayPerYear * Years
'—Calculate and display the total finance charge
Finance = Payment * NumberPay - Principal
lblFinance = Format(Finance, "currency")
'—Calculate annual percentage rate, APR
APR = 2 * PayPerYear * Finance / (Principal * (NumberPay + 1))
'—Check the Format frame buttons
If optDecimal Then
   lblRate = Format(APR, "0.00###")
Else
   lblRate = Format(APR, "Percent")
End If
```

2. Save the form and project files.
3. Run the program. Click on the Calculate button and note the results.
4. Click on the Weekly option button. Click on Calculate and note the results.
5. Experiment with different values for the principle, the years, and the payment.

QUESTIONS AND ACTIVITIES

1. Set up the frame and option buttons necessary to let a user choose between temperature measured in Fahrenheit and Celsius. Set the Caption, Name, and Value properties.
2. What happens to an option button in a frame when an option button on the form is clicked? To find out, put a third option button onto the form created for problem 1 above. Use a message box to display the values of the all three option buttons (Fahrenheit, Celsius, and the third button) when the value of the third button changes.

3. Create a frame with properly labeled check boxes for choosing type style. Provide choices of bold, underline, italic, strikethrough, superscript, and subscript. Give appropriate names to the check boxes. Use the Value properties to set Bold and Underline to **True**.
4. Modify the program code and the form of the Truth in Lending program to include a display label for the total amount paid back over the life of the loan. To calculate this amount, multiply the total number of payments by the payment entered by the user. Display the result, in currency format.
5. What are the differences and similarities between option buttons and check boxes?
6. What is the significance of placing two or more option buttons within a frame?
7. Option buttons and check boxes have properties and generate events. What are two significant properties of these controls and why are they significant? What is one significant event of each, and why is it significant?
8. Given a monthly payment of $401.24, a four-year loan, and an initial loan amount of $14,560, use the Truth in Lending program to calculate the total finance charge for the loan.

Multiple Forms Within a Form

In the Truth in Lending program, you used frames to group option buttons. In the exercises following the program, you used frames to group related check boxes. In both cases, frames worked well as containers for controls. Sometimes, however, you need a container that is bigger than a frame. You need a container that can contain forms. In these cases, you can use a special kind of a form called an MDI form as a container for other forms.

In Microsoft Word or another word processing application, you have probably opened more than one file at a time. By using the Window menu, you can move between one file and another. You can also move material from one file to another by cutting and pasting. Each of the documents is held in its own form. See Figure 13-5.

When you open Microsoft Word, a document is already loaded called **doc1.doc**. You can use this document to create a file, or you can open a previously saved file. Have you noticed that the Word menu bar

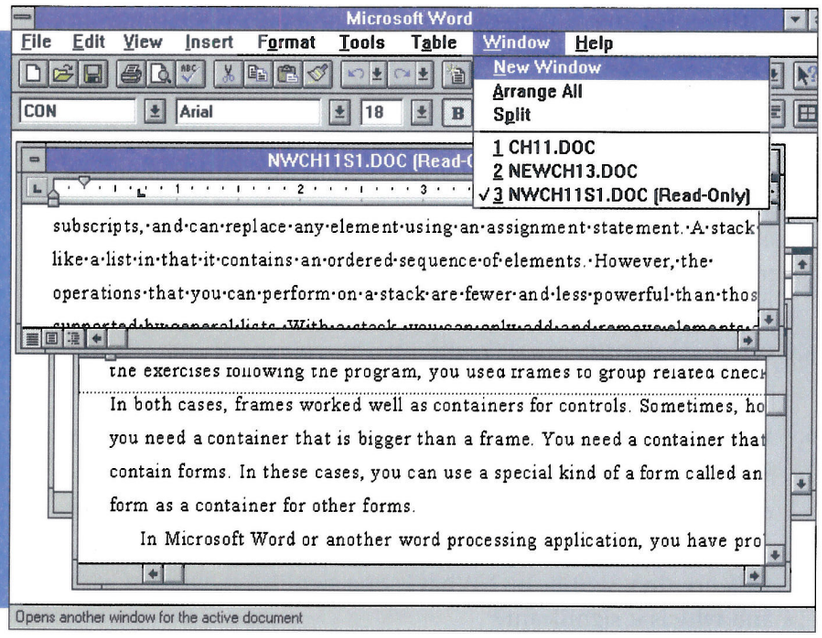


Figure 13-5
Microsoft Word with multiple documents and Window menu displayed

looks different when one or more documents is open, compared to when all documents are closed? Figure 13-6 shows the menu bar with a document open. Figure 13-7 shows the menu bar with all active documents closed.

Figure 13-6
Word menu bar with an open document

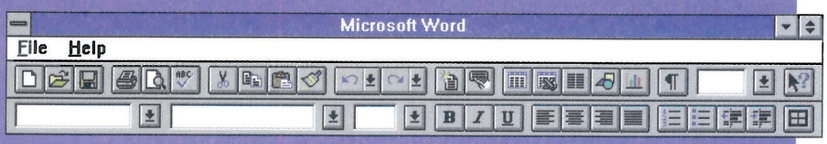


Figure 13-7
Word menu bar with all documents closed

Each of these Word documents exists in its own form.

When you have closed all the documents, there is still a form open. This is the parent form. Any document you open is placed in a child form. The child forms are identical copies of each other (except, of course, that the documents placed in them are probably different).

When a child form is created and loaded, its menu bar replaces the menu bar of the parent form. When that document is closed, the child form is unloaded and the menu bar of the parent form becomes active

again. Typically, choices in the menu bar of the parent form activate various child forms. A child form may include a menu command that executes the code to *unload* the child and return control to the parent form. Several child forms may be open at one time within a parent form.

Working with MDI: Stage 1

In this first stage, you create an MDI parent form and experiment with what kinds of controls can be placed on the form. You will find a number of restrictions that don't apply to normal forms. In addition, you turn the default form into a child form and experiment with maximizing and minimizing the child form within the parent while the program is running.

Experiment with the Multiple Document Interface by following these steps:

1. Start Visual Basic. If Visual Basic is running, select New Project from the File menu.
2. Choose New MDI Form from the File menu to open a form on the screen. The default name and caption of this form is MDIform1. See Figure 13-8. This is a parent form.
3. Open the Project window from the Window menu. See Figure 13-9.
   Notice that both the icon and the name of the MDI form is different from that of a non-MDI form.

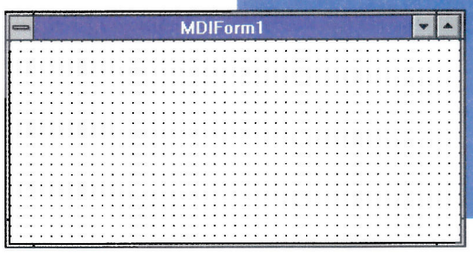


Figure 13-8
Newly created MDI form

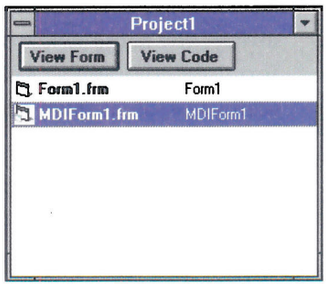
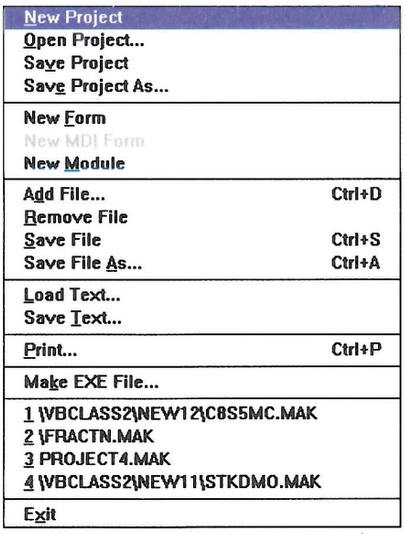


Figure 13-9
Project window displaying new MDI form



4. Open the File menu again. As you can see, the command New MDI form is dimmed. An application can have only a single MDI parent form, although it may have many child forms. See Figure 13-10.

Figure 13-10
File menu with command dimmed

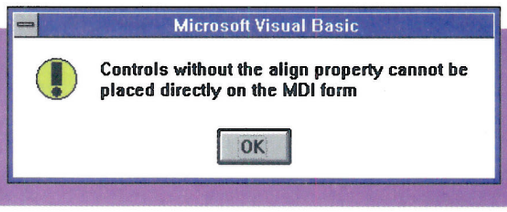


Figure 13-11
Experimenting with adding controls to the parent form

5 Select the MDI parent form, then double-click on the textbox icon in the Toolbox. What happens? See Figure 13-11.
As you can see, you cannot place any control that lacks the Align property on a MDI form. That includes, for example, textboxes, labels, and command buttons (though you can create menus for these forms).

6 Double-click on the picture box icon in the Toolbox to place a picture box on the form. See Figure 13-12.
The only control that you can place on a MDI form is a picture box. The toolbars you see in many Windows applications are actually picture boxes that stretch from one side of the form to the other.

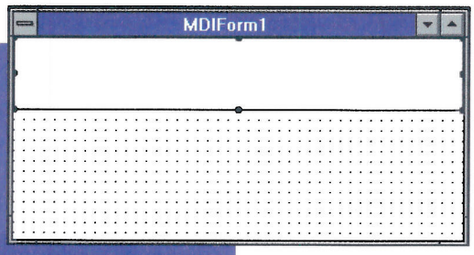


Figure 13-12
Placing a picture box on a MDI form

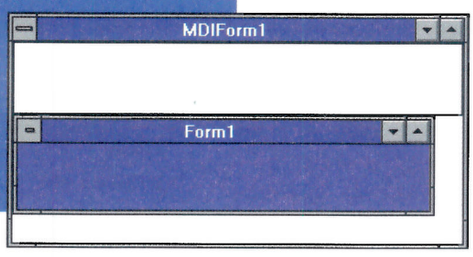
7 Try to resize the picture box by selecting the box and dragging the handles. You can adjust the box's height, but not its width. The box is always the same width as the parent form. The part of the parent form not covered by a picture box is called the client area. All child forms you create will be scaled to fit in this area.

8 Turn the default form (form1.frm) into a child form. Make that form active, open its Properties window, and toggle the MDIChild property of the form to **True**. During design time, you will not notice any change resulting from the toggle.

9 Change the BackColor property of Form1.

10 Run the program from the toolbar and note the results. See Figure 13-13. What happened? You design each child form independently of the parent form. Both parent and child forms can have menus, and child forms may have menus that are different from each other. Whenever a child form is loaded, its menu bar, if it has one, replaces the menu bar of the parent form.

Figure 13-13
MDI parent and child form in run mode



11 Click on the maximize button in the upper-right corner of the child form.

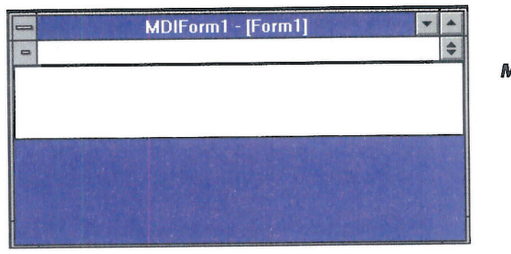


Figure 13-14
Maximized child form

12 When a child form is maximized, its name is added to the name of the MDI parent in the caption. See Figure 13-14.
Note that the area occupied by the picture box has not been covered by the child form.

13 Restore the child form by clicking on the restore box in the upper-right corner of the form.

14 Minimize the child form by clicking the minimize button.

15 Note the results. The icon representing the form is contained in the parent form. See Figure 13-15.

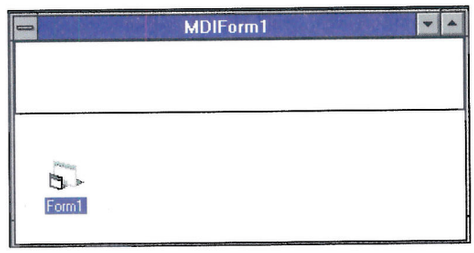


Figure 13-15
Minimized child form represented by an icon in the parent form

Working with MDI: Stage 2

In this stage, you create additional child forms, giving each a distinctive caption. With the program running you'll activate one form after another and experiment with minimizing and maximizing the forms.

To continue your work with MDI:

1 Change the name of the MDI parent form to **USA**.

2 Change the name of form1 to **Illinois**.

3 Add another form to the project. Set the MDIChild property to **True**. Change the caption to **Texas**.

4 Add another form to the project. Set the MDIChild property to **True**. Change the caption to **California**.

5 Add another form to the project. Set the MDIChild property to **True**. Change the caption to **New York**.

6 Select the parent form. Delete the picture box.

7 Double-click on the Illinois form to open the Code window. Enter the form_Click procedure and enter the following line of code:

```
form2.Show
```

8 Double-click on the Texas form to enter its Code window. Enter the form_Click procedure and enter the following line of code:

```
form3.Show
```

9 Alter the California form similarly.

10 Run the program. Click on the Illinois form. Click on the Texas form. Click on the California form. See Figure 13-16.

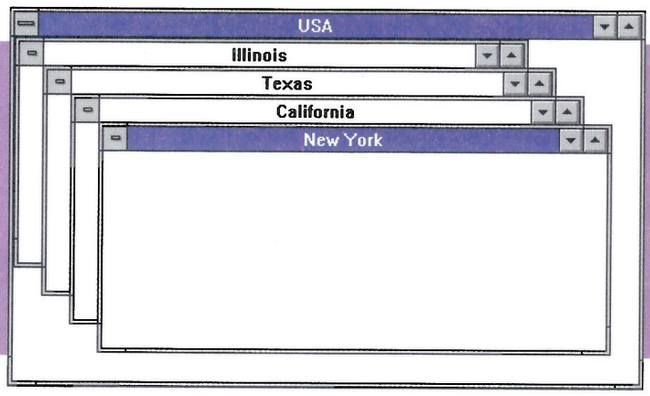


Figure 13-16
Multiple child forms open in a parent form

11 Minimize each of the forms. See Figure 13-17.

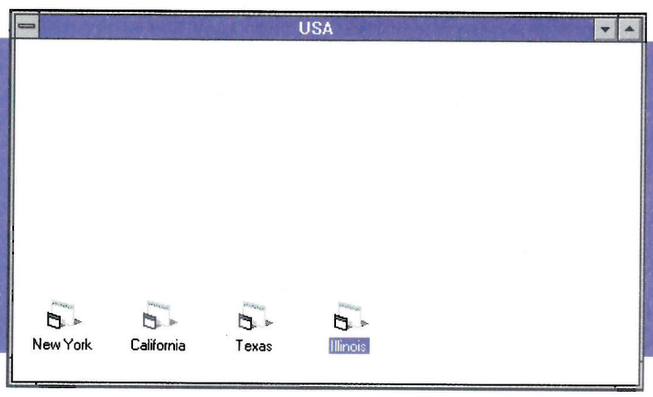


Figure 13-17
Minimized child forms

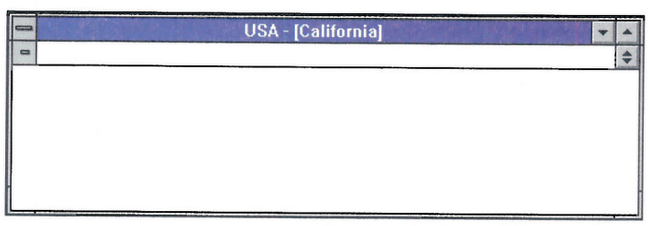


Figure 13-18
Maximized child form

12 Click on the California form and maximize the form. The caption of the child form joins the caption of the parent. See Figure 13-18.

13 Close the project without saving.

QUESTIONS

1. Describe the relationship between the parent form and the child forms in the Multiple Document Interface.
2. What are some of the significant differences between a MDI parent form and a regular form?
3. How are picture boxes used in conjunction with a MDI form?
4. What is the client area in a parent form?

3

Section

The Drawing Project

Visual Basic is a very rich language. There are so many features it's hard to know where to stop. This section looks again at mouse events. A drawing application that allows the user to draw lines, rectangles, triangles, and circles on a form using mouse events and graphics methods is presented in this section; the finished form is shown in Figure 13-19.

If you've used the Paintbrush program, you may have noted some of its limitations. It's hard to accurately position circles in a drawing, for example. There is no freestyle drawing mode. To create triangles, you have to carefully align the third side to connect to the other two sides. You don't have the option of using a grid to place objects accurately. The drawing application you create in this section addresses several of these deficiencies. In addition, you can add or alter features to meet your particular needs.

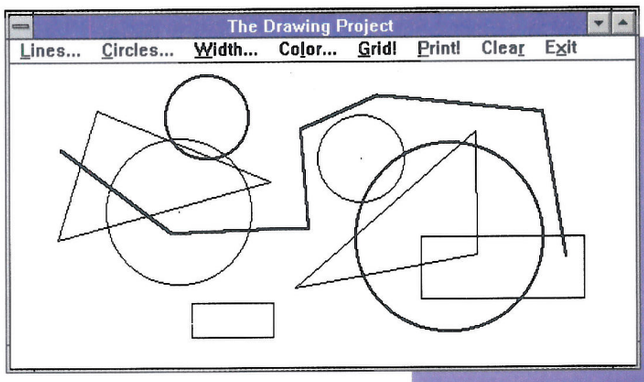


Figure 13-19
Finished form for the drawing project

While writing the program, you'll be reviewing a number of events generated by the mouse. You'll use the **Line** and **Circle** methods introduced in a previous chapter. And, you'll use variables as flags indicating the current state of the drawing. As a result, you'll be better able to use mouse events and graphics routines in your programs.

More on Mouse Events

Forms respond to many events. The Form_Load event, for example, is executed whenever the form is activated. As a result, it is a good place to put start-up code for an application. For this Drawing program, four events generated by mouse clicks are particularly important. These are:

- MouseDown
- MouseUp
- MouseMove
- Click

Think about how you work in other drawing programs. Often, for example, you select a shape, such as a rectangle, with a mouse, then click and drag in the window to draw the rectangle. In that process, you generated all four of the events listed above. These events control the drawing on the form.

For example, in many drawing programs, you see dotted lines outlining shapes you are drawing; when you release the mouse, the shapes appear. What makes a shape appear at that point? It appears because the designer of the program wrote the appropriate code for the MouseUp event.

MOUSEDOWN EVENT

The MouseDown event is generated when a user depresses the mouse button. Clicking the mouse button is the rapid depression and release of the button. The first part of a click, when the button is pressed, is called pressing or depressing the mouse button. The event procedure has four parameters:

```
Sub Form_MouseDown (Button As Integer, Shift As
Integer, x As Single, y As Single)
```

The *Button* parameter indicates which mouse button is pressed (your mouse may have two or three buttons). The *Shift* parameter indicates whether the user pressed the Alt, Control, or Shift keys in combination with pressing the mouse button.

The *X* and *Y* parameters represent the coordinates of the mouse pointer at the time the user pressed the mouse button. The units in which these coordinates are reported depend on which coordinate system has been selected. In this Drawing program, you will leave the default coordinate system, the twip system, unchanged. The ScaleHeight and ScaleWidth properties of the form hold the dimensions of the form in twips.

CLICK EVENT

In addition to the MouseDown and MouseUp events, depressing and then releasing a mouse button with the pointer over the same control or form generates another event: a Click event. The MouseDown and MouseUp events correspond to individual physical actions. By contrast, the Click event is a higher-level abstraction, a logical event that Visual Basic detects and informs your program about by calling a click handler. The Click event of a form or control occurs when the user presses a mouse button with the cursor over that object, and then releases the same button with the cursor over that same object. In between the button press and release, the user may move the mouse outside the object; a Click event will still occur.

When the mouse button is clicked, the status of the Alt, Control, and Shift keys, and the position of the mouse pointer, are not available with the Click or the DblClick procedures. If you need to know the location where the mouse button is released when you process a Click event, you should handle the MouseUp event by saving the coordinates of the mouse cursor, and then use these values in the Click or DblClick event, which will occur *after* the MouseUp event.

MOUSEUP AND MOUSEMOVE EVENTS

The MouseUp event and the MouseMove event are both sent the same parameters as the MouseDown event: *Button*, *Shift*, *X*, and *Y*. The MouseUp event is generated when the mouse button is released. A MouseMove event is generated when the coordinates of the mouse pointer change.

Starting Out

You may have experimented with drawing programs before. If so, what did you think of them? Could you do everything you wanted to do? Chances are, you found something that didn't seem quite right to you. It is almost impossible to design a drawing tool that everyone will think is perfect. This section explains how to create the rudiments of a new drawing tool. You will then have the opportunity to customize and enhance the tool to fit your specific preferences.

Like many of the programs covered in this book, the Drawing project is deliberately left unfinished. It runs and it is useful as it stands; yet it suffers from the same serious limitation of other drawing programs you may have tried. It has not been customized for your purposes.

Before you start customizing, start with the basics. Which shapes will the user be able to draw with this program? Start with the obvious ones:

- Single lines, drawn from point to point
- Continuous lines, drawn from point to point to point
- Freestyle lines, lines drawn to follow the mouse movement as long as the button is depressed
- Triangles, drawn between three points
- Rectangles, sides parallel to the form, drawn from one corner to another
- Circles, drawn by clicking the center and a point on the circle
- Circles, drawn by clicking the endpoints of a diameter of the circle

Besides drawing shapes, users will want to perform functions such as modifying their drawings and so forth. What functionality are you going to provide? There is much you could add; for now, stay with these:

- Changing the line width
- Changing the color of the line drawn
- Printing the form
- Providing a grid on the form to aid alignment
- Clearing the form

Program development starts with setting goals for the program. In this case, the goals listed above were designed to remedy some of the limitations of other drawing programs, specifically Paintbrush. Once the goals are set, think through the techniques you'll use to actually write the program: the variables needed, the graphic methods used, how you'll handle differentiating between the three points of a triangle.

Much of the program development is determined by the user interface you design. Once the menu structure is determined, a lot of other program elements fall into place. You will develop the program in this order:

- Define the user interface (the menu)
- Declare and initialize the variables
- Code the routines that control the drawing of circles
- Code the line drawing routines
- Code the grid, colors, and line widths
- Code the mouse event routines

Creating the Menu

The program needs only a single form, so most of the work on the look-and-feel of the program goes into the menu.

The menu itself is arranged to meet the goals for the program discussed above. You may have goals of your own for the project. If so, this is a good place to decide how you will implement those goals. The first step in designing the menu structure is to identify components that go together. There are two kinds of circles—they should share a single menu. There are three types of lines, as well as two shapes made of lines: the triangle and the rectangle. They could all share the same menu or they could be split into two menus: a line menu and a shape menu.

Another decision to make concerns building menu control arrays. Can two or more menu items be grouped together in the same event procedure? Only closely related actions should share the same procedure.

As you look over the menu structure given below, see if you agree with the design decisions that have been made. Change any items that seem wrong. Add new items to make the program more useful.

To create the menu for the Drawing project:

1. Create a new directory for the Drawing project.
2. Start Visual Basic. If Visual Basic is running, click New Project from the File menu.
3. Change the name of the default form to **frmDrawing**. Change the caption of the form to **The Drawing Project**.
4. Open the Menu Design window (Figure 13-20), then use the chart below to create the menu for the Drawing project.

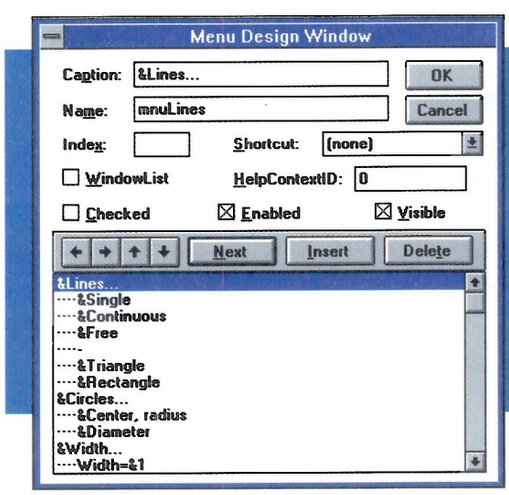
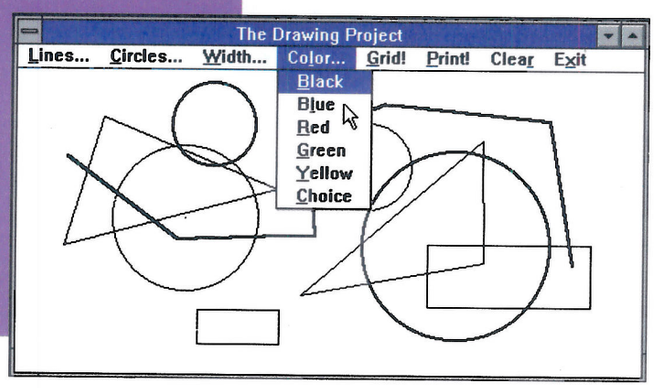


Figure 13-20
Menu Design window for the Drawing project

Menu Item	*Event Name*	*Description*
&Lines...	mnuLines	initializes True/False variables
&Single	cmdLines()	sets True/False
&Continuous	cmdLines()	sets True/False
&Free	cmdLines()	sets True/False
&Triangle	cmdTriangle	sets True/False
&Rectangle	cmdRectangle	sets True/False
&Circles...	mnuCircle	initializes True/False variables
&Center, radius	cmdCircle()	sets True/False
&Diameter	cmdCircle()	sets True/False
&Width...	mnuWidth	no function
Width=&1	cmdWidth()	set DrawWidth to 1
Width=&2	cmdWidth()	set DrawWidth to 2
Width=&3	cmdWidth()	set DrawWidth to 3
Width=&5	cmdWidth()	set DrawWidth to 5
&Choice	cmdWidth()	user enters DrawWidth
Co&lor	mnuColor	no function
&Black	cmdColor()	sets Colr=QBColor(0)
B&lue	cmdColor()	Colr=QBColor(1)
&Red	cmdColor()	Colr=QBColor(4)
&Green	cmdColor()	Colr=QBColor(2)
&Yellow	cmdColor()	Colr=QBColor(14)
&Choice	cmdColor()	user enters color number
&Grid!	cmdGrid	sets grid of points
&Print!	cmdPrint	prints form
Clea&r	cmdClear	clears form
E&xit	cmdExit	exits program

Figure 13-25
Color menu for the Drawing project



5 When the menu is complete, close the Menu Design window. The Color menu is shown in Figure 13-21.

Visual Basic lets you reposition and resize the form to accommodate the size of your drawing.

Working with the Global Variables

Before you start writing code for the project, you should determine the variables you are going to need. If any of those variables are going to be used by more than one event procedure, you should make them global variables by declaring them in the general declarations section of the form.

Table 13-1 suggests which variables you should declare as global and briefly explains their use. Except for the variable *Colr*, each one is used as a **True/False** (or Boolean) variable indicating the status of some aspect of the program. Some of these variables indicate whether the user has chosen a particular option such as the continuous line style. The rest are used by the program to keep track of when the user is in the process of drawing a figure using the mouse. The *Colr* variable is used to represent the value returned by the **QBColor()** function in the routine that sets the drawing color.

Table 13-1. Variables for the Drawing Project

Name	Use
Sing	Set to True when the "single" line option is selected.
Cntin	Set to True when the "continuous" line option is selected.
Fre	Set to True when the "freestyle" line drawing tool is selected.
Drwing	Set to True when a Figure is being drawn.
Center	Set to True when a circle specified by clicking the center and a point on the circle is being drawn.
Diameter	Set to True when a circle specified by clicking the endpoints of a diameter is being drawn.
Triangle	Set to True when the triangle option is selected.
Rectangle	Set to True when the rectangle option is selected.
Colr	Set to the color number.

Declare these variables in the general declarations section of the form:

```
Option Explicit
Dim Sing As Integer, Cntin As Integer, Fre As Integer
Dim Colr As Long, Drwing As Integer
Dim Center As Integer, Diameter As Integer
Dim Triangle As Integer, Rectangle As Integer
```

Setting the Boolean Variables

When the user clicks on Lines or Circles on the menu bar, a drop-down menu appears. The Lines menu contains various lines and polygons supported by the program. The Circles menu contains choices for the two circles drawn by the program.

Each of the event procedures called by these menu choices sets one or more of the global variables to True, indicating that, for instance, a single line is to be drawn. The drawing actually occurs in the MouseDown event handler.

Each command of the Lines and Circles submenus sets one of these **True/False** variables. The main menu command itself—&Lines, &Circles—also generates a Click event when that item is clicked. In previous programs, this event has been ignored for menu commands that have submenus. In this program, it is used to initialize the **True/False** variables to **False**.

To set the Boolean variables:

1. Select frmDrawing from the Project window.
2. Click on View Code.
3. Enter these lines in the mnuCircle() procedure:

```
'—Turn off each kind of circle
Center = False
Diameter = False
mnuLines_Click  ' Turn off all the lines
```

4. Enter these lines in the mnuLines() procedure:

```
'—An itemized list of all the True/False variables
Drwing = False     ' not drawing
Sing = False     ' single lines
Cntin = False     ' continuous lines
Fre = False     ' freestyle lines
Triangle = False     ' triangles
Rectangle = False     ' rectangles
Center = False     ' circles from center and point
Diameter = False     ' circles from diameters
```

Coding the Circles Menu

Selecting Circles from the menu bar lets the user draw circles by choosing a center and a point on the circle's circumference, or by selecting

two endpoints of the circle's diameter. Both commands on the menu call the same event procedures, making them part of a menu control array. The first command generates an index of 1 and the second, an index of 2. You use the **Select Case** statement to choose between the two. If a user selects the "center" type circle, the Circle variable is set to **True**. If a user chooses a "diameter" type circle instead, the Diameter variable is set to **True**. These values control the drawing of each figure.

To write the necessary code:

1. Select frmDrawing from the Project window.
2. Click on View Code.
3. Enter these lines in the cmdCircle_Click() procedure.

```
Select Case index
   Case 1     ' center to radius type circle
      center = True
   Case 2     ' diameter type circle
      diameter = True
End Select
```

Coding the Lines Menu

By selecting Lines from the menu bar, a user is presented with a choice between single lines, continuous lines, freestyle lines, triangles, and rectangles. Each of these call the same event procedures.

To code the menu:

1. Select frmDrawing from the Project window.
2. Click on View Code.
3. Enter these lines in the cmdLines_Click() procedure.

```
Select Case index
   Case 1     ' single lines
      Sing = True
   Case 2     ' continuous
      Cntin = True
   Case 3     ' free form
      Fre = True
End Select
```

Adding the Miscellaneous Routines

Follow these steps to code some miscellaneous routines.

1. Select frmDrawing from the Project window.
2. Click on View Code.
3. Type **End** in the cmdExit_Click() procedure.
4. Type **Cls** to clear the screen in the cmdClear_Click() procedure.
5. Type **PrintForm**, a command to print the active form, in cmdPrint_Click().
6. Enter the following line in cmdTriangle_Click().

```
Triangle = True
```

7. Enter the following line in cmdRectangle_Click().

```
Rectangle = True
```

Adding the Grid Routine

This routine has nothing to do with the grid control in the Visual Basic toolbox. This item calls a procedure to plot a grid of fine dots on the drawing area to help the user line up graphic elements. Using nested loops controlled by the values of the ScaleHeight and ScaleWidth properties of the form, it spaces small points every 500 twips from top to bottom and from side to side.

A MEMORY JOG

A twip is the measure used in the default coordinates system to measure graphic elements. A twip is 1/20th of a printer's point. It is the standard of all Windows graphics routines.

ScaleHeight is the height of the form expressed in twips. ScaleWidth is the width of the form expressed in twips.

The only challenge in the Grid procedure is saving the value of the DrawWidth property. The current value of the DrawWidth property is read and saved in a local variable, *d*. The DrawWidth is set to 1, to plot an inconspicuous point. When the loops have finished, the original value of DrawWidth is restored.

To code this routine:

1. Select frmDrawing from the Project window.

2 Click on View Code.

3 Enter these lines in cmdGrid_Click.

```
Dim x, y, d     ' local variables for the loops
d = DrawWidth     ' current value of DrawWidth
DrawWidth = 1     ' tiny point
For x = 0 To ScaleWidth Step 500 ' every 500 twips
   For y = 0 To ScaleHeight Step 500
      PSet (x, y), 0  ' color is black
   Next y
Next x
DrawWidth = d     ' restore the DrawWidth
```

Adding the Color Routine

When users select Color from the menu bar, they see a small selection of colors. The event procedure called by these menu items uses **QBColor()** to provide color information for the *Colr* variable. You will use this variable every time you use the **Line**, **PSet,** and **Circle** method to set the color of the lines and dots drawn.

Users can choose a new color at any point while they are drawing. The color may be changed in the middle of a "freestyle" line session or a "continuous" line sequence. The last item in the menu lets the user choose a color with which to draw, still using the **QBColor** function.

To add this routine:

1 Select frmDrawing from the Project window.

2 Click on View Code.

3 Enter these lines in cmdColor_Click.

```
Select Case Index
   Case 1
      Colr = QBColor(0)      ' black
   Case 2
      Colr = QBColor(1)      ' blue
   Case 3
      Colr = QBColor(4)      ' red
   Case 4
      Colr = QBColor(2)     ' green
   Case 5
      Colr = QBColor(14)     ' yellow
   Case 6
      Colr = QBColor(Val(InputBox("Enter color #<0-15>:", , "1")))
End Select
```

Adding the Line Width Routine

This routine lets the user choose a DrawWidth of 1,2,3, or 5 with a click of the mouse. A fifth item lets the user enter a value for the DrawWidth property of the form from the keyboard.

To add the routine:

1. Select frmDrawing from the Project window.
2. Click on View Code.
3. Enter these lines in cmdWidth_Click.

```
Select Case Index
   Case 1
      DrawWidth = 1
   Case 2
      DrawWidth = 2
   Case 3
      DrawWidth = 3
   Case 4
      DrawWidth = 5
   Case 5
      DrawWidth = Val(InputBox("Enter the width:", , "1"))
End Select
```

Drawing a Single Line

The purpose of this interlude is to try out the drawing program. As you go through the steps required to draw a line, the text describes what is going on "under the hood". The use of the Boolean variables is traced to show how the program knows what to do each step of the way.

To draw a single line on the form:

1. Click on the Lines menu command. Result: All **True/False** variables are set to false. This means no figure has yet been selected and drawing has not yet begun.
2. Click on the Single command. Result: the **True/False** variable *Sing* is set to **True**. A line is being drawn.
3. Click the mouse on the form. Result: The *Drwing* variable is checked. It is **False** (set in step 1), so the line drawing has just begun. *Drwing* is set to **True** and the first point of the line is PSet.
4. Click the mouse a second time. Result: *Drwing* is **True** (set in step three), so this is the second click. A line is drawn from the first point to the current position of the mouse pointer. Because single lines are being drawn, *Drwing* is reset to **False**.

Coding the MouseDown Event Procedure

The menu command handlers that have already been defined set Boolean variables which indicate what kind of figure is being drawn. The MouseDown event procedure actually handles drawing dots on the screen. The procedure uses the Boolean variables to determine what graphics methods to use to draw the figure.

1. Select frmDrawing from the Project window.
2. Click on View Code.
3. Enter these lines in the Form_MouseDown() procedure of the drawing form. These lines declare local variables. Static variables retain their values from call to call. x1 and y1 keep track of previously clicked positions.

```
Static x1 As Single, y1 As Single
'-P keeps track of point number
Static p
```

4. Enter the lines to handle single-line drawing as described above. Continue after the lines entered in step 3.

```
'-Single line handler
If Sing Then     ' single lines
   If Not Drwing Then
      Drwing = True     ' beginning of line
      PSet (x, y), Colr     ' initial point
   Else     ' second point of line
      Line -(x, y), Colr     ' draw to clicked spot
      Drwing = False     ' end of line
   End If
End If
```

5. Continuous lines are much like single lines. *Drwing* is not reset to **False** when the second point of the line is clicked. This allows line after line to be drawn. Enter these lines in the MouseDown event.

```
If Cntin Then     ' continuous lines
   If Not Drwing Then
      Drwing = True     ' start a set of lines
      PSet (x, y), Colr     ' at the clicked point
   Else
      Line -(x, y), Colr     ' draw to clicked spot
   End If
End If
```

6 Freestyle lines are drawn with the right mouse button. Hold the button and drag the mouse around the form. At the first mouse click, the first point is plotted and the *Drwing* variable is set to **True**. The rest of the drawing takes place in the MouseMove event to be presented later. Enter these lines in the MouseDown event.

```
'—Freestyle lines, use the right mouse button
If Fre And Button = 2 Then
   Drwing = True
   PSet (x, y), Colr
End If
```

7 To draw a triangle on the form, you need to differentiate between each of three clicks—one for each vertex of the triangle. In addition you need to store the coordinates of the first point clicked. You need those coordinates to draw the third side of the triangle. Enter these lines in the MouseDown event:

```
'—Triangle
If Triangle Then
   '—P keeps track of what point is being drawn
   If p = 0 Then     ' first point of triangle
      PSet (x, y), Colr
      x1 = x: y1 = y     ' save coordinates to close triangle
      p = 1              ' set p to next point
   ElseIf p = 1 Then     ' if second point,
      Line -(x, y), Colr     ' draw line from first to second
      p = 2     ' set p to next point
   ElseIf p = 2 Then     ' if third point
      Line -(x, y), Colr     ' draw line from second to third
      Line -(x1, y1), Colr     ' draw line from third to first
      p = 0              ' reset p to first point
   End If
End If
```

8 Rectangles drawn by this program are specified by clicking on two opposite corners of the rectangle. Rectangles drawn from these points have sides parallel to the sides of the form. The coordinates of the first point are saved in the variables *x1* and *y1*. The *x* and *y* values of this coordinate are used to generate the appropriate **Line** methods. The diagram below shows how the coordinates of the first click, saved in *x1* and *y1*, are used along with the coordinates of the second mouse click to draw the sides of the rectangle. See Figure 13-22.

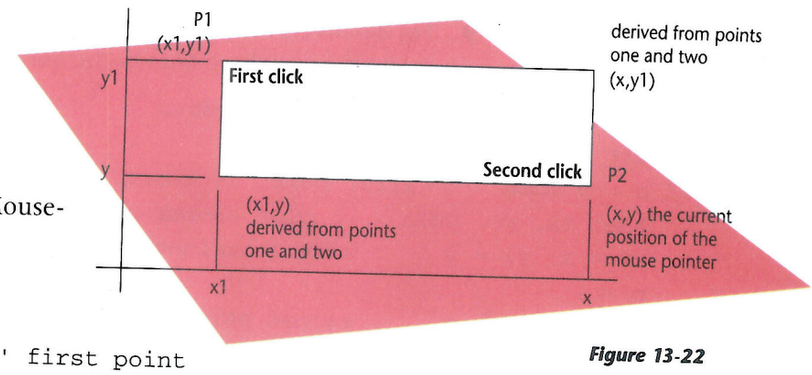


Figure 13-22
Using coordinates to draw a rectangle

Enter these lines in the Mouse-Down event.

```
'—Rectangle
If Rectangle Then
   If p = 0 Then     ' first point
      PSet (x, y), Colr
      x1 = x: y1 = y     ' save coordinate
      p = 1
   ElseIf p = 1 Then     ' diagonal point
      Line -(x1, y), Colr     ' draw four lines
      Line -(x, y), Colr
      Line -(x, y1), Colr
      Line -(x1, y1), Colr
      p = 0     ' back to first point
   End If
End If
```

THE CIRCLE METHOD

The **Circle** method has the syntax:

Circle *(x, y), r, color*

Where *(x,y)* is the center of the circle, *r* is the radius, and *color* is the color.

To draw a circle, use the first mouse click to determine the center of the circle, and the second to determine the radius. Plot a point at the center of the circle and save the coordinates in the variables *x1* and *y1*. Use the value of *p* to determine which point is being clicked: the center, or a point on the circle.

Once the second point is clicked, calculate the radius. The coordinates of the first point are *(x1,y1)*. The coordinates of the second point are the current values of the position of the mouse pointer, given by *X* and *Y*. A distance function is added to the general section of the form, to take care of the calculation. The length of the radius is calculated by using the distance formula and the coordinates of the two points.

9 For details on the **Circle** method, see the textbox. To code the method, enter these lines in the MouseDown event.

```
'—Circle with click on center and radius
If Center Then
   If p = 0 Then     ' first point is center
      PSet (x, y), Colr     ' plot center point
      x1 = x: y1 = y     ' save coordinates
      p = 1
   ElseIf p = 1 Then     ' second point is at the end of a radius
      '—Calculate the distance (the radius) and draw the circle.
      Circle (x1, y1), distance(x, y, x1, y1), Colr
      p = 0
   End If
End If
```

10 The second kind of circle is drawn by specifying the endpoints of the diameter of the circle. From these coordinates, the midpoint is found. The midpoint of the diameter is the center of the circle. The distance formula is called to find the radius.
Enter these final lines in the MouseDown event.

```
'—Draw circle with diameter
If diameter Then
   If p = 0 Then
      PSet (x, y), Colr   ' plot first point
      x1 = x: y1 = y      ' save coordinate
      p = 1
   ElseIf p = 1 Then
      '—Calculate midpoint of diameter, which is the center
      '—Calculate distance from the midpoint to the edge; the radius
      Circle ((x1 + x) / 2, (y1 + y) / 2), distance(x, y, x1, y1) / 2, Colr
      p = 0
   End If
End If
```

Coding the MouseMove and MouseUp Event Procedures

The MouseMove event continuously reports the position of the mouse pointer as the pointer is moved. This continuous reporting makes it the perfect routine to draw the "freestyle" or continuous lines. While the mouse is moving the MouseMove events updates the position of the pointer. The update occurs so frequently, and the lines drawn are so

short, the result appears to be a continuous line that follows the position of the mouse pointer.

The MouseUp event terminates most drawing events. When the mouse button is released, the MouseUp event is fired. This resets the *Drwing* variable to **False**, which stops the drawing of any figure.

1 Enter these lines in the MouseMove event procedure:

```
'—If freestyle lines are being drawn, and
'Drwing is True, continue to draw lines from
'the last point plotted to the current position
'of the mouse pointer.
If Fre And Drwing Then
   Line -(x, y), Colr
End If
```

2 Enter this line in the MouseUp event procedure:

```
If Fre Then Drwing = False
```

Finishing Up

You are almost finished. Take these final steps:

1 The distance function is used to calculate the radius for use with the **Circle** method. Enter these lines in the general section of the form.

```
Function Distance (x1 As Single, y1 As Single, x2 As Single, y2 As
Single) As Single
   '—The distance function returns a single value,
   'the calculated distance between (x1,y1) and (x2,y2).
   Distance = Sqr((x1 - x2) ^ 2 + (y1 - y2) ^ 2)
End Function
```

2 Save the form and project files in the directory created for the project.

3 Run the program and try out each feature. See Figure 13-23.

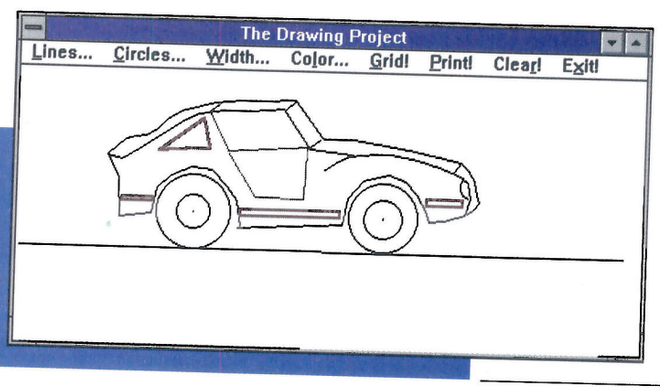
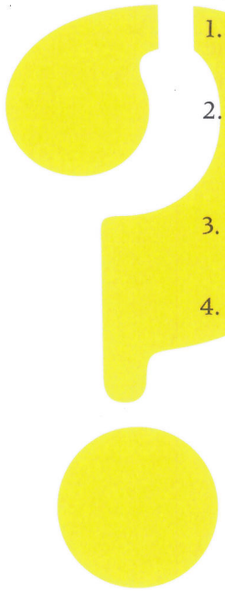


Figure 13-23
Drawing with the Drawing program

QUESTIONS AND ACTIVITIES



1. What are the four parameters sent to a MouseDown event? What is the purpose of each one?
2. Using the Visual Basic Help system, look up the Form object. What are the names and functions of two Form events that are not mentioned in this section?
3. Describe the differences and similarities of the Form_Click and the Form_MouseDown event procedures.
4. To explore the differences and similarities between the Form_Click event and the Form_MouseUp events, follow these steps:
   a) Create a new project. Add the following code to the project's form:
   Form_Click event: `MsgBox "Click!"`
   Form_MouseUp event: `MsgBox "MouseUp!"`
   b) Press the mouse button while the cursor is on the form (notice: there is no Click event), drag the mouse outside the form, and release the button. Again notice: the MouseUp event fired, but no Click event occurred.
   c) Press the mouse button while the cursor is outside the form, drag the cursor onto the form, and release. A MouseUp event occurred, but no Click event.
   d) Press the mouse button inside the form, drag the mouse outside the form, then back inside it. Release the button. A MouseUp event occurred (press Enter to close the message box), then a Click event.
5. Rewrite the Color event procedure to use the **RGB** function instead of the **QBColor** function.
6. Write the statements to add an additional case to the Color event procedure so that a user could choose a random color. To generate a random color, generate a random integer in the range 0 to 15 and use that value in the **QBColor** function. Remember, the **Rnd** function in Visual Basic returns a random number in the range from 0 to 1. Multiply by 15 and use the **Int()** function to convert the number to an integer.
7. The **Line** method can be used to draw a rectangle. Replace the four **Line** statements in the rectangle section of the MouseDown event with a single **Line** statement. The *B* parameter draws a box.

8. Add a command to the Lines menu that draws a filled box of the current color. (Hint: add the *BF* parameter to the **Line** method.)
9. Add an erase function. Use the **Line** method, with the *BF* parameter and a color read from the BackColor property to draw a small, fixed size rectangle.

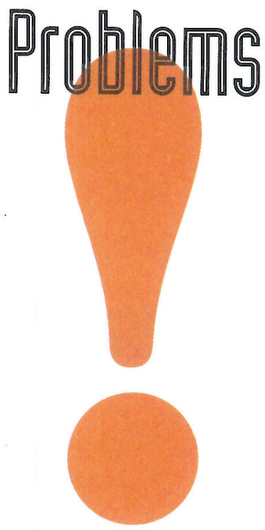
The frame is used to group logically related collections of option buttons or check boxes. In a group of option buttons, only one can be selected at a time. In contrast, more than one check box in a logically related group can be checked at one time.

You can gain access to the contents of these controls through their Value property. This property is **True** if the control is selected.

The Multiple Document Interface is characterized by the presence of a single parent form that becomes the container for one or more child forms. The menu bar of the child form replaces the menu bar of the parent form when the child form becomes active.

An MDI parent form is created by selecting New MDI Form from the File menu. A MDI child form is created by changing the MDIChild property of a normal form to **True**.

You cannot place any control on a MDI form that does not have the Align property. Controls you cannot use include textboxes, labels, and command buttons. The picture box does have the Align property; therefore, you use it to contain the graphic toolbar of programs such as Visual Basic.



1. The Car Option Program

Write a program that allows a car buyer to specify options and features to be included in a car. Start by sketching a form including frames with option buttons and check boxes to choose features. When all the options and features are chosen, use a multiline textbox to display a summary of the choices made.

Provide a form that gives the following choices (you decide whether to use option buttons or check boxes):

- Four door
- Two door
- Station wagon
- Sport utility
- Colors: bold blue, sea green, hot pink
- Wheel size: 14" or 15"
- Interior: bucket seats, bench seats, leather, vinyl, AM-FM, AM-FM CD player, short wave, trip computer, satellite positioning system, built-in phone, fax
- Engine: 6 cylinder, 8 cylinder, 12 cylinder

2. Versatile Grapher Display of Coefficients

Modify the graph procedure of one or more of the forms in the Versatile Grapher to display the values of *a*, *b*, *c*, and *d*, in marked labels near the bottom of the graph.

3. Change the Scale

Modify the graph procedure of one or more of the forms in the Versatile Grapher to allow the user to set the scale by entering the coordinates of the upper-left-hand corner and of the lower-right-hand corner. For an extra challenge, put the following new choices into the menu of each child form: Zoom Out, Zoom In. These choices will automatically change the scale, the **For-Next** statement, and redisplay the graph.

4. Determining the Features of a Computer System

Write a program using option buttons to let a computer buyer choose features to include on a computer. Do some research and find prices and descriptions for alternate choices for monitors, hard drives, CD ROMs, sound cards, processors, memory, and so forth. Put together a form allowing the user to choose what options to include. When the choices have been made, display a list showing the options chosen and the final cost of the machine.

5. The Stock Chart

Write a program with a picture box and an array that allows the user to plot the performance of stock over a period of time. Prompt the user to enter ten closing values for a stock. Display a graph reflecting those closing prices. Choose the scale so the daily differences show up well. How does the scale chosen affect the possible interpretation of the fluctuations of the graph?

6. Tic-Tac-Toe

Write a program that will allow two players to play a game of tic-tac-toe. Use a picture box with some lines to display the playing field. Use the MouseDown event to determine in which box the mouse was clicked, then fill that box with an X or an O, depending on whose turn it is.

7. The Mouse Trail Program

Use the MouseDown event along with the **PSet** command to make a simple drawing program. When the mouse is down, the mouse leaves a trail. When the mouse is up, no mark is left.

8. Saving the Drawings

Modify the graph program to save and recall the drawing. The **Point(x,y)** method returns the RGB color of the coordinate *(x,y)*. Use the ScaleWidth and ScaleHeight properties to determine the dimensions of the form. Save the dimensions of the form, the ScaleWidth and ScaleHeight, and each of the RGB values read from the form.

```
If Index = 7 Then
Datal.RecordSource
= "SELECT * FROM
Animals ORDER BY
CommonName"
Datal.
Refresh
```