Learning Microsoft Visual Basic 4.0


Instructions


How Visual Basic Works


Creating an Application


Writing Event-Driven Programs


Working with Forms and Controls


Adding Menus


Debugging Your Application


Accessing Databases


Introduction to OLE Automation Objects


Using Color and Graphics

How to use the Learning Visual Basic Tutorial

Use the following buttons to navigate through the tutorial:

To go forward in a lesson, click the Next button.

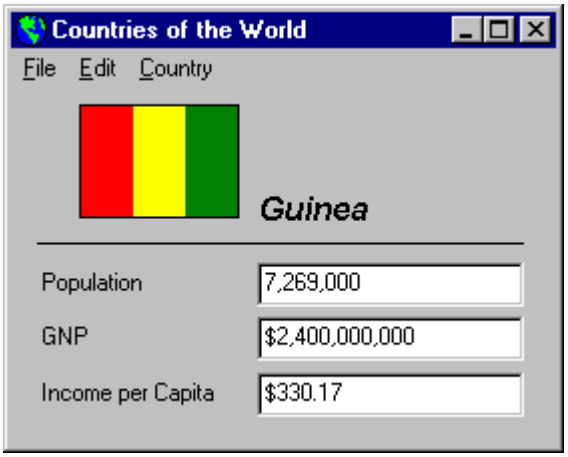
To go backward in a lesson, click the Back button.

To return to the main menu, click the Contents button.

How Visual Basic Works

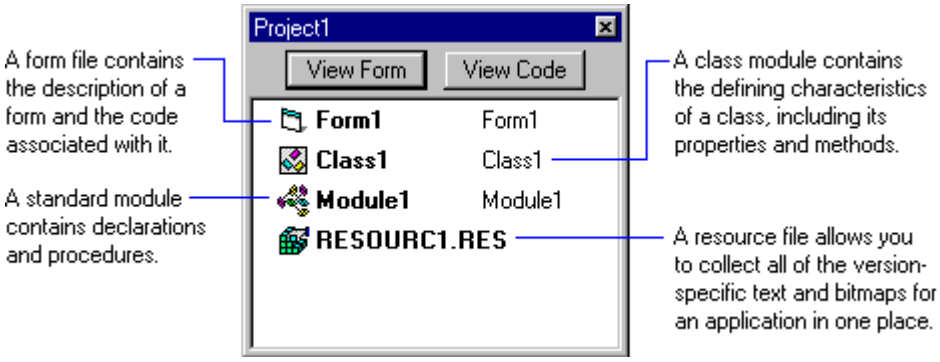
Visual Basic provides many tools that you can use to design graphical applications. This lesson will introduce:

- Projects
- Forms and Controls
- Modules
- The Visual Basic language
- Menu bars
- The Color palette



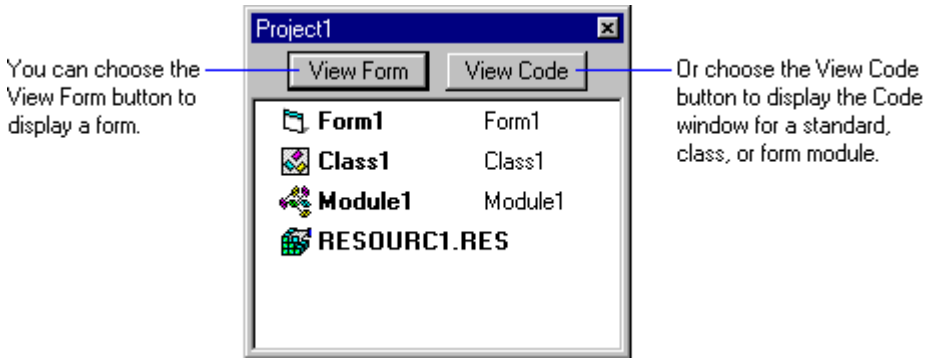
How Visual Basic Works

A **project** is a collection of the form modules, standard modules, class modules, and resource file that make up an application. The project window lists all the files in an application.



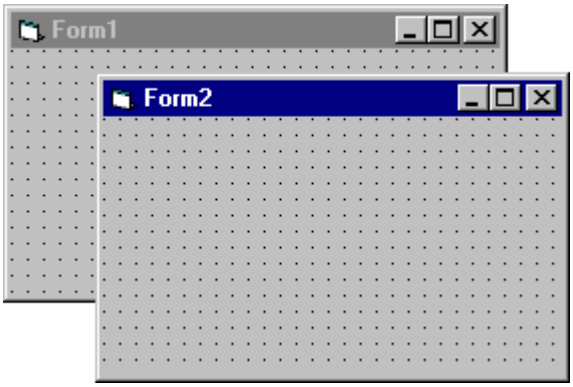
How Visual Basic Works

A **form** includes the controls and code associated with that form. You can share code throughout an entire project by putting the code in a form module or standard module and declaring the procedure **Public**.



How Visual Basic Works

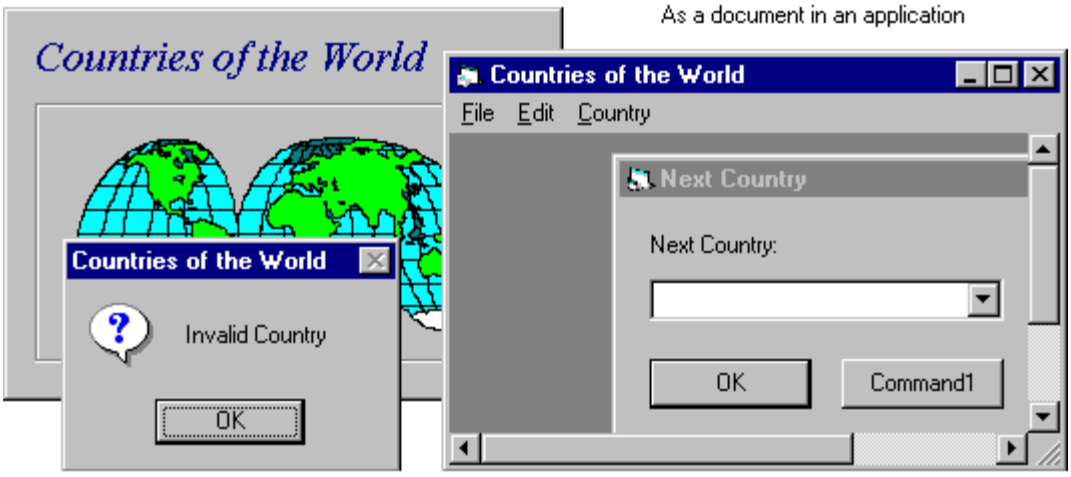
You create forms to serve as the interface of your application. Each form is a window that displays controls, graphics, or other forms.



How Visual Basic Works

You can use forms in many different ways:

As an illustrated introduction screen to an application

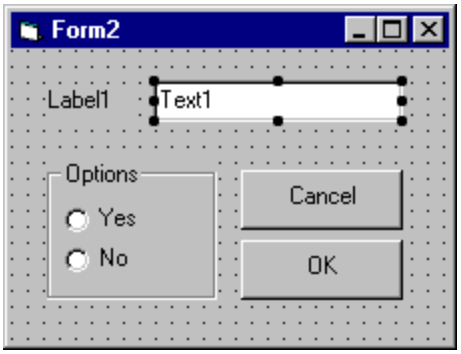


Or, as a dialog box

How Visual Basic Works

Controls are tools such as boxes, buttons, and labels you draw on a form to get input or to display output. They also add visual appeal to your forms.

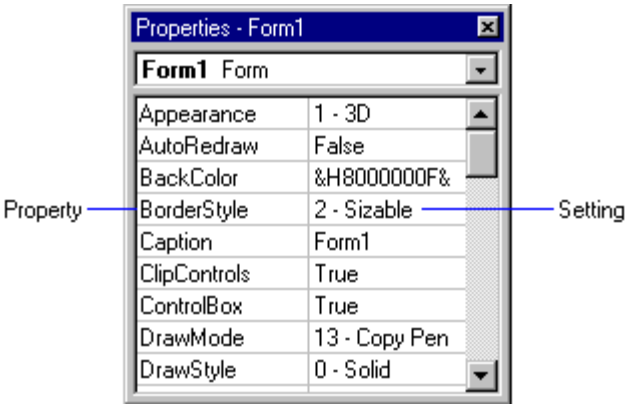
You use the Toolbox to draw controls on a form.



How Visual Basic Works

You set the properties of forms and controls by using the Properties window. Properties specify the initial values for such characteristics as size, name, and position.

The Properties window lists all the properties and their settings for the currently selected module or control.



How Visual Basic Works

In applications with many commands, Visual Basic lets you group your commands on a menu bar.

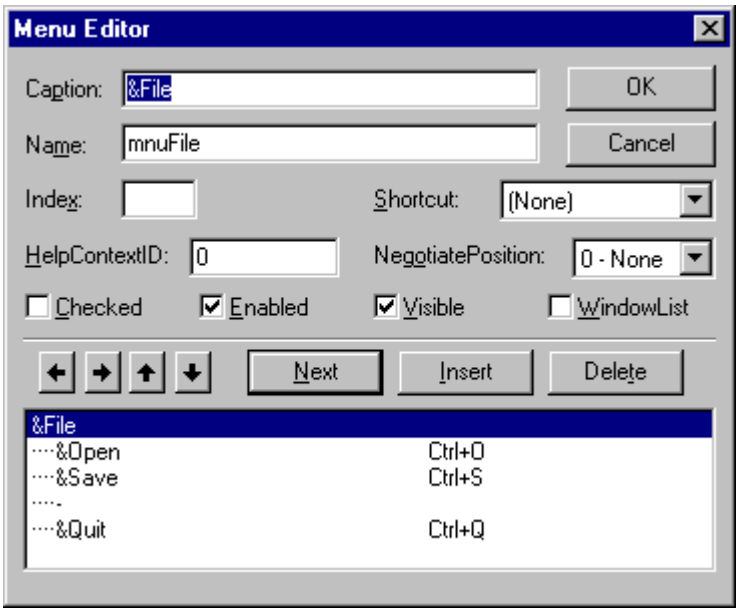
Instead of crowding your form with command buttons . . .

. . . you can organize your commands on a menu bar along the top of the form.

How Visual Basic Works

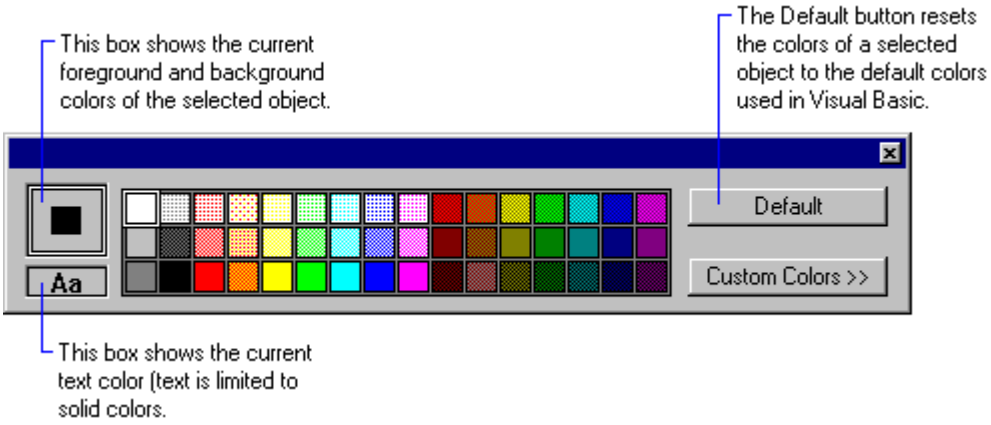
You design menus for your forms in the Menu Editor.

You can display this window by choosing Menu Editor from the Tools menu or by clicking the Menu Editor button on the toolbar.



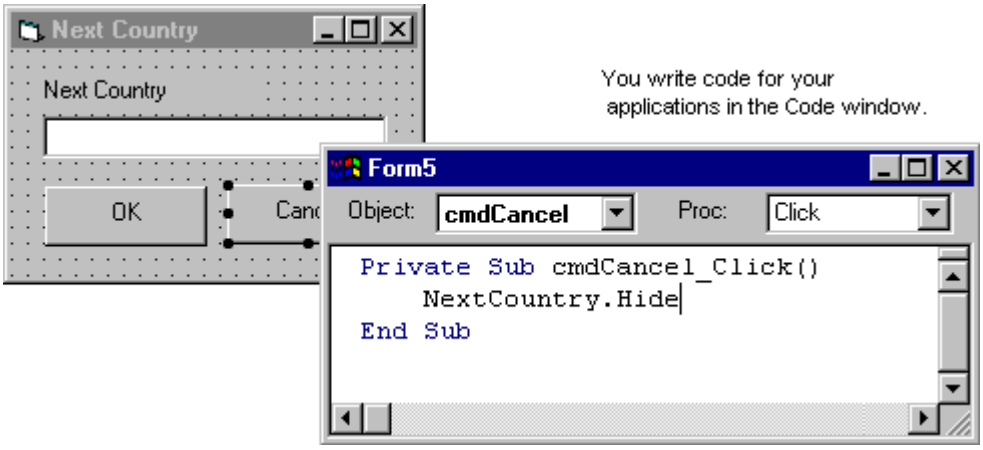
How Visual Basic Works

Visual Basic also provides a Color palette with 48 standard colors you can use to add color to your forms. Visual Basic can take advantage of 256-color, high-color, and true-color systems.



How Visual Basic Works

To make an application respond to user actions or system events, you write code for your forms and controls.



How Visual Basic Works

Visual Basic 4.0 now includes Visual Basic for Applications, the same language found in many Microsoft Office applications including Microsoft Excel and Microsoft Project. Language features include:

If...Then...Else blocks

```
If Pop > 0 And GNP > 0 Then
    Income = GNP / Pop
Else
    Income = 0
End If
```

Loops

```
Do While I <= 50
    Print I
    I = I + 1
Loop
```

Eleven data types

Integer	Currency
Long	String
Single	Boolean
Double	Object
Variant	User-defined Type
Byte	

Numerous math and string functions

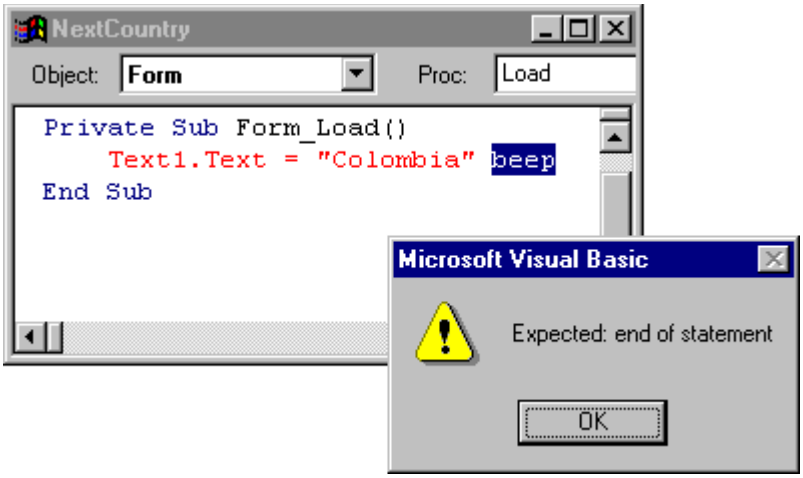
```
Abs(Rate)

Print UCase(Country)

MySqr = Sqr(4)

MyStr = Right(AnyString, 6)
```

How Visual Basic Works

To help you review your code quickly, Visual Basic provides:

- Automatic syntax checking

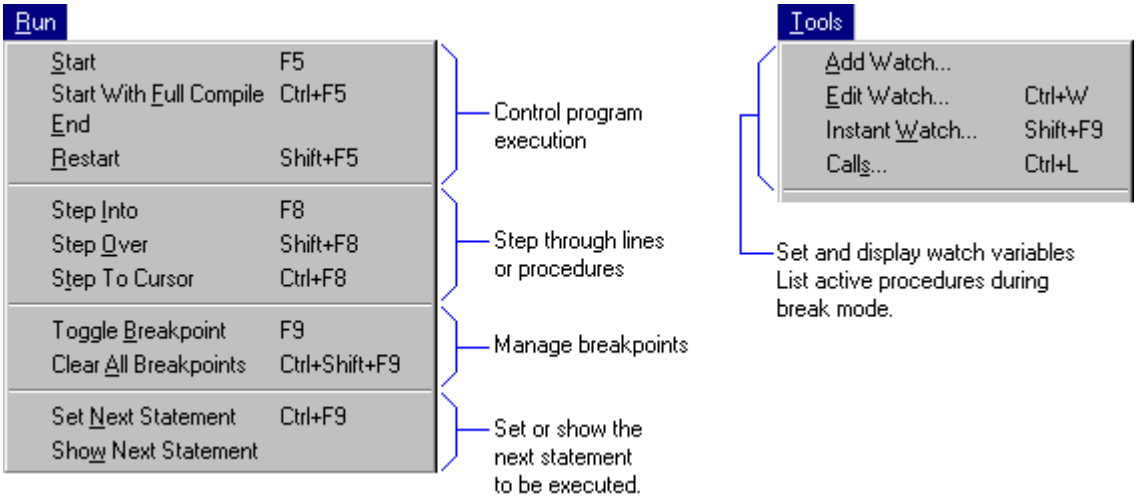
Visual Basic displays a message when it finds an error in your syntax.



How Visual Basic Works

To help you review your code quickly, Visual Basic provides:

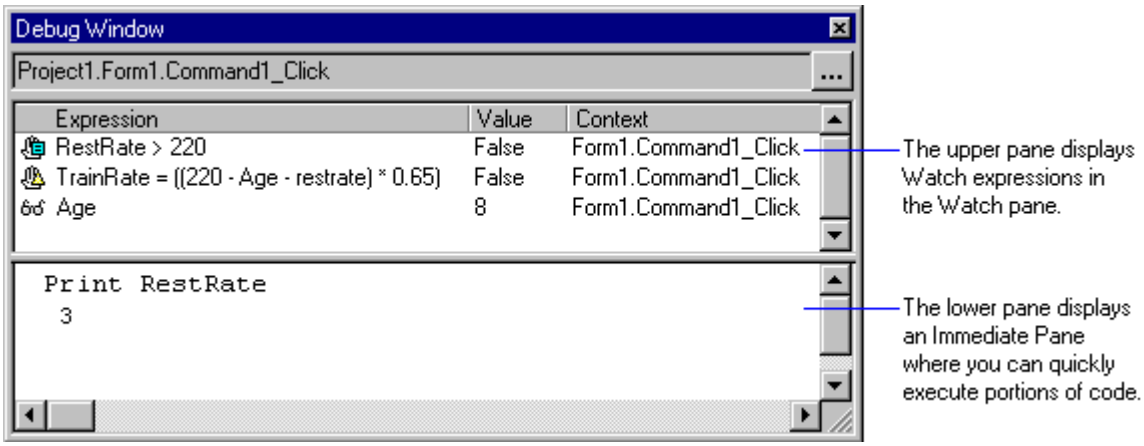
- Automatic syntax checking
- Debugging tools



How Visual Basic Works

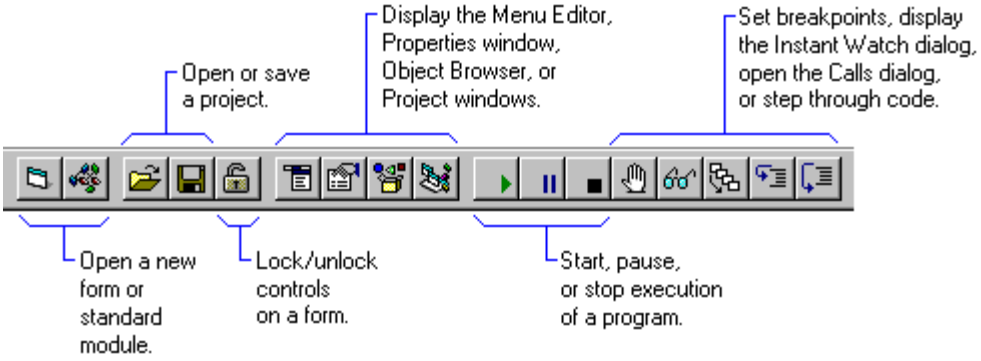
To help you review your code quickly, Visual Basic provides:

- Automatic syntax checking
- Debugging tools
- A Debug window



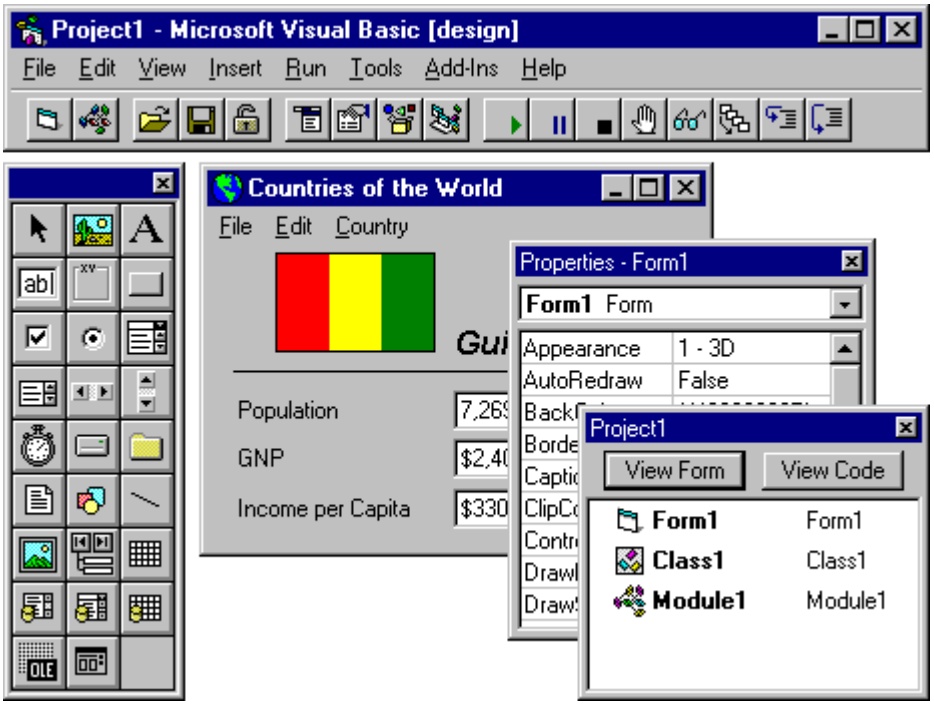
How Visual Basic Works

The Visual Basic toolbar provides shortcuts for many common design and debugging commands.



How Visual Basic Works

By combining the Visual Basic forms, tools, and programming language, you can build powerful applications quickly and easily.



How Visual Basic Works

This lesson introduced you to the Visual Basic development environment.

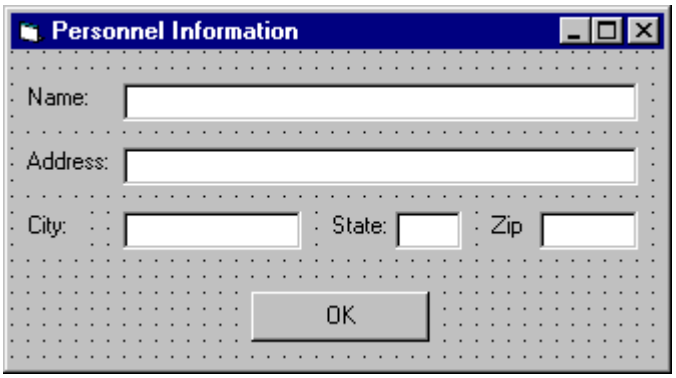
In the following lessons, you'll learn how to use Visual Basic to create applications.

How Visual Basic Works

Writing Event-Driven Programs

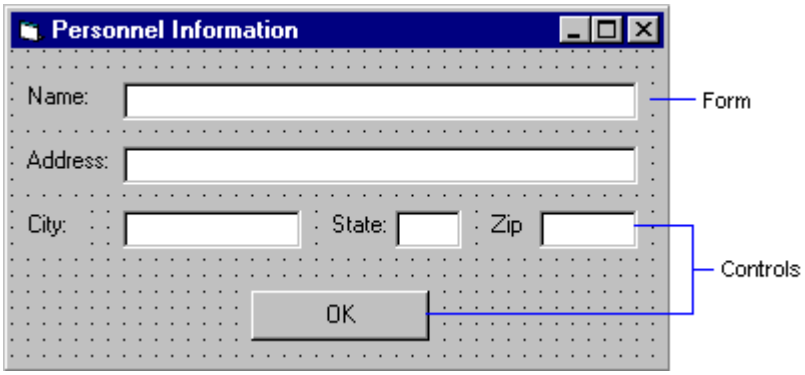
This lesson will introduce you to:

- Event-driven programming
- Event procedures
- Syntax for events



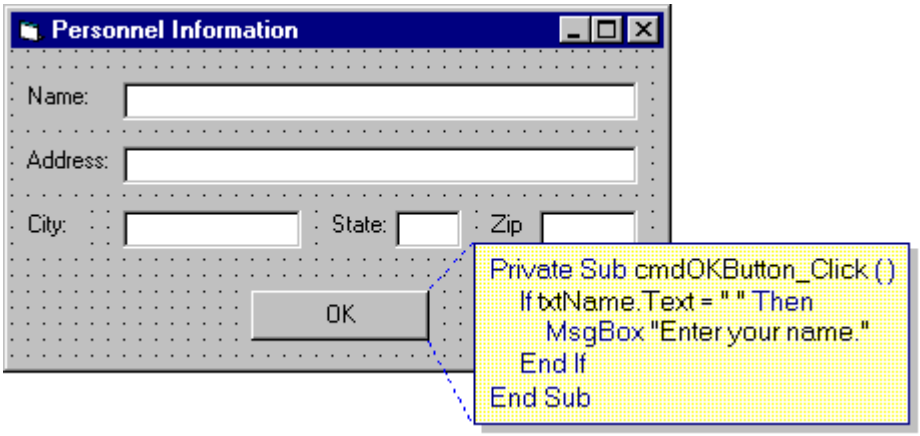
Writing Event-Driven Programs

The user interface of a Visual Basic application is made of objects—the forms and the controls you use to enable users to enter and view information. Each of these objects recognizes actions, such as the user clicking a button, opening a form, or typing in a field. These actions are referred to as **events**.



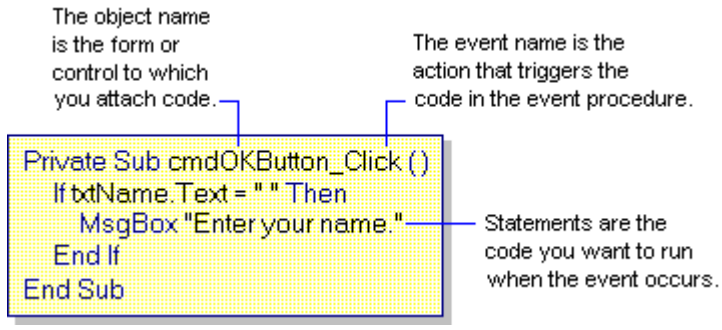
Writing Event-Driven Programs

When an event occurs in your application, Visual Basic automatically recognizes the event and runs the code that you've written for it. This code is called an **event procedure**.



Writing Event-Driven Programs

An event procedure name is made up of an object name and an event name. The object name is set by the form or control's Name property. There can be one or more statements in an event procedure.



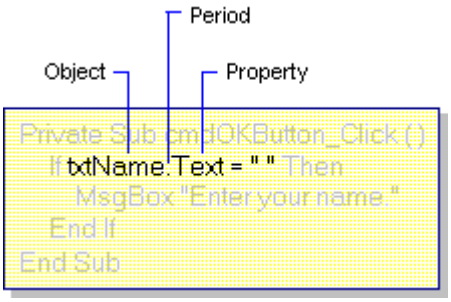
Writing Event-Driven Programs

Each form and control in Visual Basic responds to a predefined set of events. For example, a command button recognizes the following events.

Event	Action
Click	Button is selected with mouse or keyboard.
DragDrop	Control is dropped on the button.
DragOver	Control is dragged over the button.
GotFocus	Button gets the focus.
KeyDown	Key is pressed while button has the focus.
KeyPress	Key is pressed, and ASCII value is returned.
KeyUp	Key is released while button has the focus.
LostFocus	Button loses the focus.
MouseDown	Mouse button is pressed down over the button.
MouseMove	Mouse pointer is moved over the button.
MouseUp	Mouse button is released over the button.

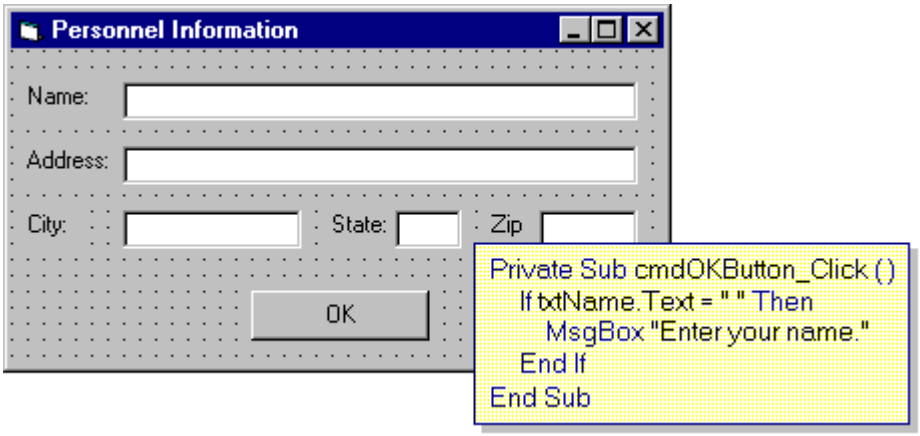
Writing Event-Driven Programs

You refer to a control and its properties in code using the `object.property` notation. For example, to refer to text displayed in a text box, the syntax would look like this:



Writing Event-Driven Programs

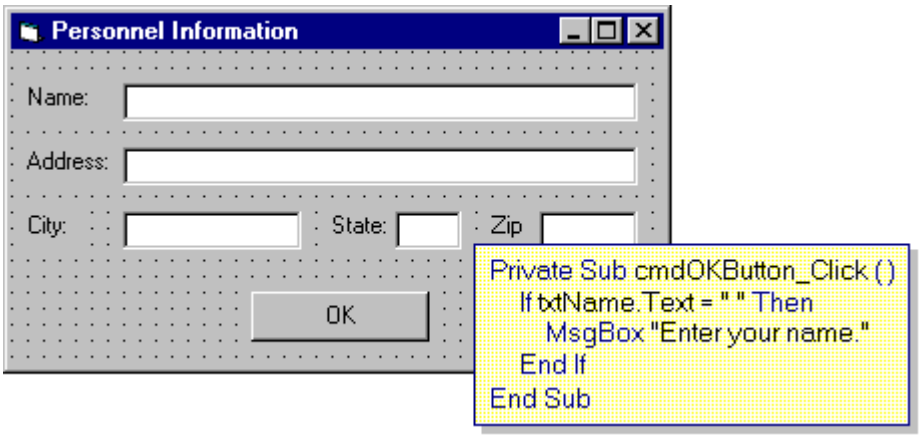
An event procedure runs only when the event occurs, and Visual Basic remains idle until that happens. For instance, the procedure below only runs when the OK button is clicked or when ENTER is pressed.



Writing Event-Driven Programs

By clicking the OK button, you've run the code in this event procedure. Every time the button is clicked, the code runs again.

You need to write code only for the events that you want your application to respond to.

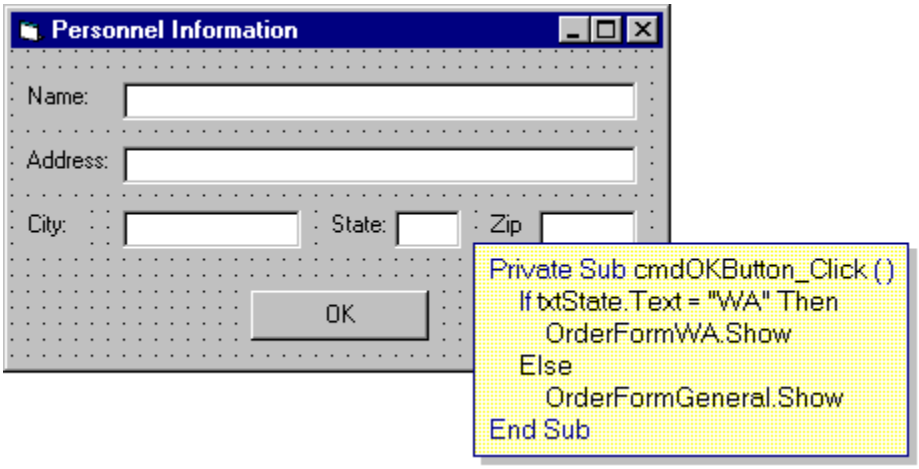


Writing Event-Driven Programs

To determine which events to write code for, think about what the user will do, and how you want the program to respond.

For example, you might want to check the contents of the text box when the user chooses the OK button.

Clicking the OK button triggers the cmdOKButton_Click event.



Writing Event-Driven Programs

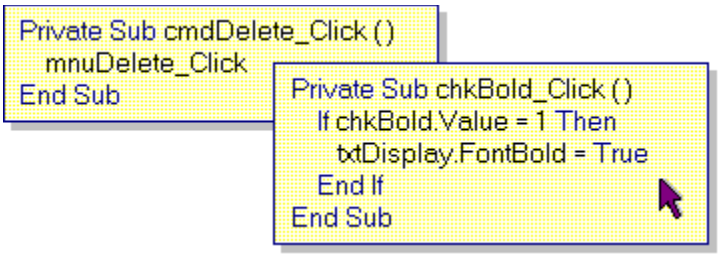
Event procedures can also:

- Trigger other event procedures.

Writing Event-Driven Programs

Event procedures can also:

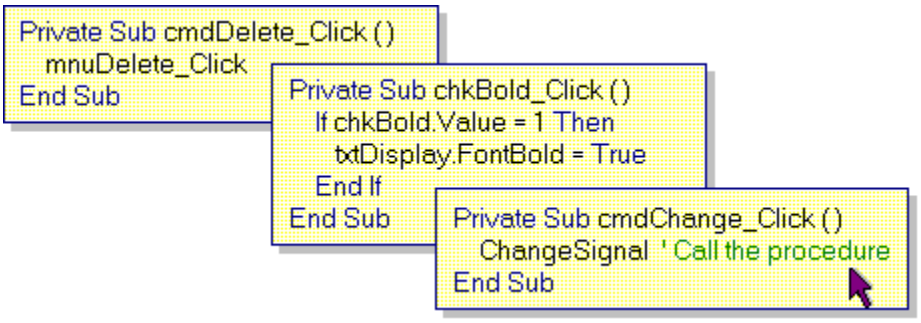
- Trigger other event procedures.
- Change an object's properties.



Writing Event-Driven Programs

Event procedures can also:

- Trigger other event procedures.
- Change an object's properties.
- Call other general procedures that are not tied to any event.



Writing Event-Driven Programs

This lesson covered:

- Event-driven programming
- Event procedures
- Syntax for events

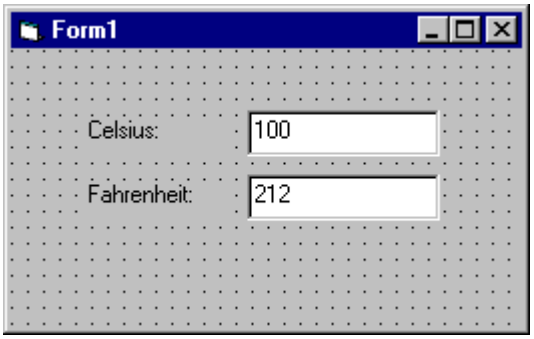
For more information on event-driven programming, see the lesson "Debugging Your Application" or Chapter 5, "Programming Fundamentals," in the *Programmer's Guide*.

Writing Event-Driven Programs

Creating an Application

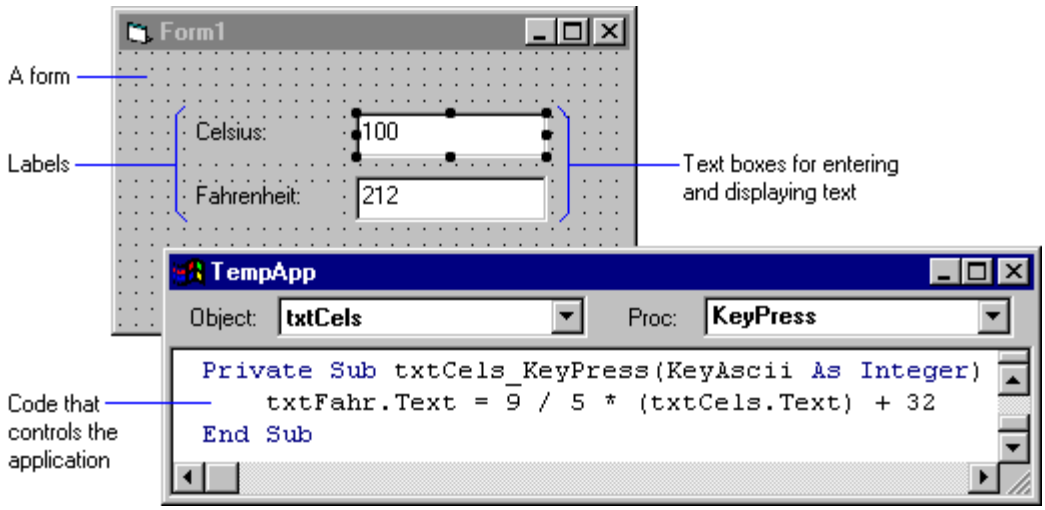
This lesson demonstrates how to build a Visual Basic application.

This sample application converts temperatures between Fahrenheit and Celsius.



Creating an Application

The Temperature Conversion application consists of these elements:



Creating an Application

There are three steps to creating an application:

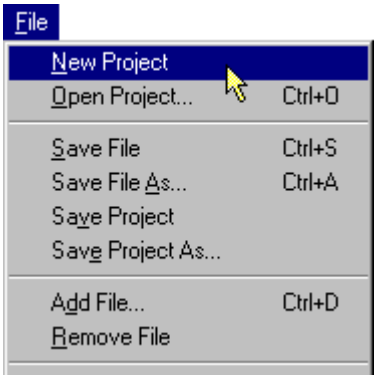
1. Create the interface.
2. Set properties.
3. Write code.

Creating an Application

1. Create the interface.

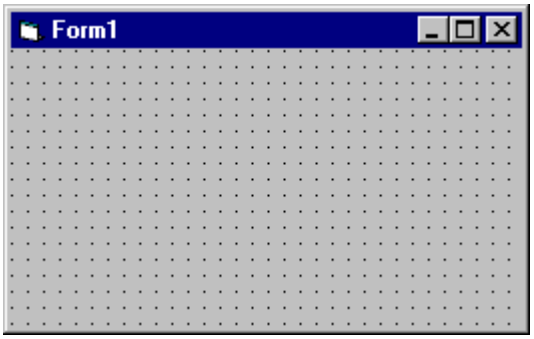
To create an application, you first need to open a new project.

You open a project by choosing the New Project command from the File menu.



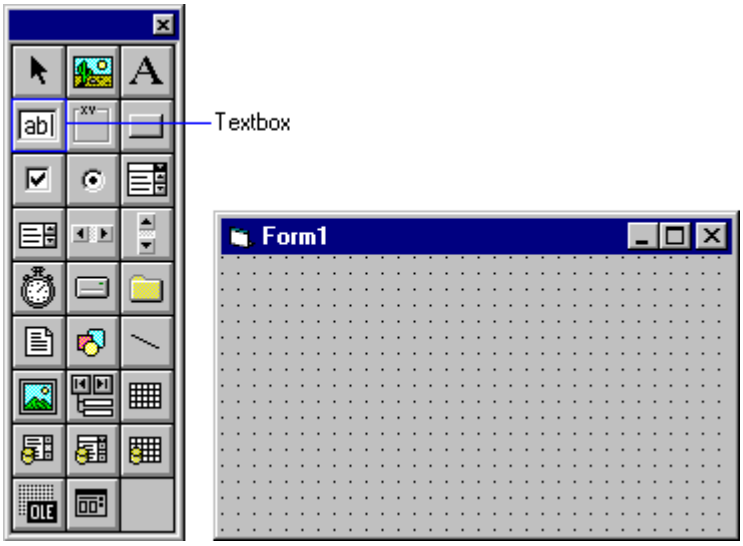
Creating an Application

Every new project contains one form. You can add as many additional forms as your application needs. The Temperature Conversion application requires only one form.



Creating an Application

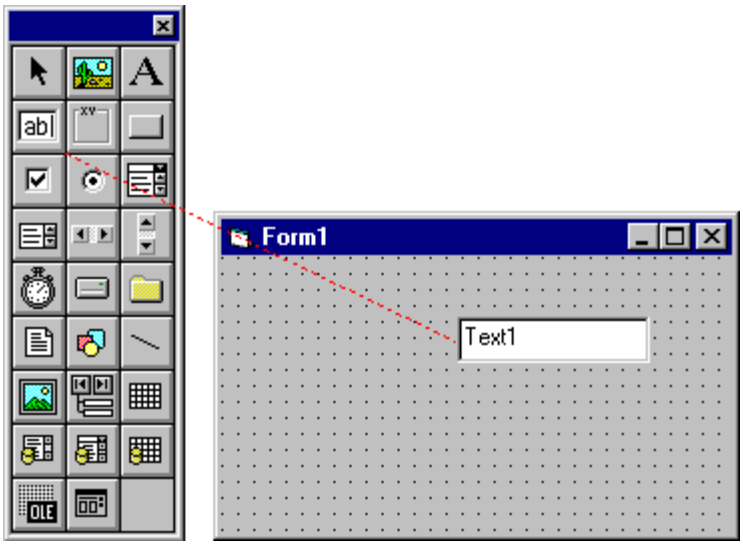
Next, you can select the tools you need from the Toolbox to draw the controls you want on the form. In this application, text boxes accept user input and display text.



Creating an Application

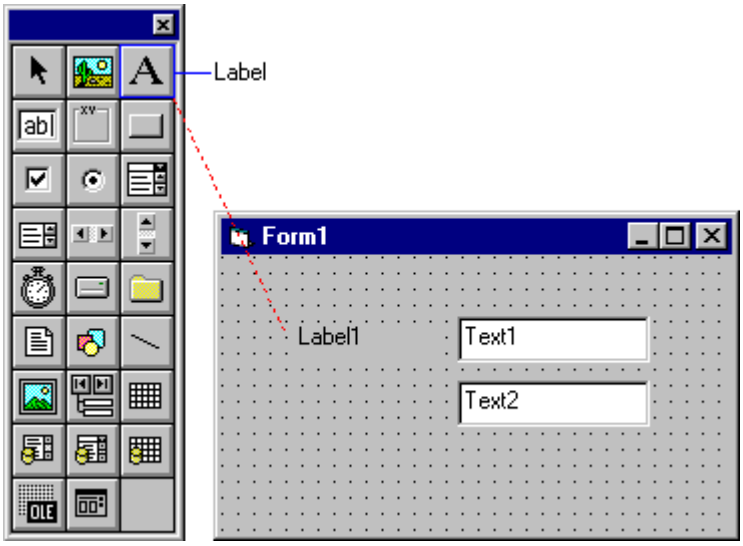
To create a control, select the tool from the Toolbox, and then hold down the left mouse button while dragging out an area on the form.

You can create a control in the default size by double-clicking the tool or selecting a tool and pressing ENTER.



Creating an Application

We'll create two text boxes, one for Fahrenheit temperatures and one for Celsius temperatures. To describe the contents of each text box, we'll use labels. Labels can't be changed by the user.

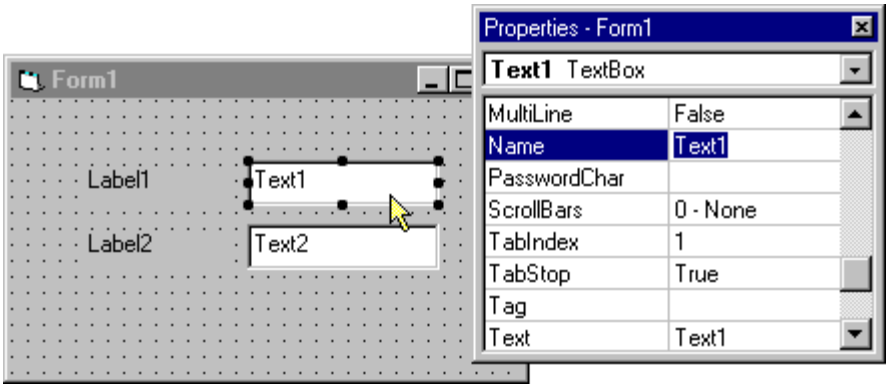


Creating an Application

2. Set properties.

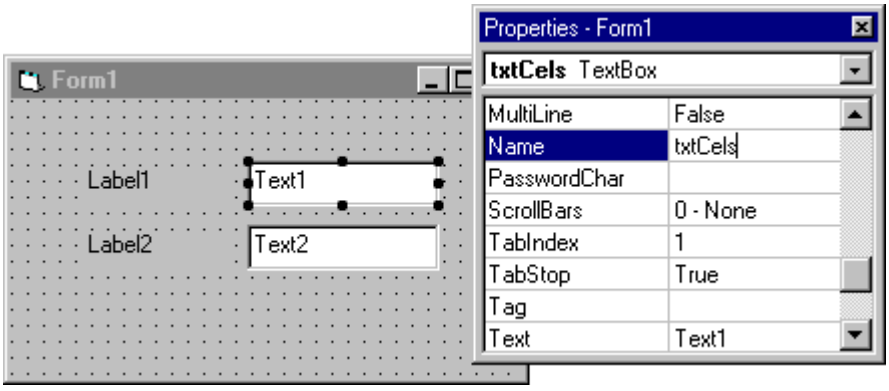
You set the properties of a control in the Properties window.

When you select a form or control, its properties and their settings are displayed in the Properties window.



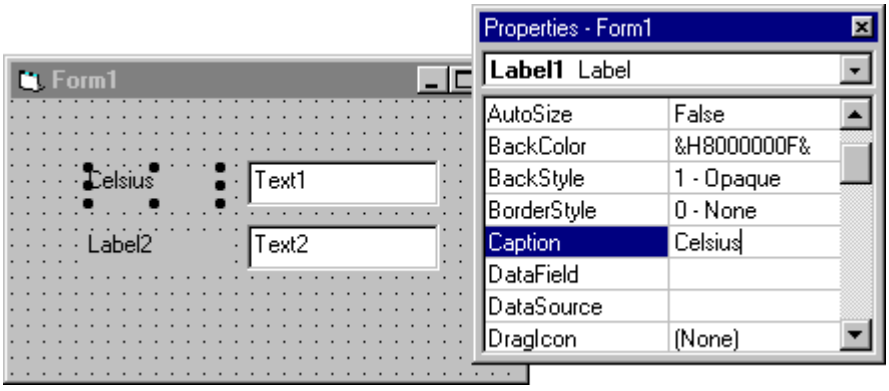
Creating an Application

You use the Name property to refer to a control in code. In this case, we'll make txtCels the name of this text box.



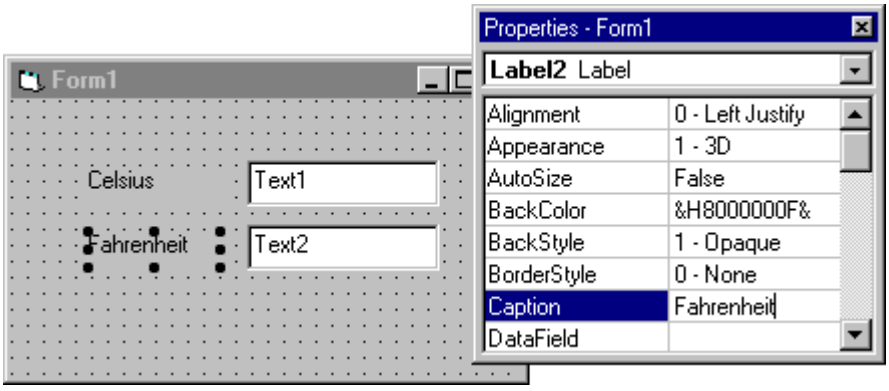
Creating an Application

The Caption property specifies the text that is displayed in the control. We'll change the label's Caption property to Celsius.



Creating an Application

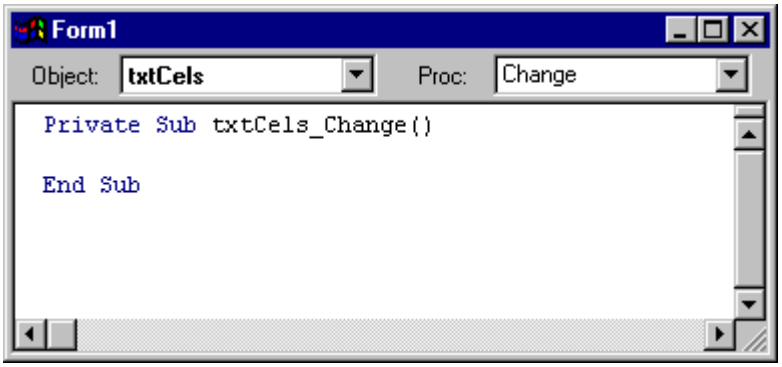
Now, we'll change the Name property of the other text box to txtFahr and the Caption property of its label to Fahrenheit.



Creating an Application

3. Write code.

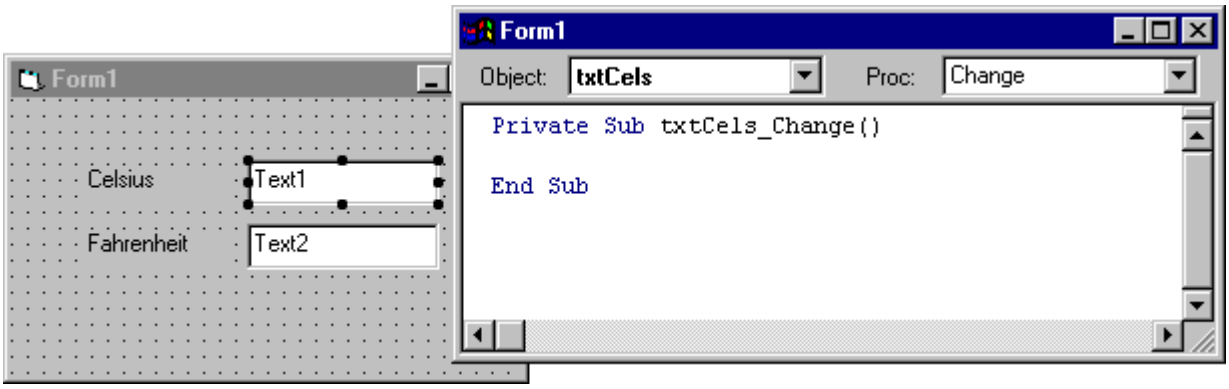
We now need to add code to make the application respond to the user's actions.



Creating an Application

When you double-click a control, the focus shifts to the Code window. The code template for the selected control's default event procedure is displayed.

The name of the control appears in the Object list box.

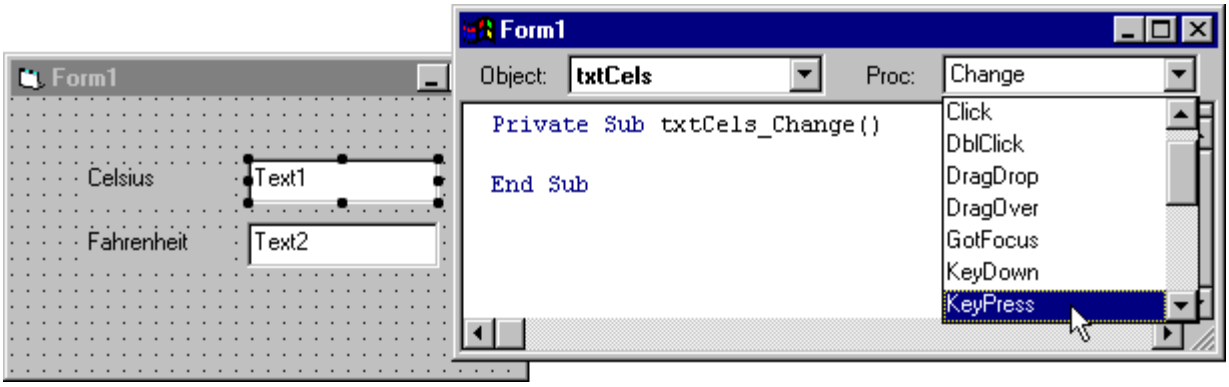


Creating an Application

The Procedure list box contains a list of events for the control.

Since we want to run the code in this application when a user presses a key, we'll select the KeyPress event from the Procedures list box.

Visual Basic then displays the template for the event procedure we want to write.

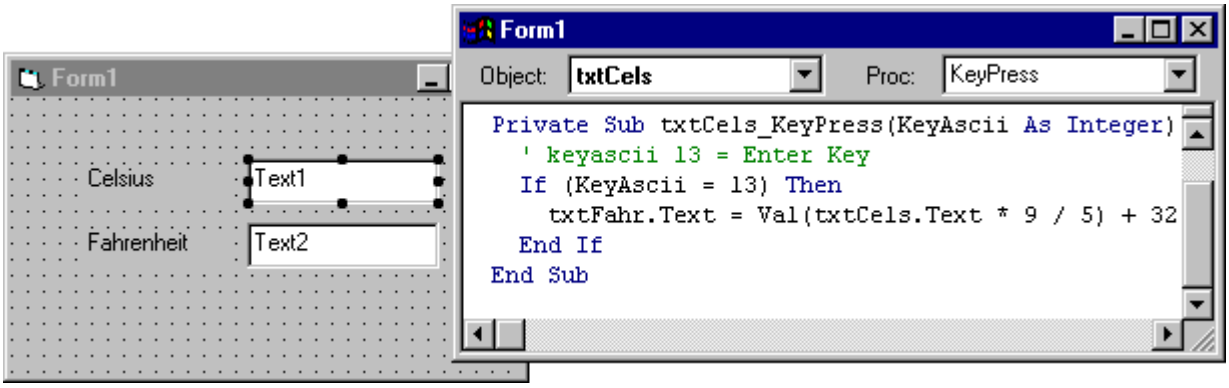


Creating an Application

We'll use the following formulas to convert the temperatures:

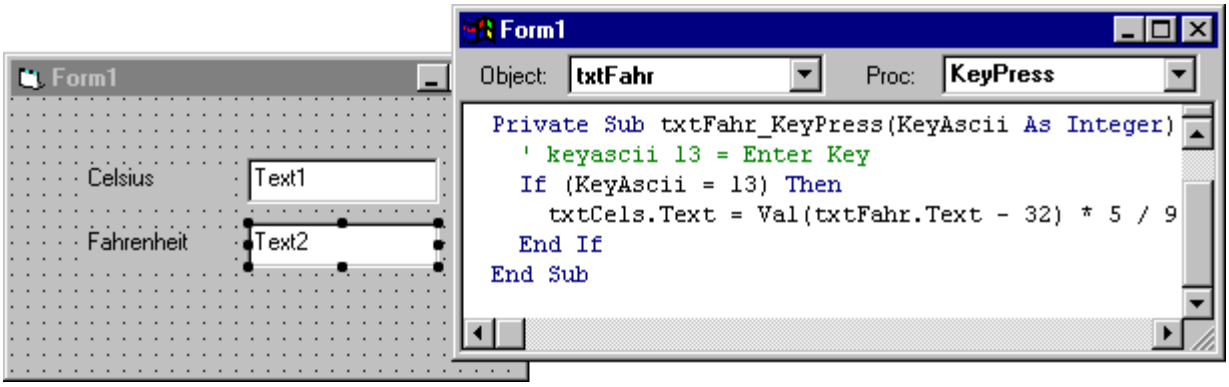
```
Cels = (Fahr - 32) * 5/9
Fahr = (Cels * 9/5) + 32
```

Next, we'll enter the code for the event procedure.



Creating an Application

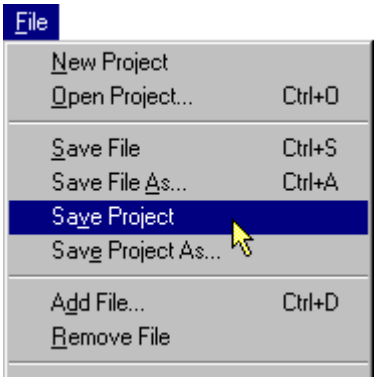
Now we'll attach a similar event procedure to the other text box.



Creating an Application

When you finish working on your application, you'll want to save the project.

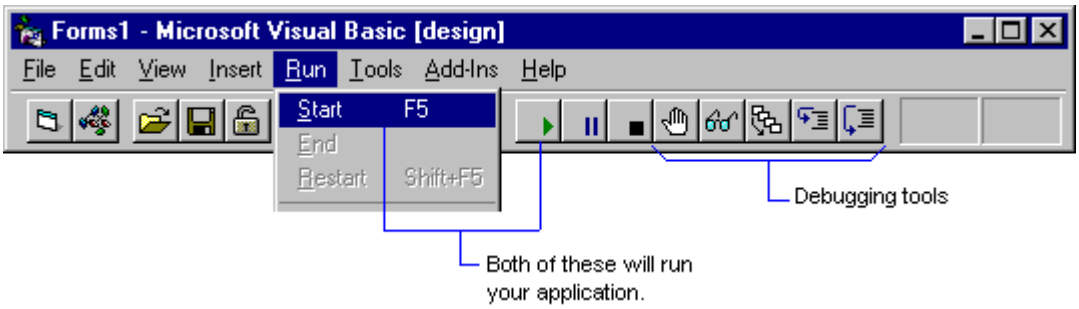
Choosing Save Project from the File menu prompts you to assign a name to the project.



Creating an Application

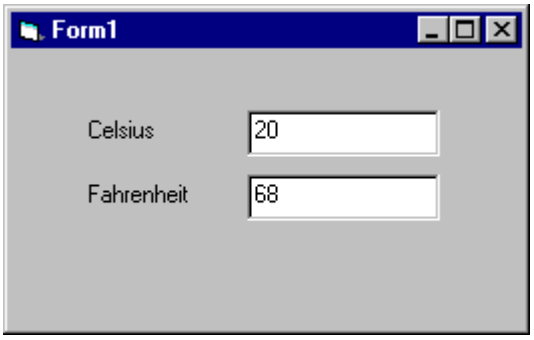
You can test your application by choosing Start from the Run menu or by clicking the Start button (▶) on the toolbar.

You can also use the debugging tools to help locate and fix problems in your application.



Creating an Application

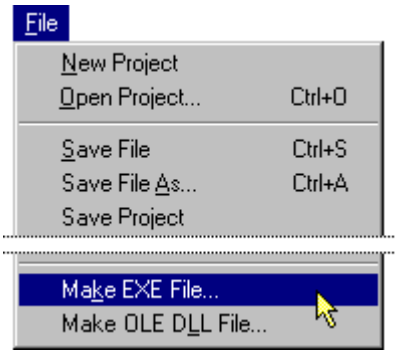
When you run the application, it will convert Celsius to Fahrenheit, or Fahrenheit to Celsius when you type in a number and press the Enter key.



Creating an Application

When your application is exactly the way you want it, you can make it into an executable file. This allows you, and other users, to run the application outside the Visual Basic environment.

To create an executable file from your project, you use the Make EXE File command from the File menu.



Creating an Application

This lesson demonstrated how to create a simple application:

1. Create the interface.
2. Set properties.
3. Write code.

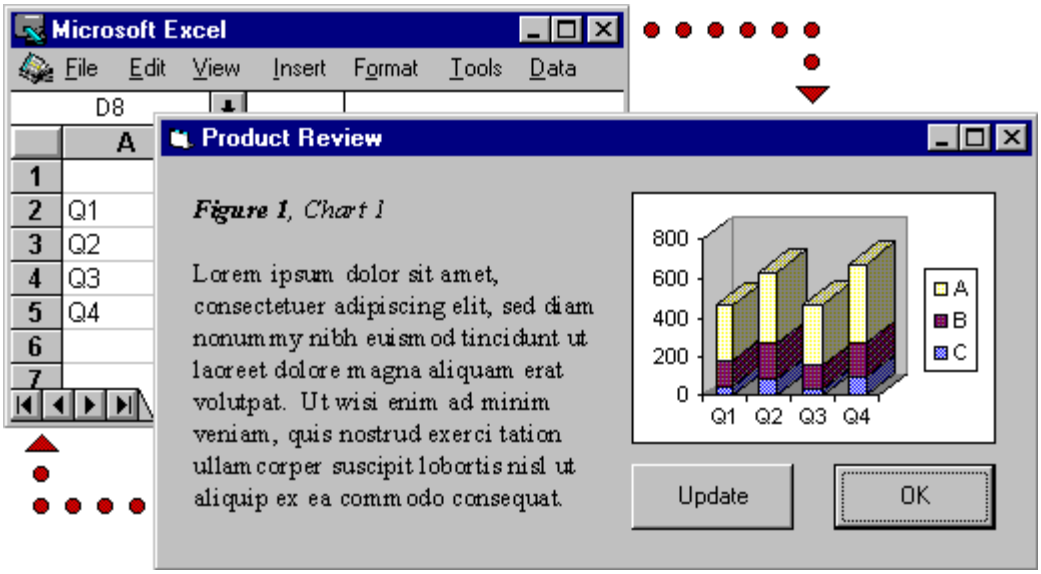
The lessons that follow explain the parts of a Visual Basic application in more detail.

For more information, see Chapter 2, "Your First Visual Basic Application," in the *Programmer's Guide*.

Creating an Application

Introduction to OLE Automation Objects

This lesson will introduce you to objects and OLE Automation.

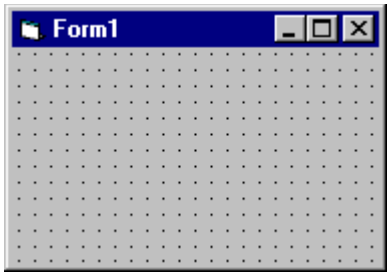


Introduction to OLE Automation Objects

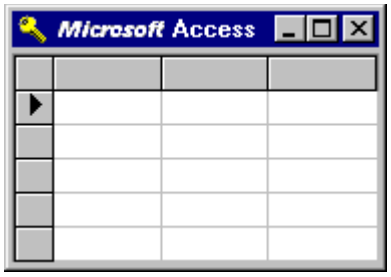
Many Windows-based applications, including Visual Basic, are composed of different objects. Each object is a combination of code and data that can be treated as a unit. An object can be a piece of an application, like a control or a form. An entire application can also be an object.

Some examples of objects include:

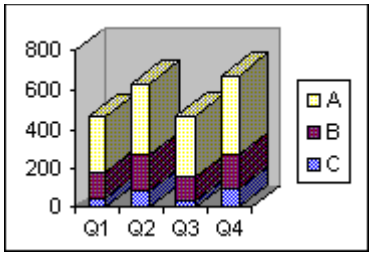
Form



Database



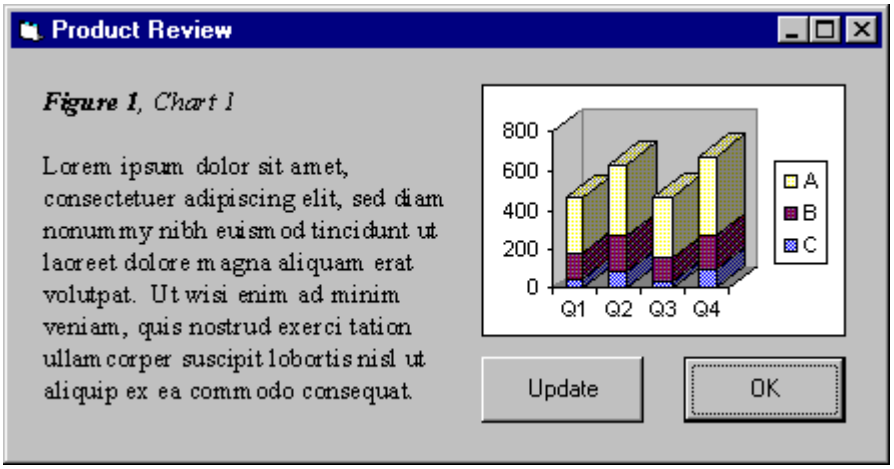
Chart



Command button

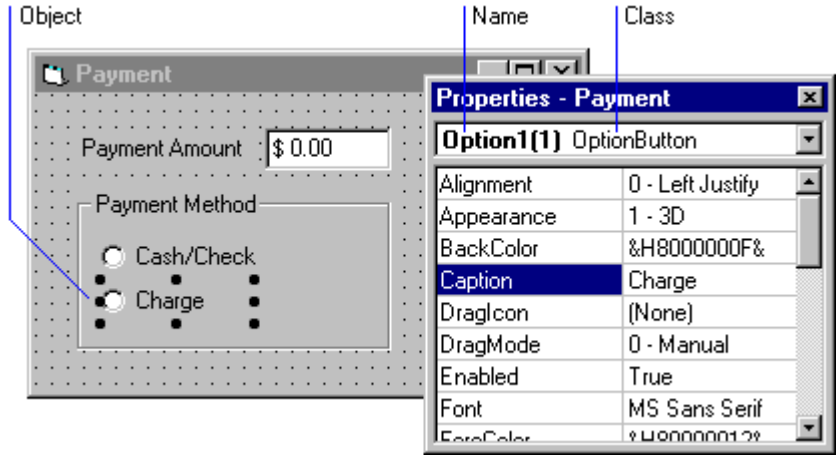
Introduction to OLE Automation Objects

Because each object is a unit of code designed for a specific purpose, you can create applications by combining different types of objects together.



Introduction to OLE Automation Objects

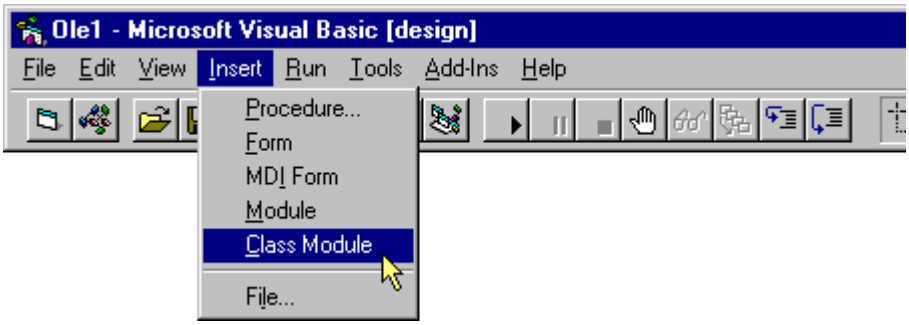
Each object is defined by a class. The class defines the characteristics of the object. In Visual Basic, the Properties window displays the class name of each object.



Introduction to OLE Automation Objects

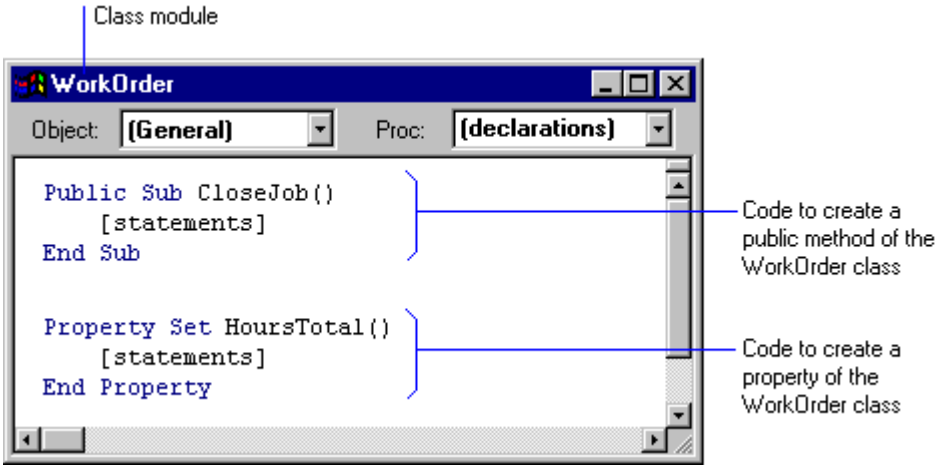
Visual Basic defines many types of objects you can use in applications, but you may need capabilities not available from existing classes. In this case you can use class modules to define new classes, or types of objects within Visual Basic.

Class modules allow you to define your own types of objects, and create properties and methods for them. The properties and methods become members of the class. The new class can be private to your application. You can also make it accessible to other applications.



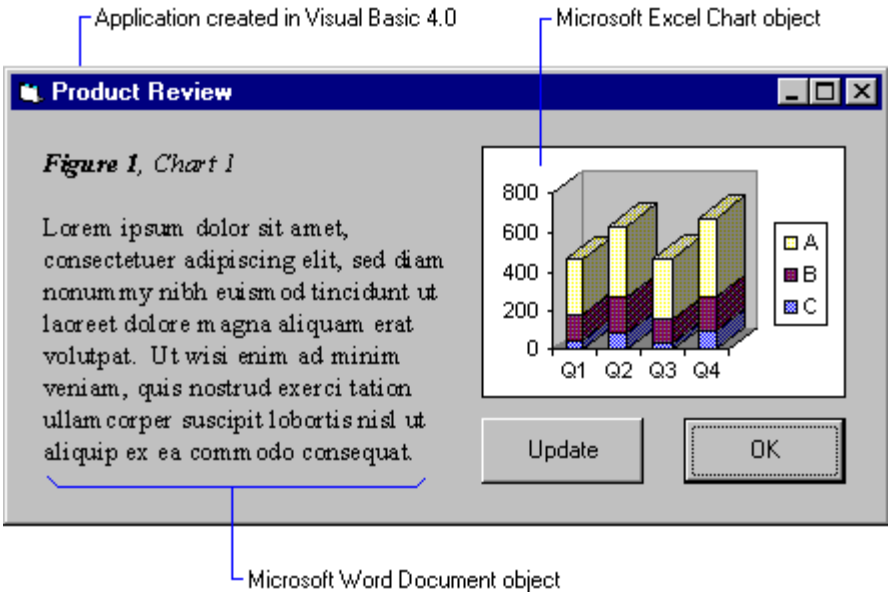
Introduction to OLE Automation Objects

You might decide, for example, that you need a 'work order object' for a new application. You can insert a class module in your project and then add code to create the properties and methods needed to provide the behaviors and characteristics of a work order.



Introduction to OLE Automation Objects

Other applications expose objects you can use in Visual Basic 4.0 applications. For example:



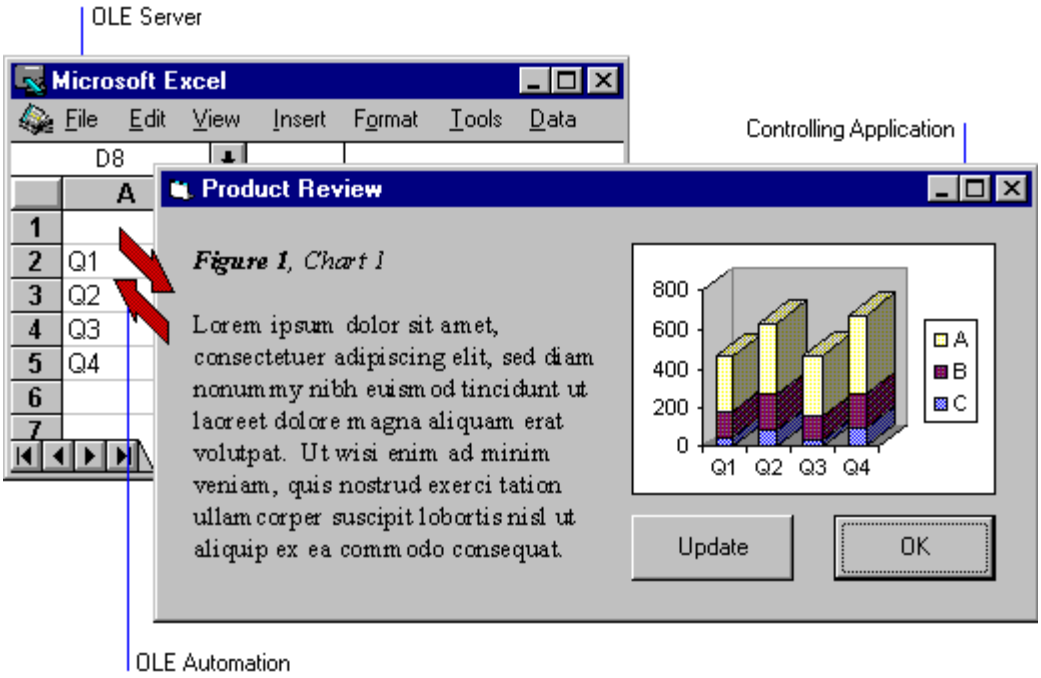
Introduction to OLE Automation Objects

Objects that are provided by other applications are called OLE automation objects. You can perform many of the same tasks with OLE Automation objects that you can with Visual Basic objects. For instance, you can:

- Set Properties.
- Return properties.
- Invoke an object's methods.

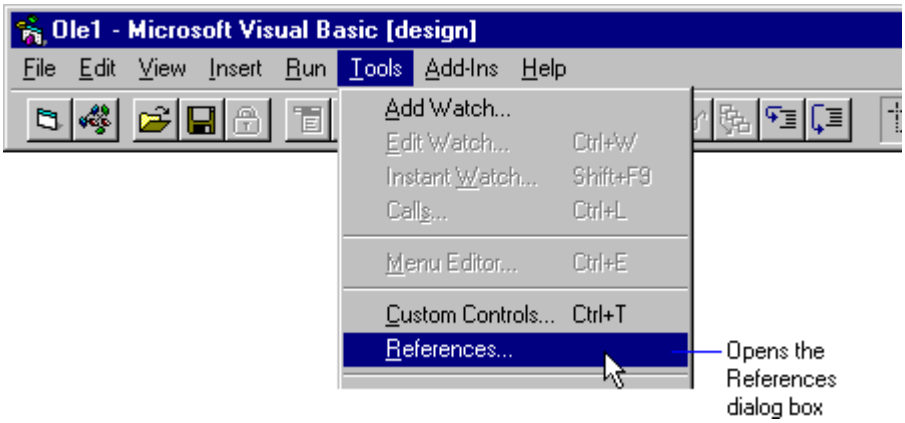
Introduction to OLE Automation Objects

Visual Basic uses OLE Automation to communicate with other applications. OLE Automation is an industry standard, designed to provide a consistent way for applications to share objects. Applications that provide objects are called OLE servers, while applications that use objects are controlling applications.



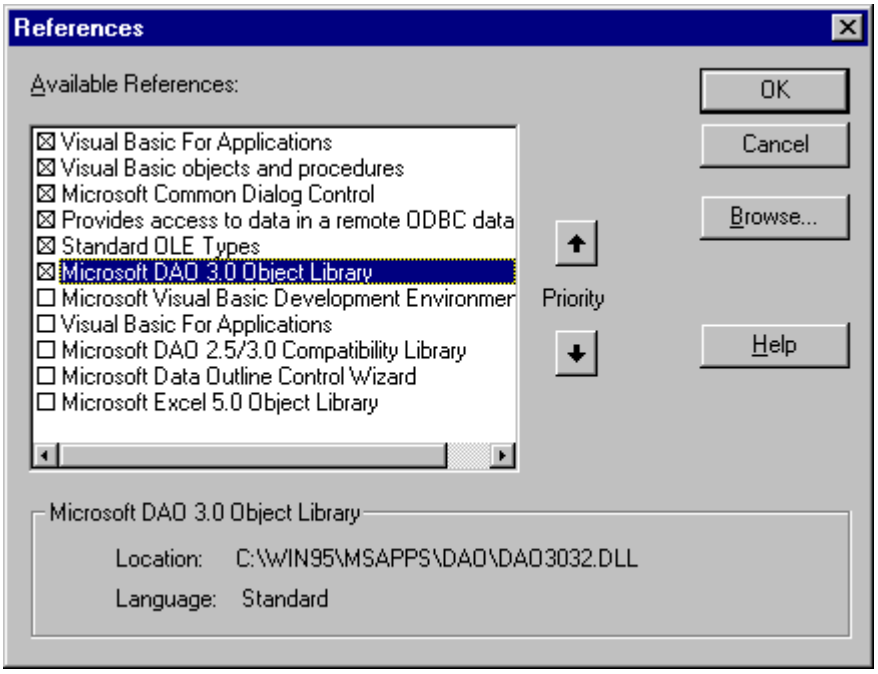
Introduction to OLE Automation Objects

The easiest objects to use are those whose applications include an object library. To use these objects, use the References dialog box, available from the Tools menu.



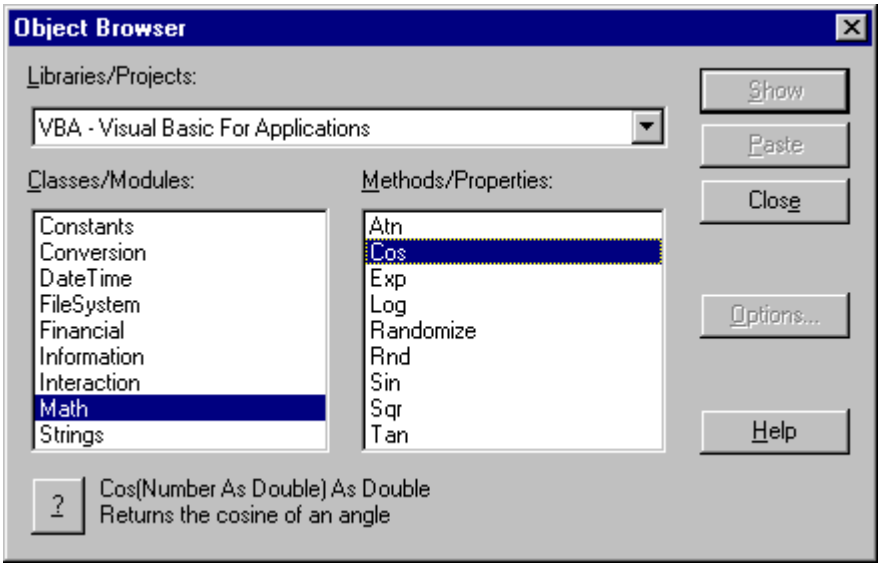
Introduction to OLE Automation Objects

If the object library is already listed, select the check box next to its name. If it isn't listed, use the browse button to search for either .TLB or .OLB extensions. You can also search for .EXE and .DLL files since they sometimes contain object libraries.



Introduction to OLE Automation Objects

After you set a reference to the object library, you can view the objects contained in that library with the Object Browser. In addition, you can also view the properties and methods associated with each object.



Introduction to OLE Automation Objects

To use an OLE Automation object, you must:

1. Create storage for a variable of an object type.
2. Set the variable to reference a new or existing object.
3. Write code using the object's properties and methods.
4. Release the object when finished.

Introduction to OLE Automation Objects

1. Create Storage for a Variable of an Object Type

To create storage, declare an object variable:

```
Dim objX As Object
```

Introduction to OLE Automation Objects

2. Set the Variable to Reference a New or Existing Class

If you need to create a new instance of a class, use the CreateObject function. For example, this will create a new Microsoft Excel Worksheet object:

```
Set objX = CreateObject("Excel.Sheet")
```

If the object already exists in a file, you can use the GetObject function to load it. For example, this will set a reference to an existing Microsoft Excel Worksheet object:

```
Set objX = GetObject("C:\EXCEL\REVENUE.XLS")
```

Introduction to OLE Automation Objects

3. Write Code Using the Object's Properties and Methods

Now you have a reference to an object provided by another application. You can use this object much like you would use a control provided by Visual Basic. For example, using the Microsoft Excel Worksheet object, objX, we can add data to cells in the worksheet using the worksheet's properties and methods.

```
objX.Application.Visible = True
For i = 1 to 10
   objX.Cells(i,i).Value = i
Next i
```

The Microsoft Excel Worksheet in this example is a visual object. You can mix and match visual objects from other applications on a form as shown earlier in this lesson. Objects without visual interfaces, such as data access objects, cannot be placed on a form but can still be accessed and manipulated through Visual Basic code.

Introduction to OLE Automation Objects

4. Release the Object When Finished

Open objects consume resources. When you are finished using an object, clear any variables that reference the object so the object can be released from memory. To clear an object, set it to Nothing. For example:

```
Set objX = Nothing
```

Introduction to OLE Automation Objects

This lesson introduced you to objects and OLE Automation.

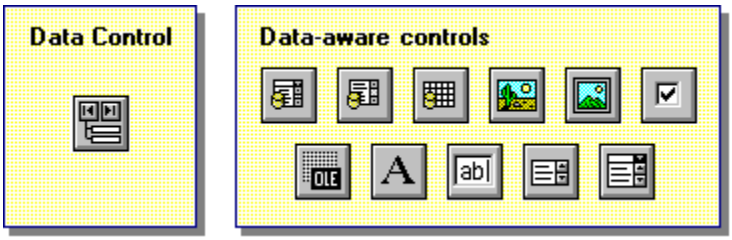
For more information on using objects and OLE Automation, see Chapter 7, "Introduction to Objects," and Chapter 9, "Programming Other Applications' Objects," in the *Programmer's Guide.*

Introduction to OLE Automation Objects

Accessing Databases

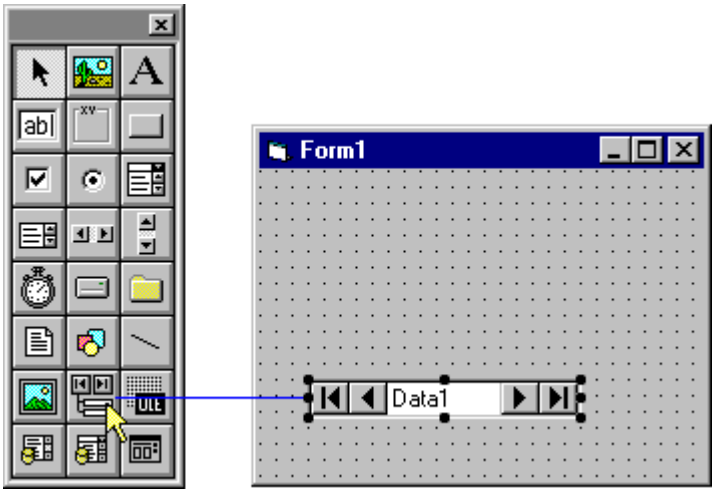
Visual Basic includes the Microsoft Jet database engine, the same engine that powers Microsoft Access. Using Visual Basic, you can display, edit, and update information from many types of databases.

- In Visual Basic Standard edition, you can use the data control to access information in existing databases.
- In Visual Basic Professional, you can also create or modify databases using the Data Access Objects programming interface.



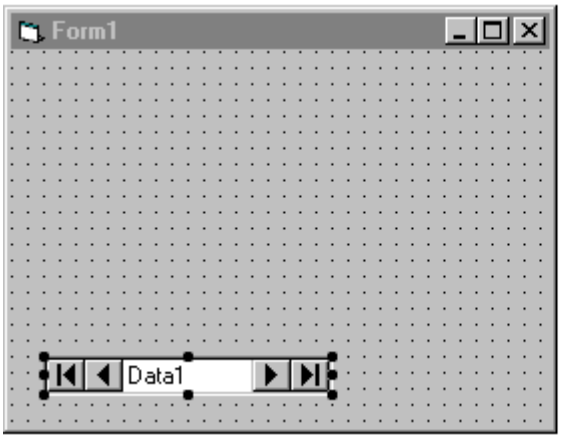
Accessing Databases

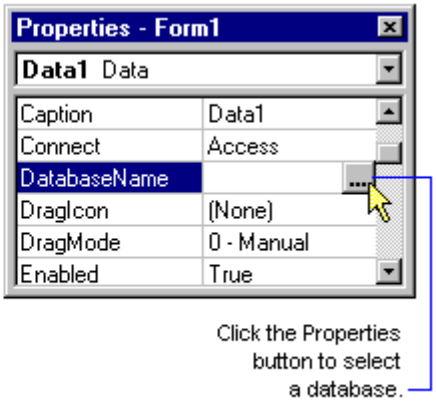
The data control provides the easiest way to access information in an existing database. Double-click the data control to add it to your form.



Accessing Databases

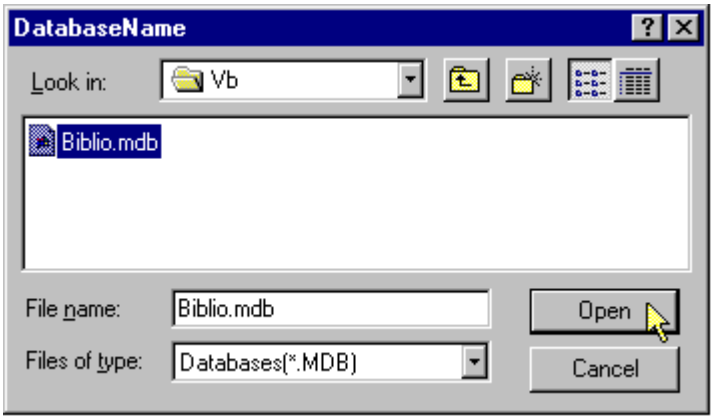
After you add the data control to your form, you set properties for it. The DatabaseName property specifies the database you want to access.





Accessing Databases

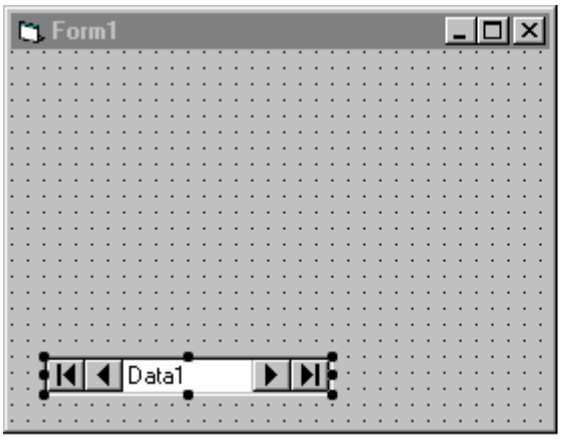
In this case, we'll use the sample database that ships with Visual Basic 4.0. The name of the database is BIBLIO.MDB.

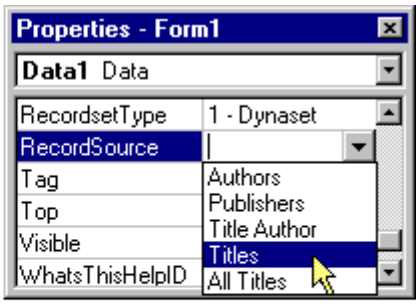


Accessing Databases

You next set the RecordSource property. Click the down arrow to display a list of tables and queries stored in the database.

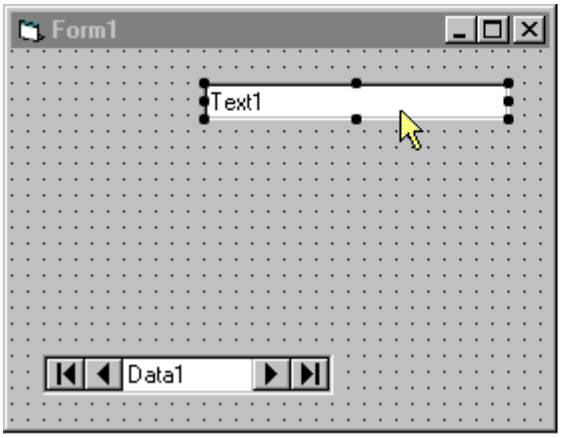
We'll set the RecordSource to Titles.

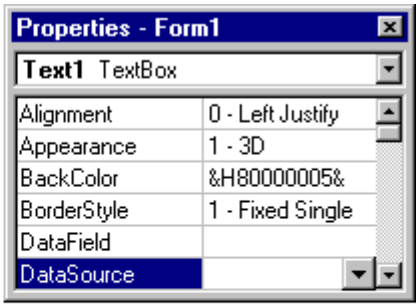




Accessing Databases

We'll now add a text box to the form. For the text box to display information from our database, we must first "bind" the text box to the data control.

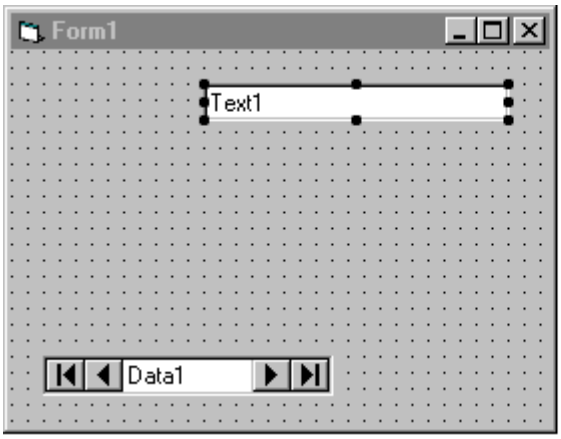


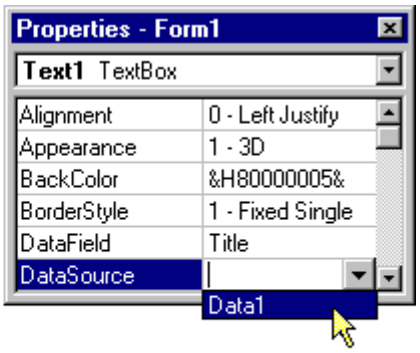


Accessing Databases

To bind the text box to the data control, you set the DataSource property of the text box to the name of the data control.

Since only one data control is on the form, that is the only item that appears in the list.

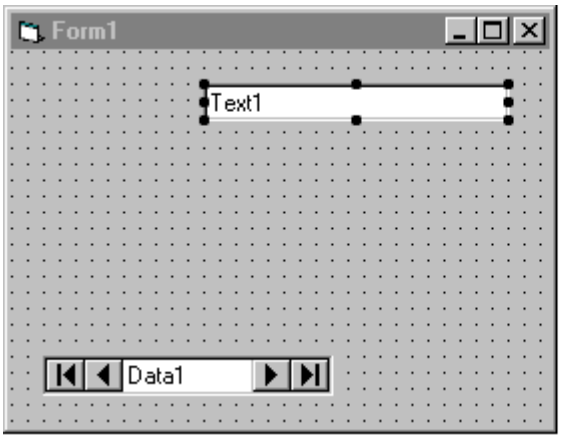


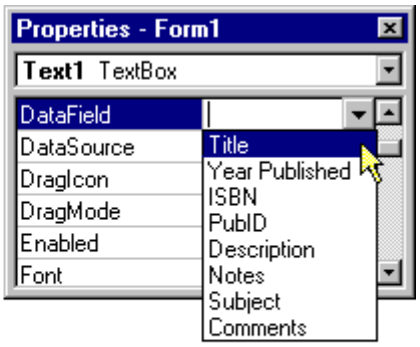


Accessing Databases

After the text box is bound to the data control, you can set the field that the text box will display. Click the down arrow for the DataField property to display a list of available fields for the current DataSource.

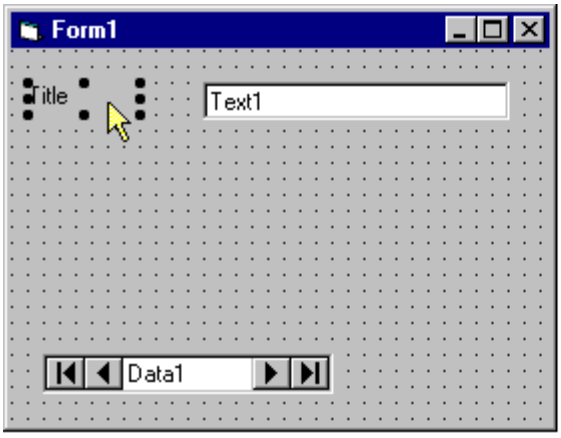
We'll set the DataField to Title.

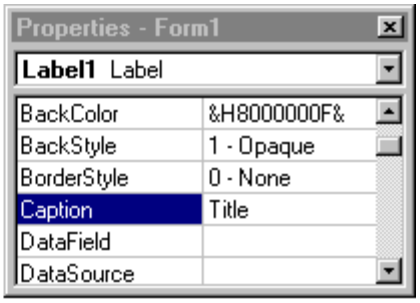




Accessing Databases

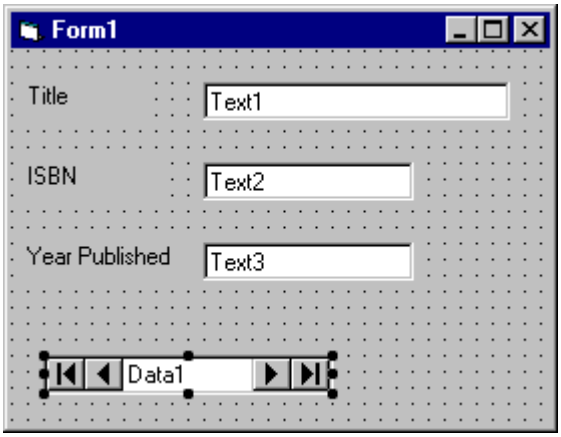
Next we'll add a label and set its caption property to Title.





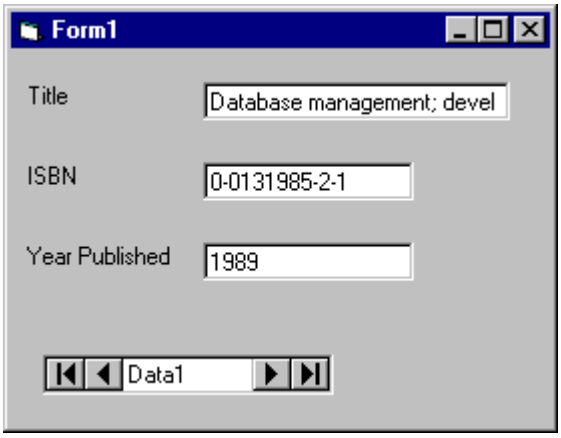
Accessing Databases

Now we'll add two more text boxes and two more labels. The text boxes are set to display values in the ISBN number and the Year Published fields.



Accessing Databases

When you run the application, the text boxes display information from the three fields you specified. You can click the buttons on the data control to scroll between records in the database.



Accessing Databases

This lesson provided an introduction to using the data control.

For more information on accessing databases, see Chapter 22, "Accessing Databases with the Data Control," in the *Programmer's Guide*.

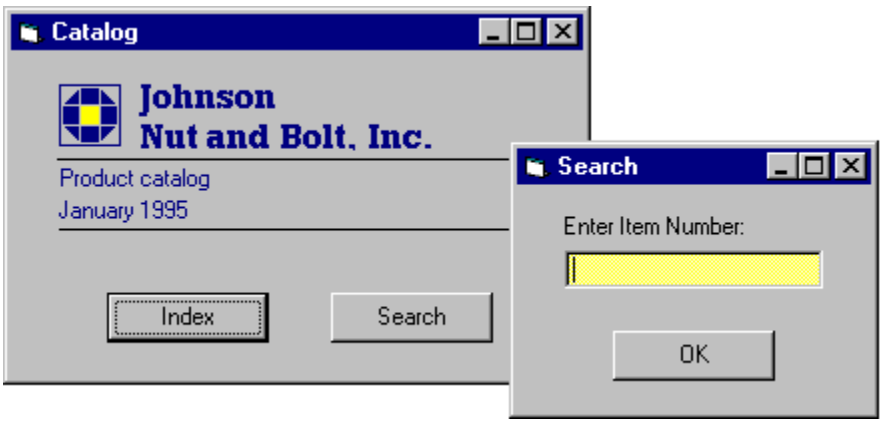
Accessing Databases

Using Color and Graphics

With Visual Basic, you can easily add color and graphics to an application.

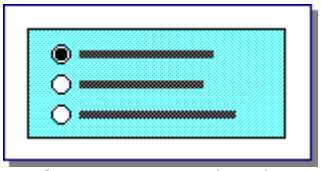
This lesson covers:

- Using graphics
- Using colors
- Changing graphics and colors with code
- Designing forms with graphics and colors



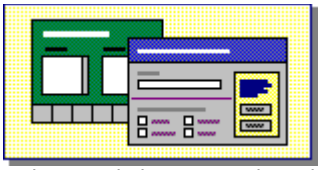
Using Color and Graphics

Graphics can increase the usefulness of and add visual impact to your forms.

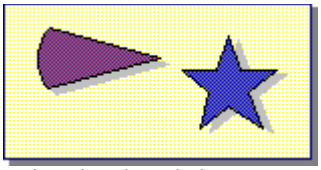


A frame can contain other controls.

A picture box or an image control can display a bitmap, an icon, or a metafile.



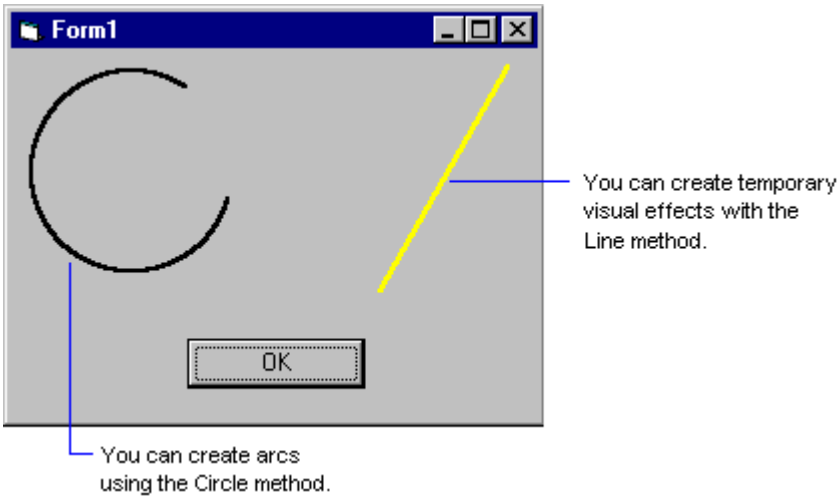
Lines and shapes can be added to forms to emphasize information.



The Visual Basic language can be used to draw other shapes.

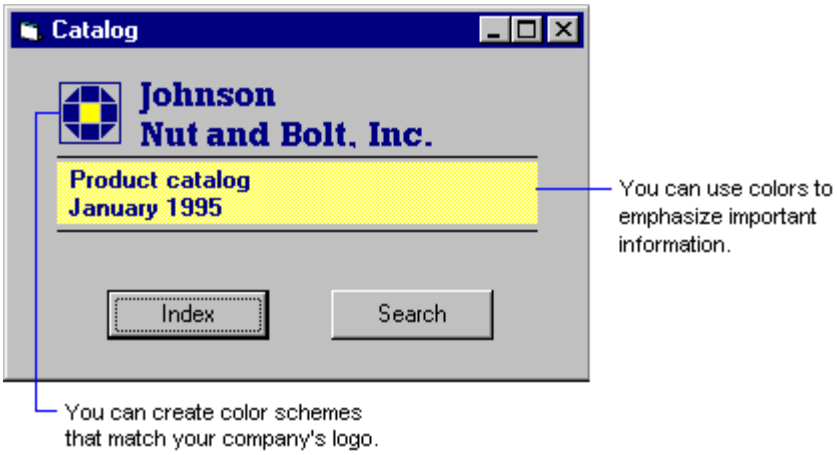
Using Color and Graphics

Graphics methods in the Visual Basic language also let you create graphics in your application. Creating graphics with graphics methods is done in code, so these images appear on your form only when the application is running.



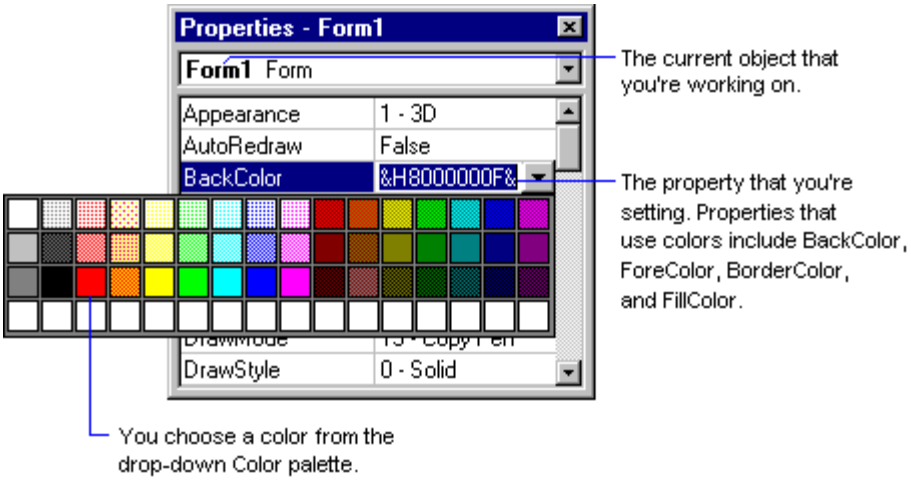
Using Color and Graphics

Adding color to your forms and controls can make your applications more attractive and easier to use.



Using Color and Graphics

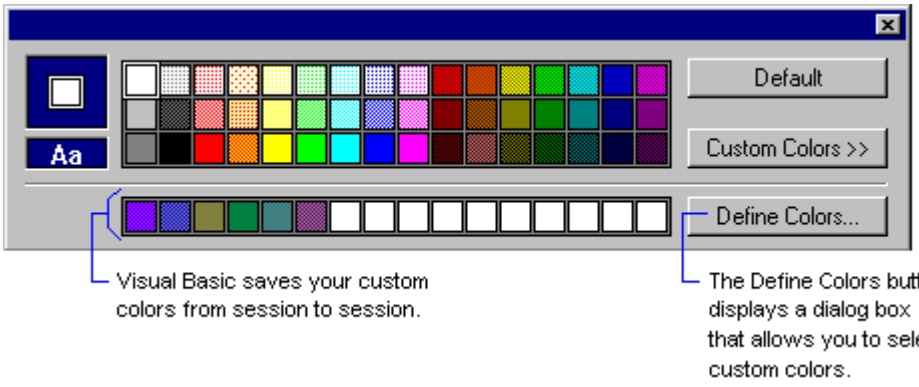
You set the color properties of objects in the Properties window.



Using Color and Graphics

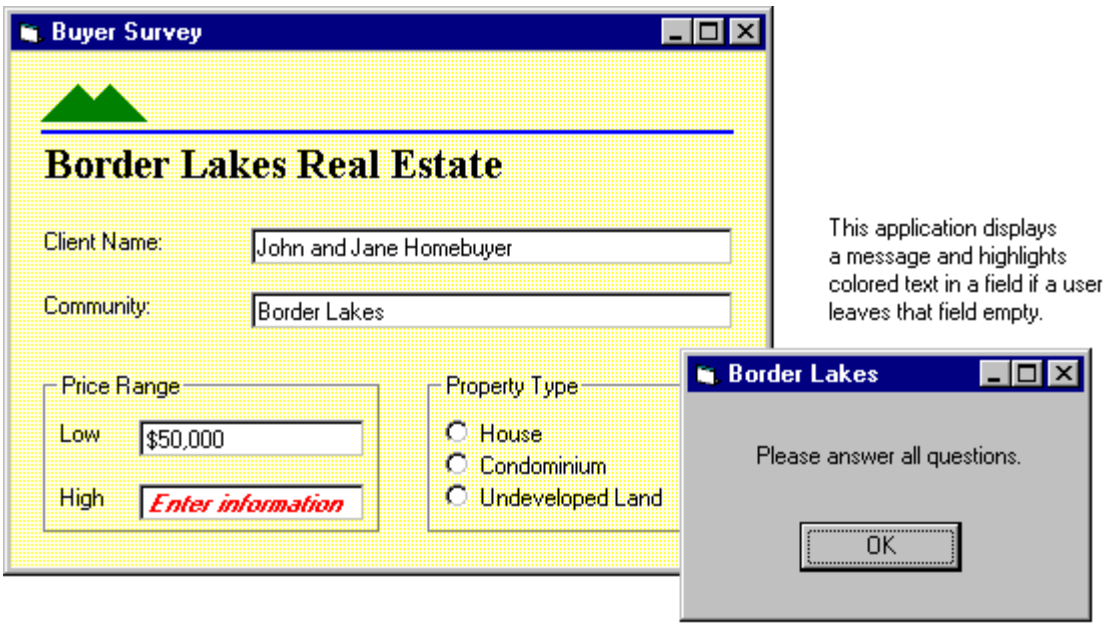
You can define your own colors and add them to the Color palette.

You can have up to 16 custom colors at one time.



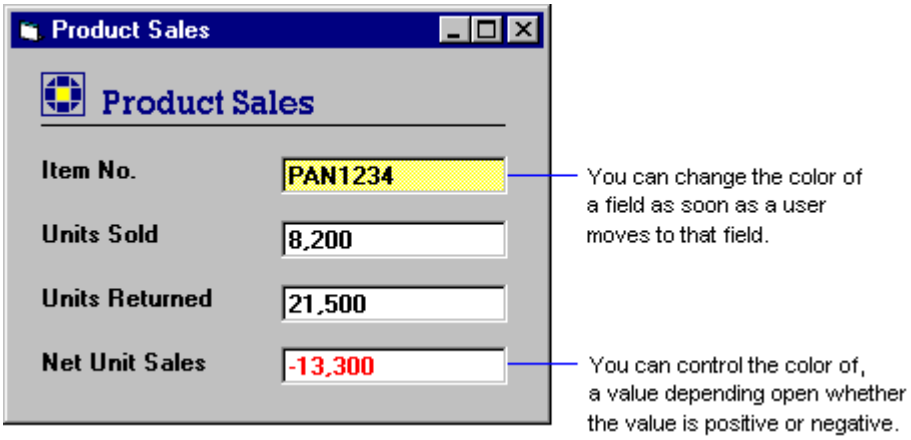
Using Color and Graphics

You can use Visual Basic code to display graphics in response to an event.



Using Color and Graphics

You can also use Visual Basic code to change the color of an object while your application is running. For example:



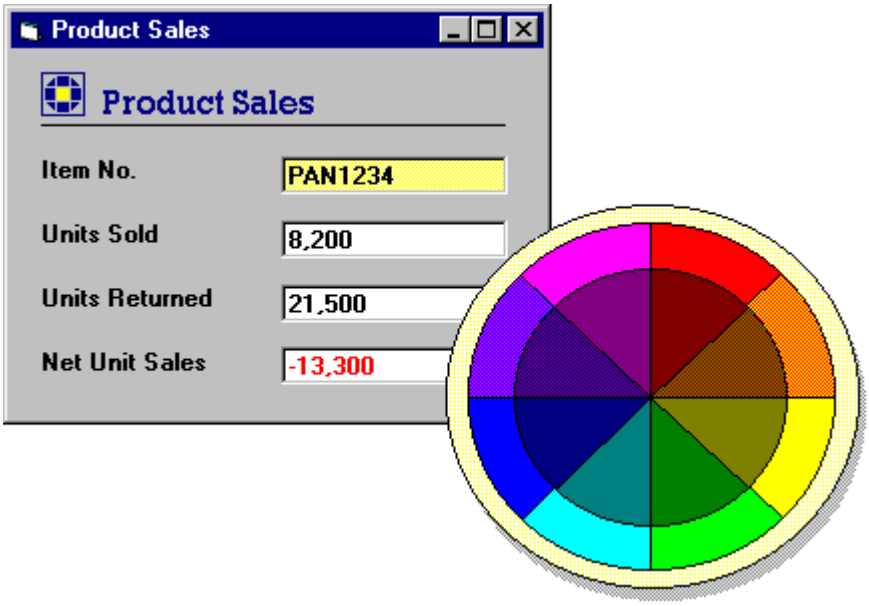
Using Color and Graphics

You can design custom command buttons for a form by attaching code to an image control or a picture box.

Each button on this form is an image control. When a user clicks a picture, Visual Basic runs the code attached to that control.

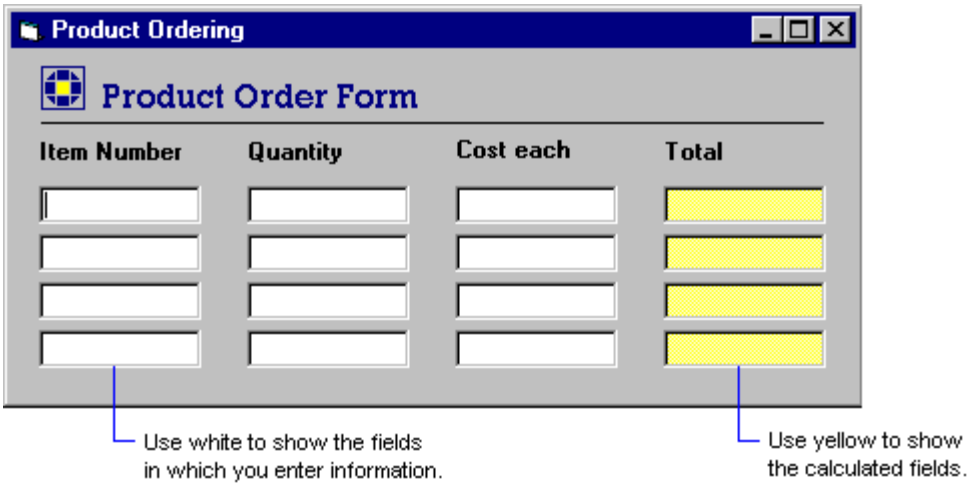
Using Color and Graphics

Using color and graphics effectively is an art. In the following screens, we offer some basic tips on the different ways you can enhance your applications.



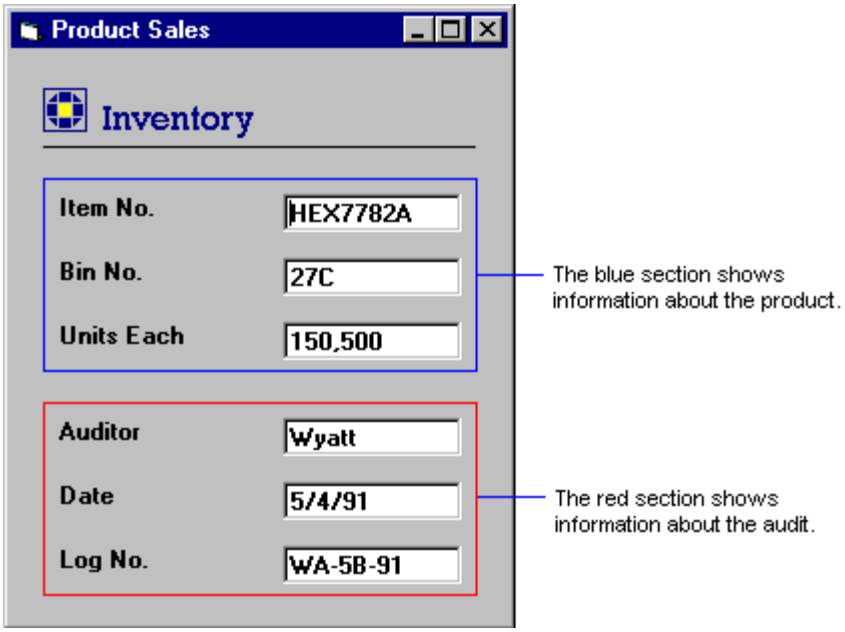
Using Color and Graphics

You can make your forms easier to use by applying the same color to similar types of fields.



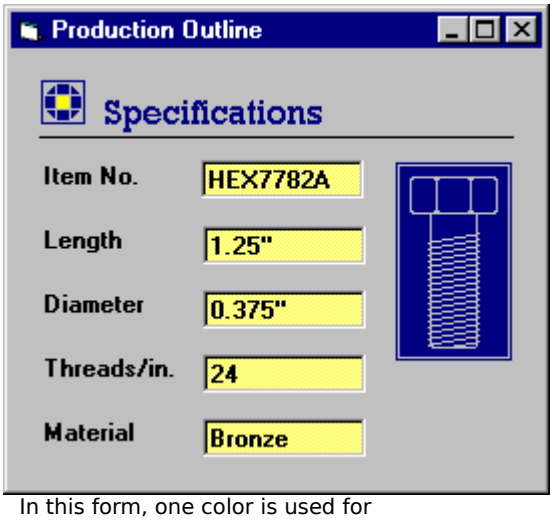
Using Color and Graphics

Similarly, you can use different colors to identify different sections of a form.

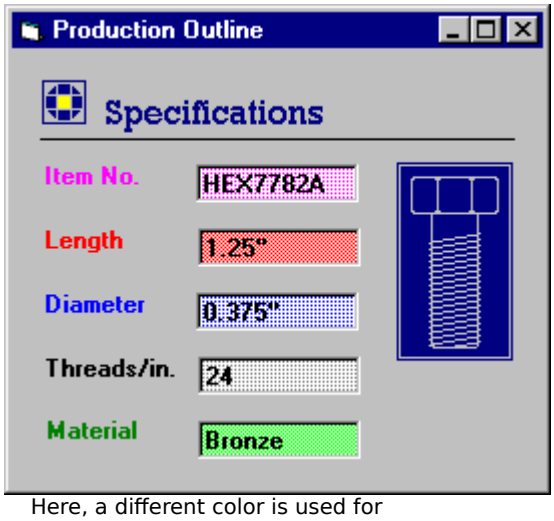


Using Color and Graphics

As you design your forms, try to avoid using too many colors. You'll have better results if you pick one group of colors and stick to them.



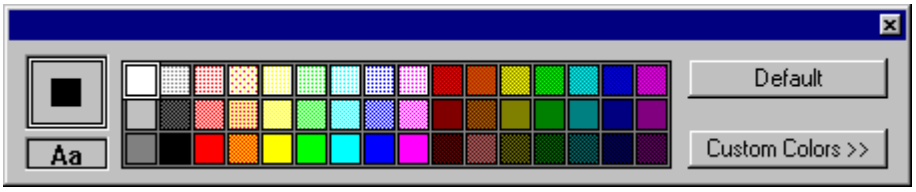
In this form, one color is used for the background for all fields.



Here, a different color is used for each field. Notice how the form appears too busy and distracting.

Using Color and Graphics

When designing applications that rely extensively on color, you may find it helpful to use a color guide or work with a designer to make the best use of color in your applications.



Using Color and Graphics

This lesson discussed using color and graphics in your Visual Basic applications.

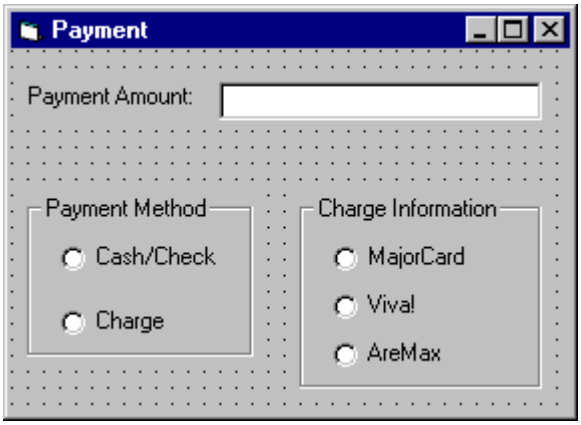
For more information on graphics, see Chapter 15, "Creating Graphics for Applications," in the *Programmer's Guide*.

Using Color and Graphics

Working with Forms and Controls

This lesson will introduce:

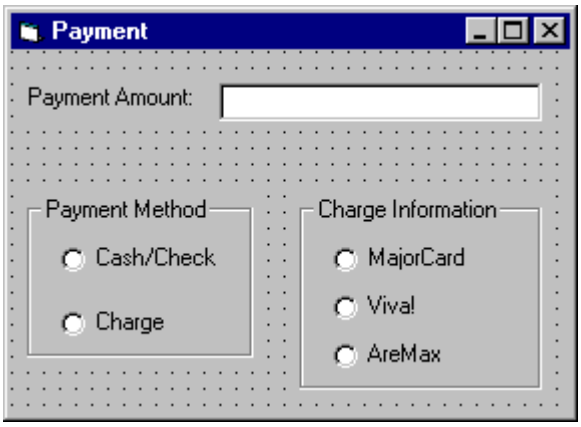
- Forms
- Adding controls
- Setting properties
- Creating event procedures



Working with Forms and Controls

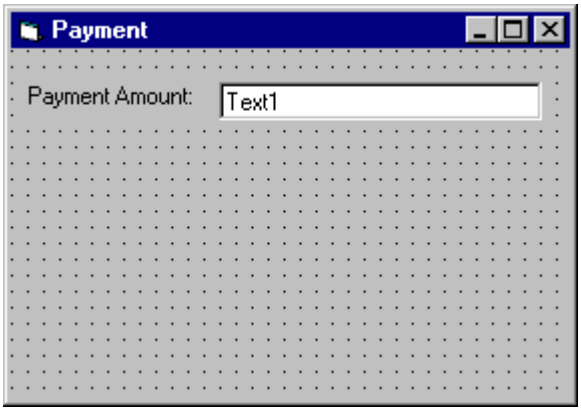
A **form** is a window or dialog box that you create with Visual Basic.

When you create a form, it is initially displayed in a default location and size. You can change both the location and the size of the form to suit the design of your application.



Working with Forms and Controls

You draw graphical objects called **controls** on a form to accept user input or display output.
You draw a control by selecting one of the tools from the Toolbox.



Working with Forms and Controls

Each control in the Toolbox has built-in capabilities. For example, even if you don't write code for text boxes, users can still cut and paste text in them.

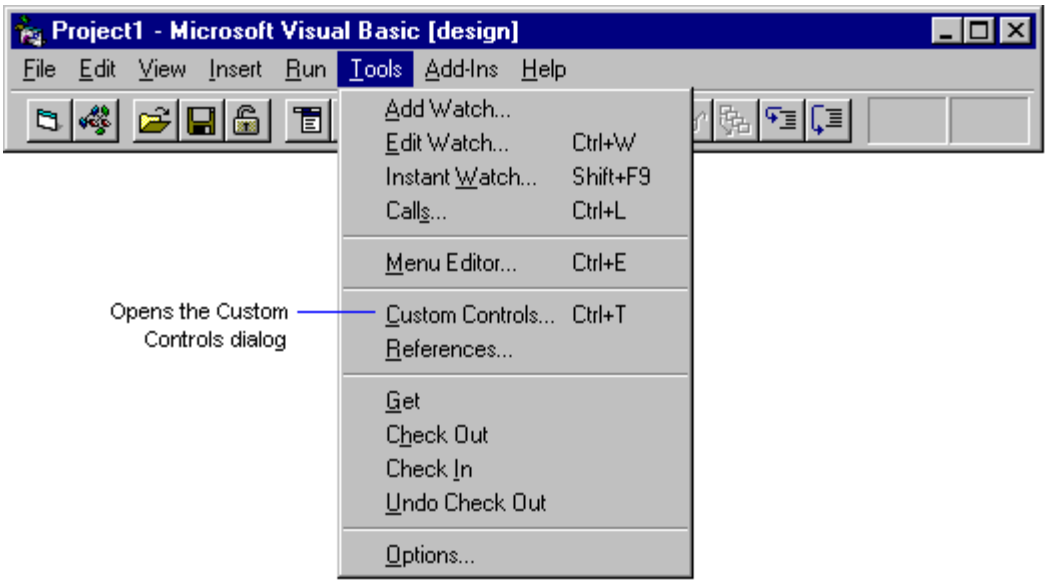
To learn more about each control:

- Click the control in the Toolbox.

Working with Forms and Controls

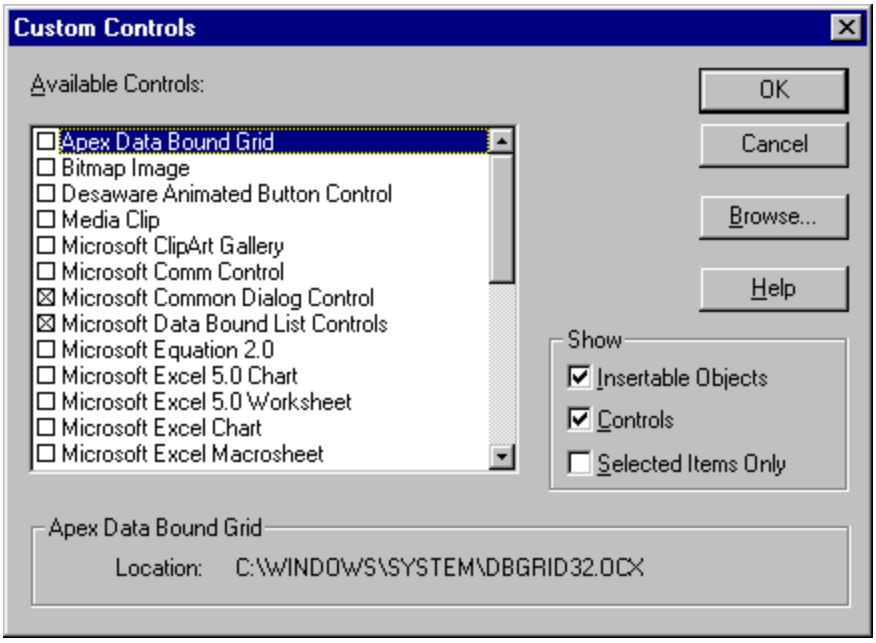
In addition to the controls that are built into Visual Basic, you can also add **custom controls** from Microsoft and other companies. Just as with the standard controls, custom controls have built in capabilities.

To add a custom control, select Custom Controls from the Tools menu.



Working with Forms and Controls

The Custom Controls dialog displays all available custom controls and insertable objects, such as a Microsoft Excel Chart, that can be added to your Toolbox. Select the check box next to each item you want to add.



Working with Forms and Controls

Every form and control has a predefined set of **properties**. These properties determine:

- Appearance  the color, size, and name of the object.
- Behavior  whether the user can move, size, minimize, or maximize the object.

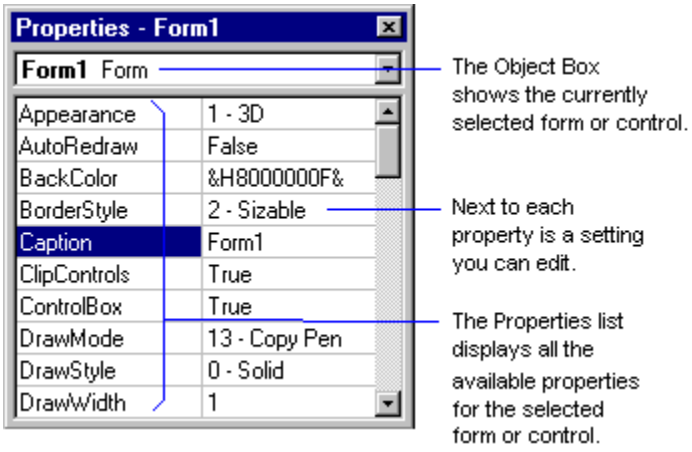
Working with Forms and Controls

You set the initial values for properties using the Properties window.

The Properties list displays all the available properties for the selected form or control.

Next to each property is a setting you can edit.

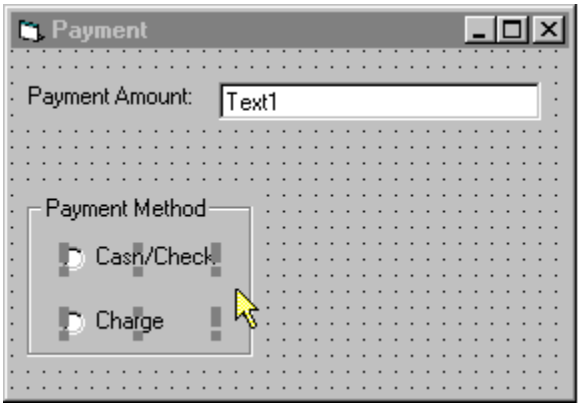
The Object box shows the currently selected form or control.

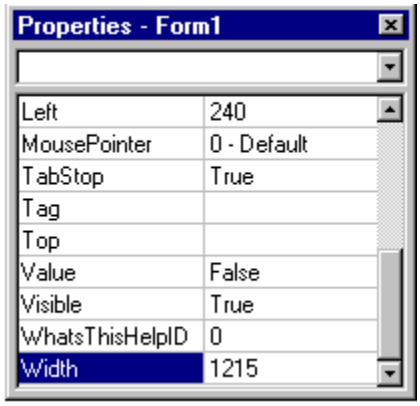


Working with Forms and Controls

If you want to set the same properties for several controls, you can select the group, then set the common properties all at once.

- Click and drag to select the group of controls you want to set properties for.
- The Properties window will then display the properties common to the controls you've selected.

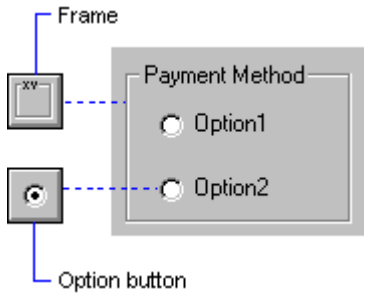




Working with Forms and Controls

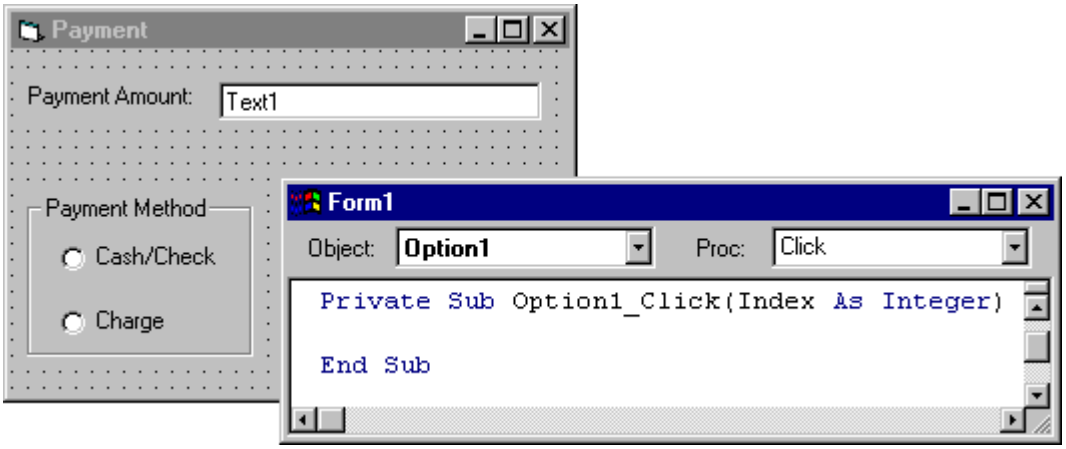
You can use frames to visually group or separate some of the controls on your form.

To put an object inside of another object, you must create the container first, and then place the object inside it.

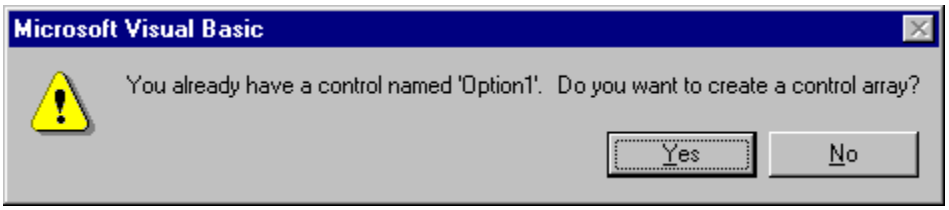


Working with Forms and Controls

If you want several controls to share the same code, you can put those controls in a **control array**. A control array is a group of the same type of controls, such as option buttons, that share the same name, and can also share the same event procedure.

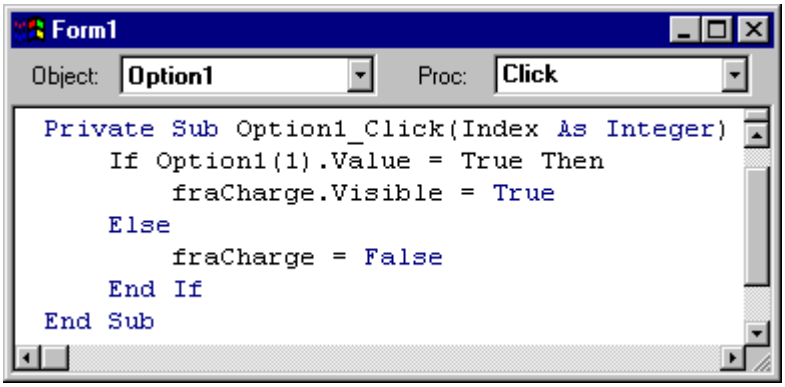


When you create a duplicate name, Visual Basic asks if you want to create a control array.



Working with Forms and Controls

To refer to an individual item in a control array, you use an index value. In the Click event for the option button array, for example, notice the parameter index **As Integer**. This indicates that the first option button will be referred to as Option 1(1), the second as Option1(2), and so on.



Working with Forms and Controls

You can run any application you're working on by choosing Start from the Run menu or by clicking the Start button () on the toolbar.

Working with Forms and Controls

This lesson introduced you to forms, controls, properties and control arrays. The next lesson covers adding menus to forms.

For more information on designing forms, see Chapter 2, "Your First Visual Basic Application," and Chapter 3, "Creating and Using Controls," in the *Programmer's Guide*.

Working with Forms and Controls

The pointer is used to manipulate existing controls on your form. With the pointer, you can select, move, and size forms and controls.

When you select a tool from the Toolbox, your mouse pointer changes to a cross hair. After you create a control, your mouse pointer changes back to the pointer.

A label displays text that cannot be changed by the user. The text can be changed by the application at run time, however, in response to an event.

You can use a label to display information for the user. For example, you could show the time or the progress of a file-copying operation.

The file list box displays all the files in a given directory. You can display a list of files based on file attributes, and users can select a file from the list.

You can use a file list box as part of a dialog box used to open a file.

The directory list box displays the directories and paths of the current drive at run time. You can use this control to display a hierarchical list of directories from the root to the selected path.

You can use the directory list box as part of a dialog box used to open a file.

The drive list box finds and switches among valid drives at run time. It displays a list of the user's valid drives.

You can use the drive list box control as part of a dialog box used to open a file.

A timer can be used to cause actions to occur at regular intervals while your application is running. For example, you can use a timer to update a clock display in your application.

Vertical scroll bars allow the user to move vertically within lists or through large amounts of information. They also provide a graphical way of displaying and setting values.

For example, you could add vertical scroll bars to a temperature conversion application to show temperatures rising and falling.

Horizontal scroll bars allow the user to move horizontally within lists or through large amounts of information. They also provide a graphical way of displaying and setting values.

For example, you could use a horizontal scroll bar to set the volume for an application that plays music or to show how much time has elapsed.

A list box contains a scrollable list from which the user can select one or more items.
For example, you can display a list of names in a list box and have the user choose from the list.

A combo box combines a text box and a list box. The user can either type in the text box or select items from the list box. There are three types of combo boxes: drop-down combo boxes, simple combo boxes, and drop-down list boxes.

You can use drop-down combo boxes and list boxes to save room on your forms.

An option button is used to select an option, usually from among a group of option buttons. When an option button is selected, the button has a black center. Unlike a group of check boxes, only one option button can be selected from a group.

For example, you might use an option button group to indicate the method of payment (cash, check, or credit card) for an invoice.

A check box is used for an option that can be turned on and off. When the user selects the option, an X is displayed in the check box. Check boxes can be used to give the user a yes/no or true/false option.

You can use check boxes for options that users can select in any combination, such as bold, italic, and underline formatting.

A command button carries out an action when the user chooses it. Typically, the user chooses a command button by clicking it or by pressing the SPACEBAR when it is selected.

OK and Cancel buttons are examples of command buttons. Or you might create a command button that a user can choose to open another form.

A frame provides a graphical and functional grouping for controls. Objects are put into frames to separate them visually from other controls.

You can place option buttons in a frame to create an option button group.

A text box is an area in which text can be entered by the user or displayed by the application. A text box can contain one or more lines of text and can be scrollable.

For example, in a security-system application, you might use a text box to prompt a user for a password.

The shape control displays a circle, square, oval, rectangle, rounded rectangle, or rounded square. Unlike graphics methods, the shape control is visible at design time.

For example, you can use shape and line controls to create a graphic of a building layout.

An image control is a graphical control that can display a picture. It is like a picture box control but uses fewer resources, repaints faster, and supports only some of the picture box properties, methods, and events.

For example, you can place a bitmap of your company logo in an image control and display text about the company when the user clicks it.

A line control displays a horizontal, vertical, or diagonal line. You can use line controls to draw lines on forms. Unlike the Line method, line controls are visible at design time.

You can use lines to display callouts on a graphic or to divide a form into sections.

A picture box control is used to display graphics on your form and to draw graphics in code. It can display a bitmap, icon, or metafile. As much of the picture as can be displayed within the picture rectangle will be shown.

You can create animation in a picture box by manipulating the graphics properties and methods.

The OLE container control lets you display data from another Windows-based application in your Visual Basic application.

At run time, you can edit the data in an OLE container control from within the application in which it was created. When you finish your edits, you close the application, and the updated data is displayed in the OLE container control on your form.

The grid control displays a series of rows and columns. At the intersection of a row and a column is a cell. A cell can contain text or graphics.

You can use a grid control to display a table of information.

The common dialog custom control allows you to display several commonly used dialog boxes: Open, Save As, Print, Color, and Font.

When you draw a common dialog control on a form, it automatically resizes itself. Like the timer control, the common dialog control is invisible at run time.

The data control lets you create applications to display, edit, and update information from many types of existing databases. You can use other data-aware controls with the data control to display information from the current record in a database.

Visual Basic implements data access by incorporating the same database engine that powers Microsoft Access.

The data-bound combo box is a data-aware combination list box and text box. The list can be filled automatically from a data control. The user can either choose an item from the list, or enter a value in the text box.

You can use the data-bound combo box to provide read-write access to a specified text data field selected from the list.

The data-bound grid is composed of multiple records. The grid can be filled automatically from a data control. The user can either choose an item from the grid, or enter a value in the new record.

You can use the data-bound grid to provide read/write-access to a specified recordset.

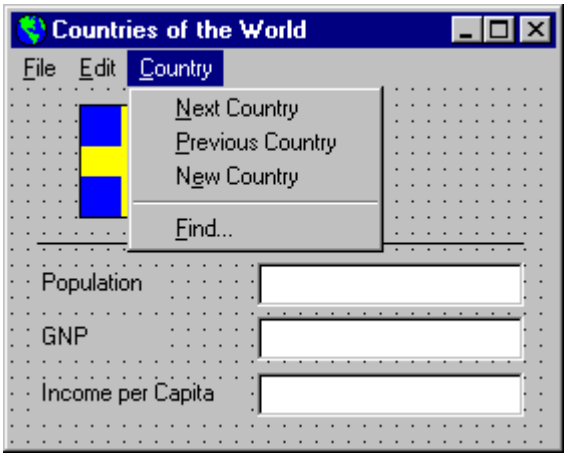
The data-bound list box is used to display a list of items from which the user can choose one. The list can be filled automatically from a data control.

You can use the data-bound list box to provide read-write access to a specified data field selected from the list.

Adding Menus

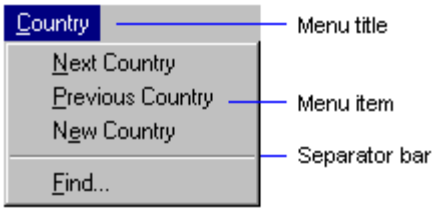
This lesson covers:

- The Menu Editor
- Menus and menu commands
- Menu design guidelines



Adding Menus

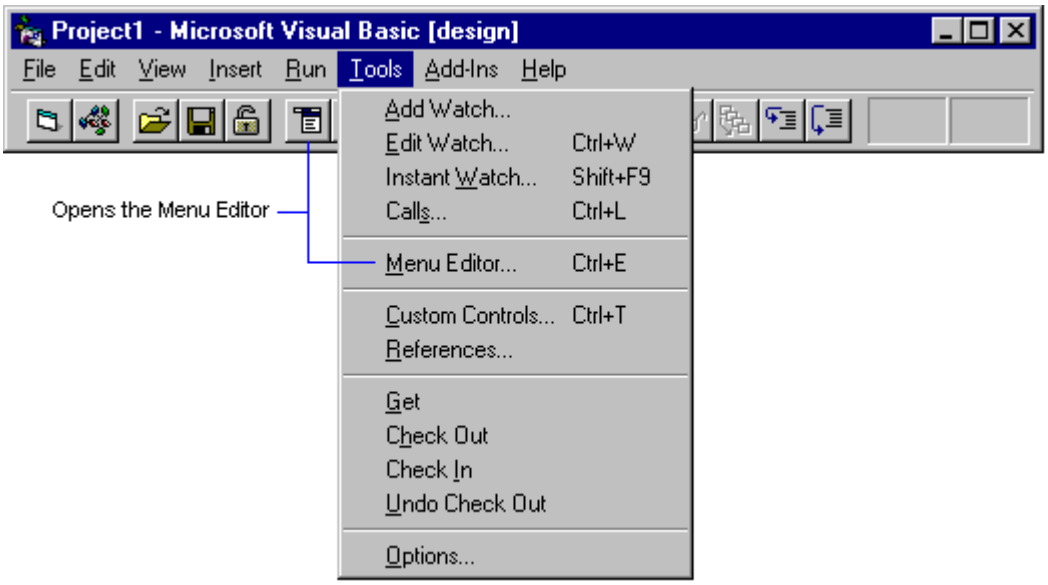
Menus consist of menu titles, menu items, and separator bars. Every part of a menu is a menu control.



Adding Menus

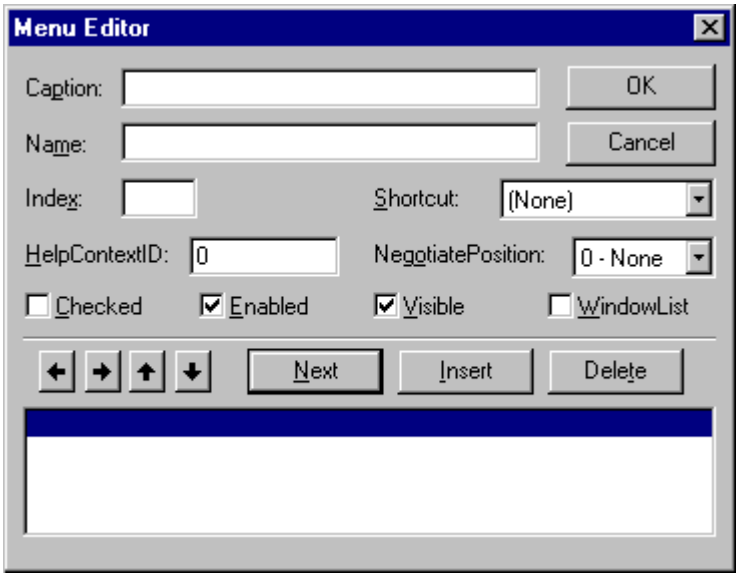
You use the Menu Editor to create menus for your form.

To open the Menu Editor, you first switch to the form to which you want to add a menu. Then, choose Menu Editor from the Tools menu or click the Menu Editor button on the Toolbar.



Adding Menus

A menu is a control, like a text box or a command button. Like other controls, a menu has a predefined set of properties and events.



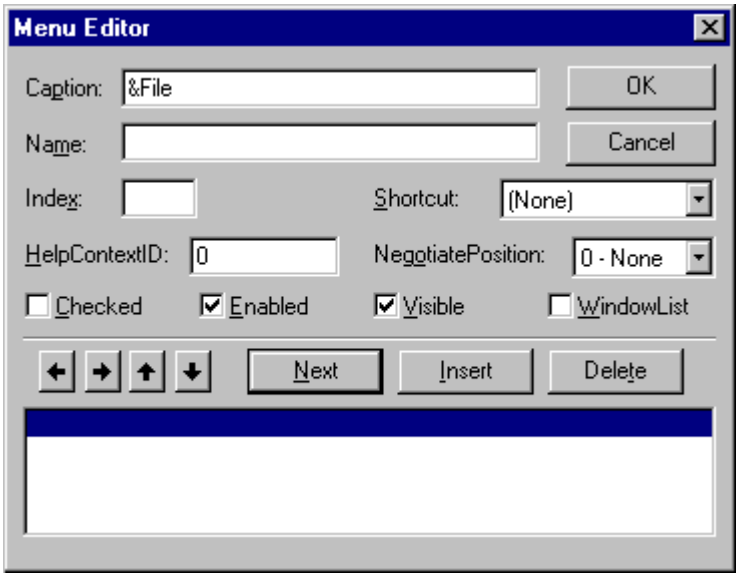
Adding Menus

You use the Menu Editor to create menu controls and set their properties:

- Caption－specifies a menu title, such as File or Edit, or an item on a menu, such as Open or Cut.
- Name－the name used to refer to the menu control in code.
- Index  a numeric value that uniquely identifies the menu control if it is part of a control array.

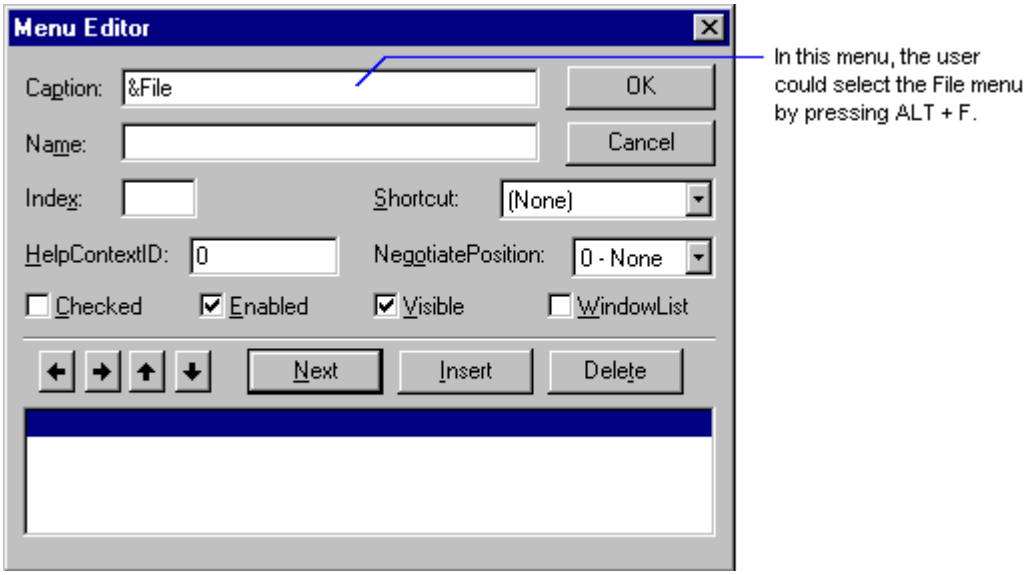
Adding Menus

The text you enter in the Caption text box defines the menu name. This is the text that appears on the menu bar.



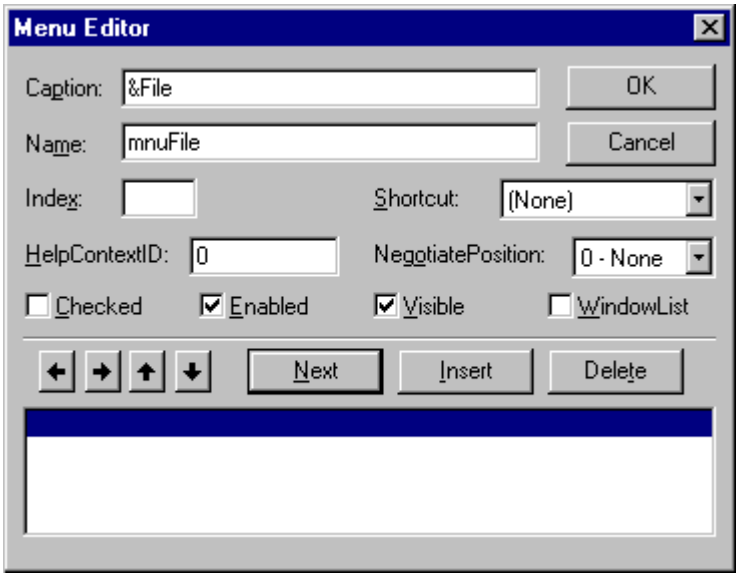
Adding Menus

Inserting an ampersand (&) before a letter gives the user keyboard access to the menu. At run time, this letter will be underlined.



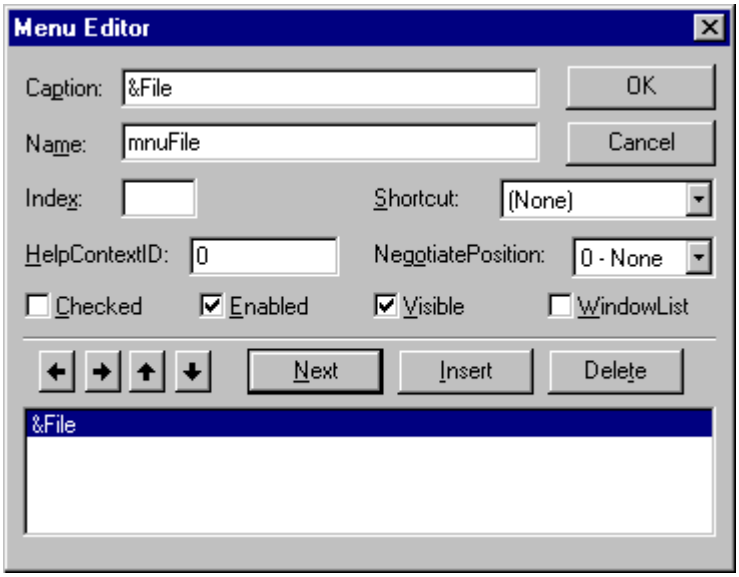
Adding Menus

The text in the Name box defines the menu's Name property. This is used to refer to the File menu in code.



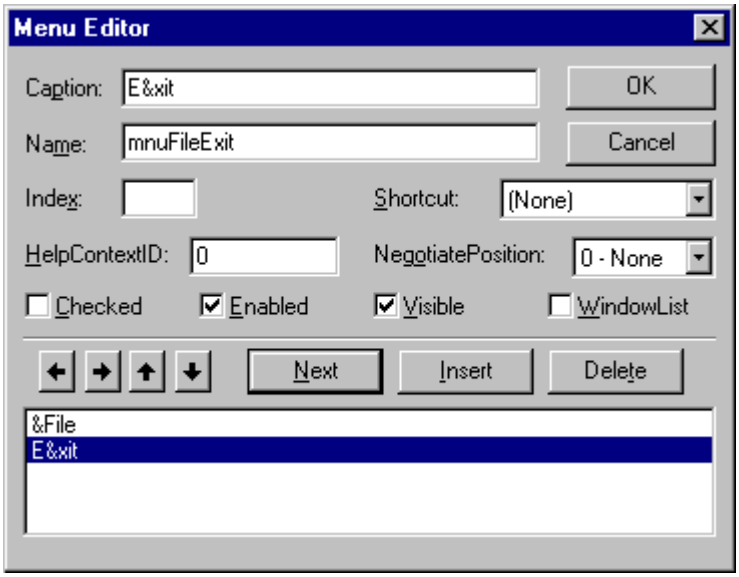
Adding Menus

After you enter the caption and name, you can click the Next button or press ENTER to create the File menu control.



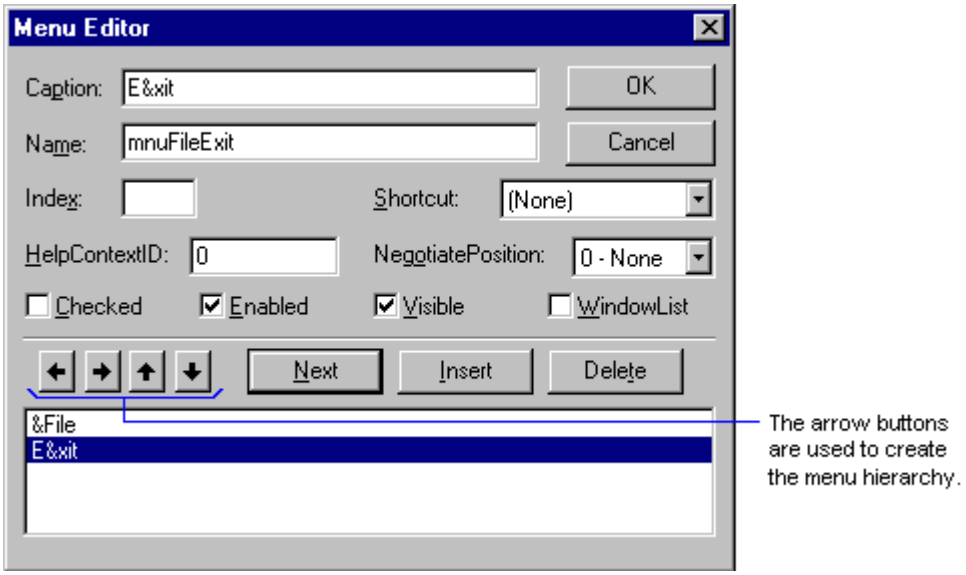
Adding Menus

The highlight then moves to a new line, and the text boxes are reset to accept the next caption and name.



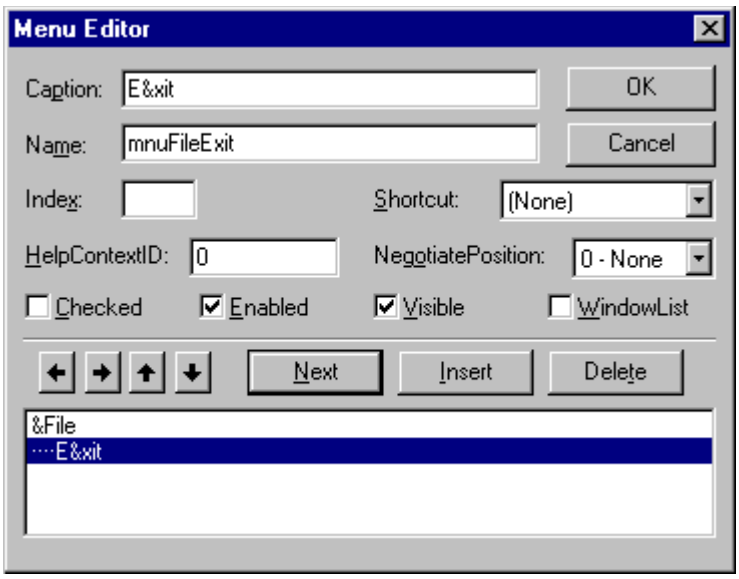
Adding Menus

To distinguish menu items from menu titles, you indent the menu items in the lower portion of the Menu Editor. To indent a menu item, select it and then click the right arrow button.



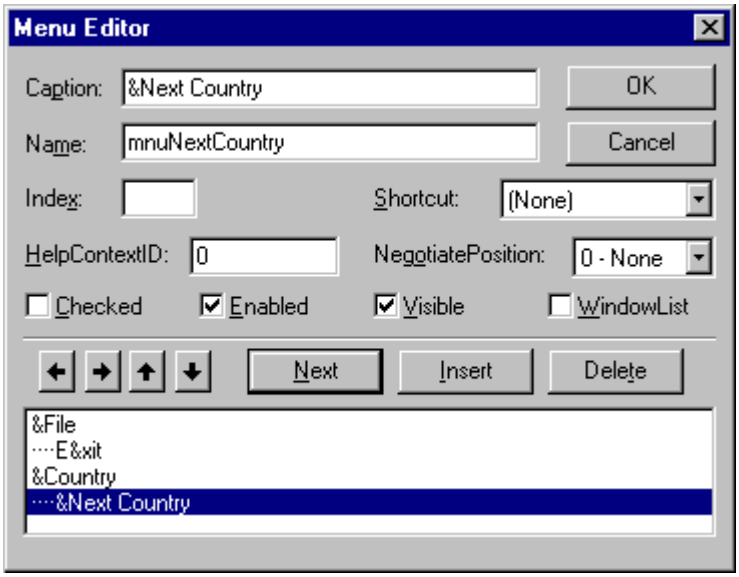
Adding Menus

The File menu now consists of one command  Exit.



Adding Menus

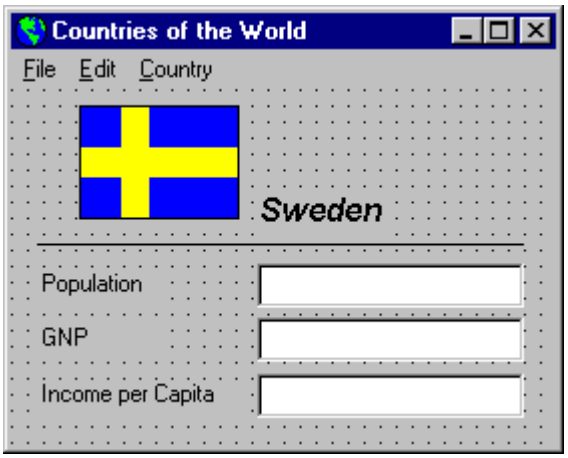
Now, we'll add the second menu to the menu bar Country and a command to the Country menu Next Country.



Adding Menus

When you close the Menu Editor, the menus are automatically displayed on the form.

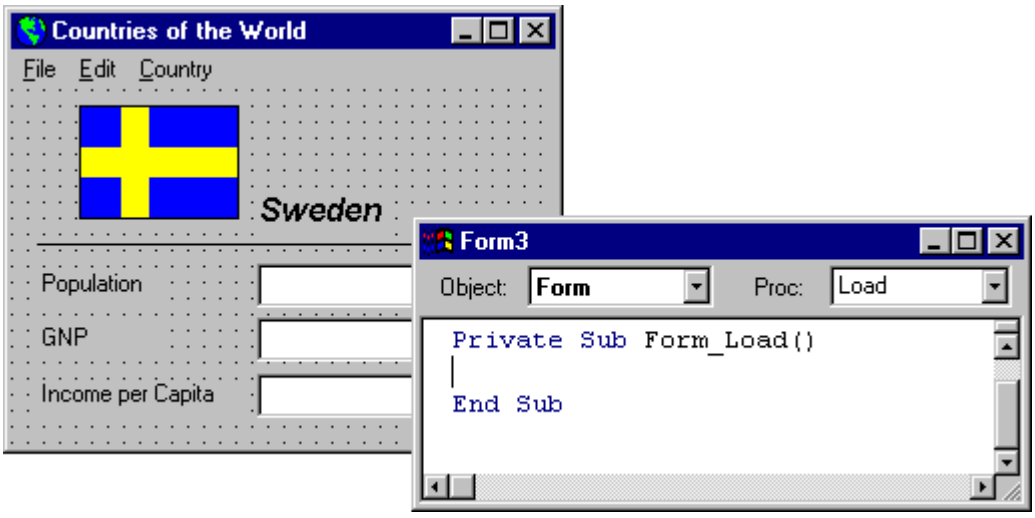
To define how each menu command responds to a Click event, you write an event procedure for each command.



Adding Menus

You write event procedures in the Code window.

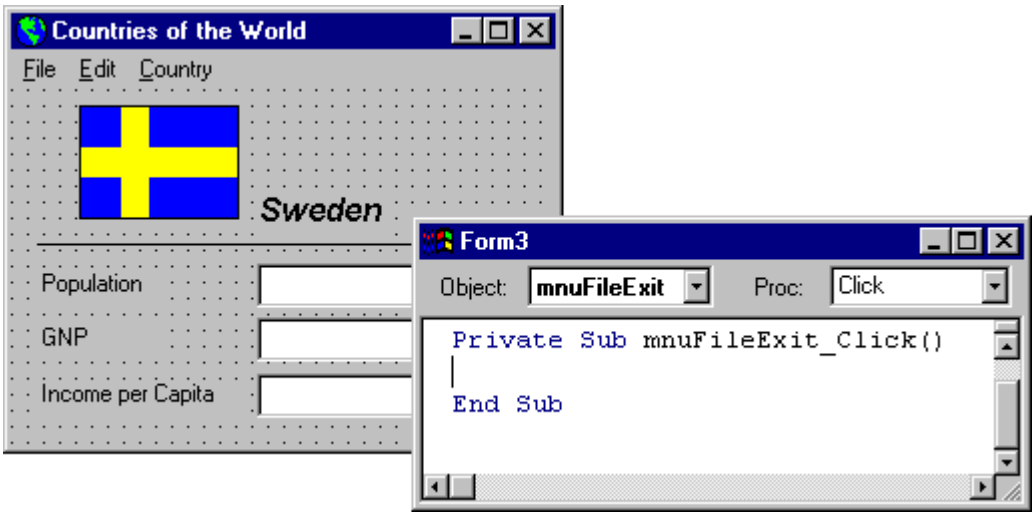
To open the Code window for a form, choose Code from the View menu, click the View Code button in the Project window, or press F7.



Adding Menus

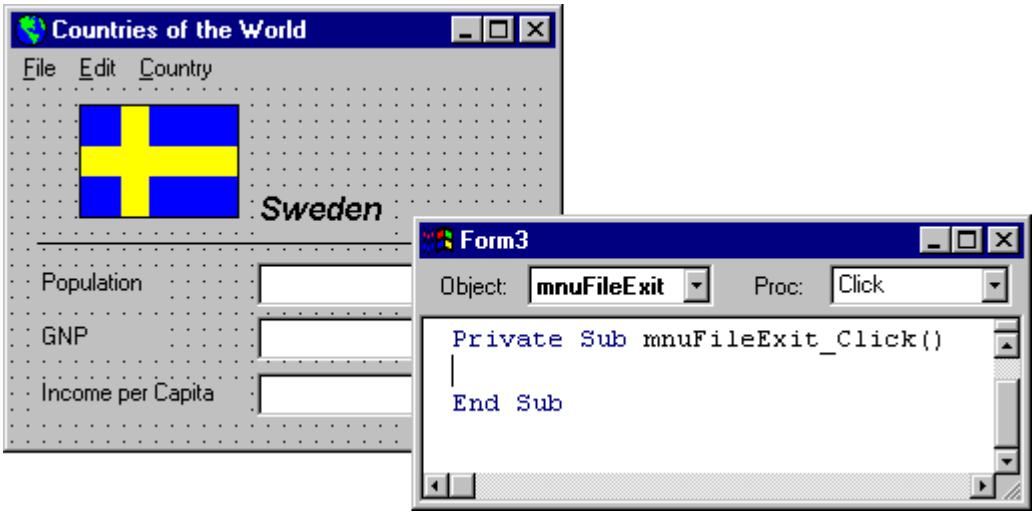
We entered mnuFileExit as the Exit command's Name property. The Name property is used to refer to a control in code.

The Click event procedure is specified, as menu commands respond only to Click events.



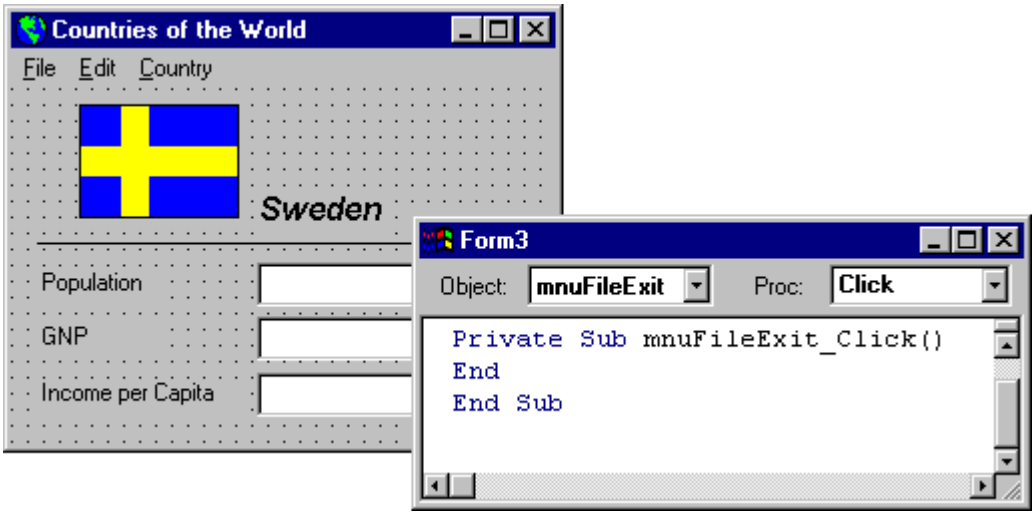
Adding Menus

We'll write code to make the application respond to the Exit command. The code goes between the Sub and End Sub statements in the Code window.



Adding Menus

The code we entered will end the Countries of the World application when the user chooses the Exit command from the File menu.



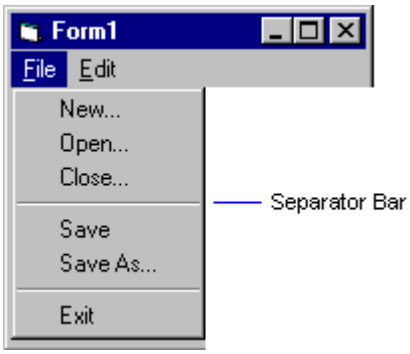
Adding Menus

When creating menus, follow these standard menu design guidelines:

- Group related commands on a menu in a way that will make sense to users of your application.

  For example, users familiar with Microsoft Windows may expect to find the New, Open, and Close commands together on the File menu. Look at existing Windows-based applications for examples.

- On long menus, separate groups of related commands with a separator bar. Separator bars are created by using a single hyphen (-) in the Caption box of the Menu Editor.



Adding Menus

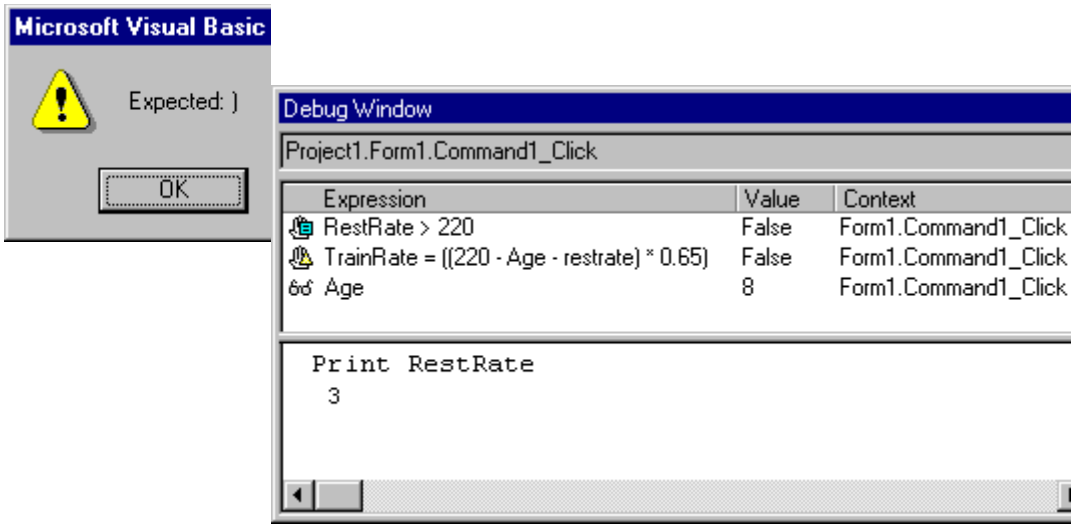
This lesson showed you how to create menus for your Visual Basic applications using the Menu Editor. For more information on creating menus, see Chapter 10, "Menus," in the *Programmer's Guide*.

Adding Menus

Debugging Your Application

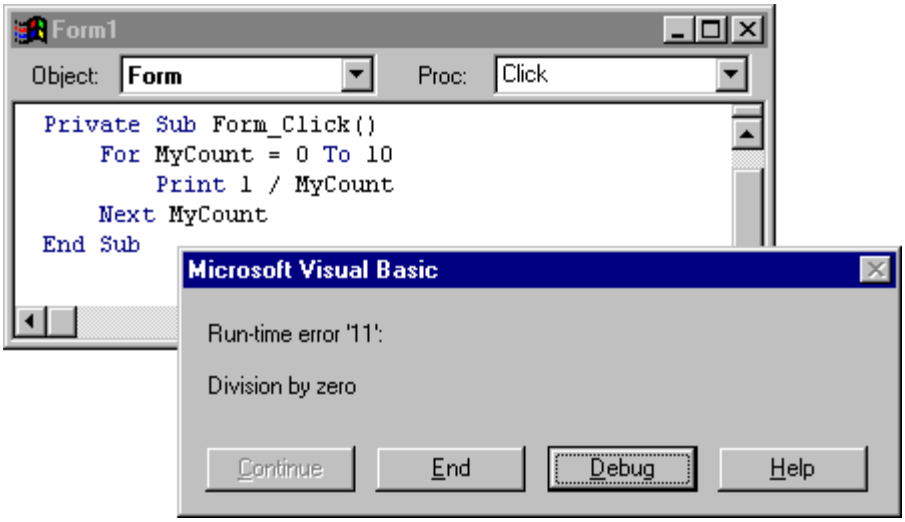
This lesson will cover:

- Compile errors
- Run-time errors
- Logic errors
- Visual Basic debugging tools



Debugging Your Application

Despite your best intentions, it's common to make mistakes when writing code.



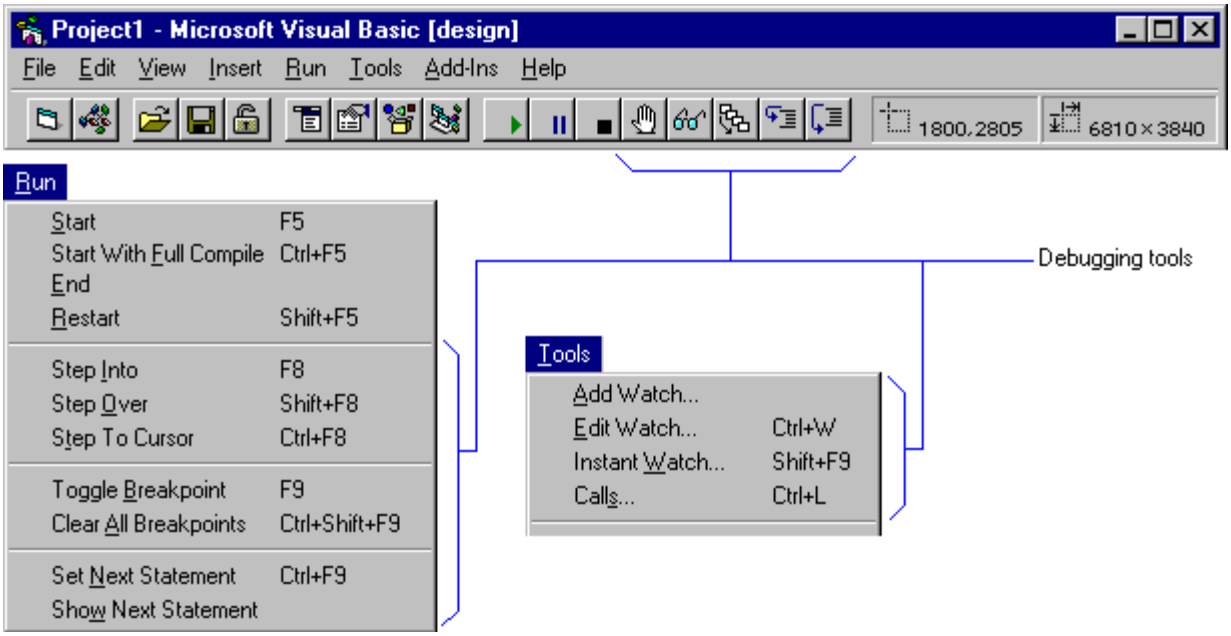
Debugging Your Application

There are three types of errors you can make when writing code:

- Compile errors  mistakes caused by incorrectly constructed code.
- Run-time errors  mistakes that Visual Basic can detect while your program is running.
- Logic errors  mistakes that cause an incorrect result, or that prevent your program from running as expected.

Debugging Your Application

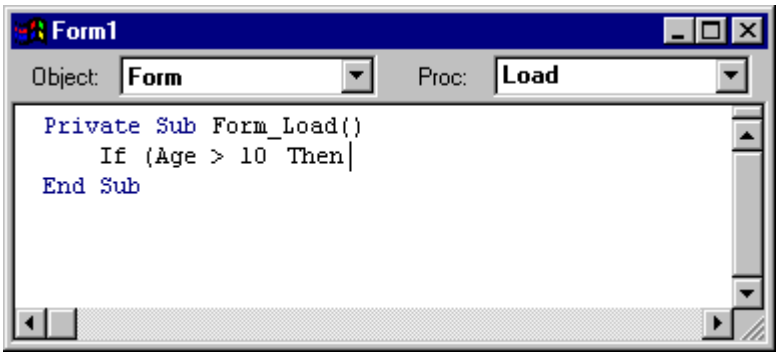
Visual Basic provides tools to help you detect these problems before you compile your application.



Debugging Your Application

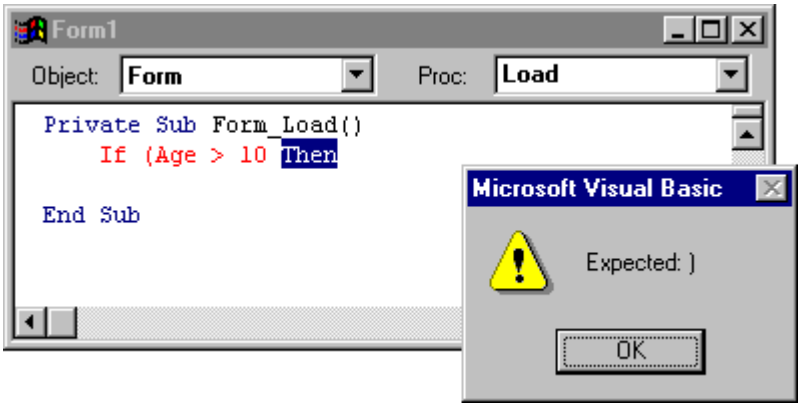
Compile errors

Compile errors are caused by code that violates the rules of the Visual Basic language. These include syntax errors.



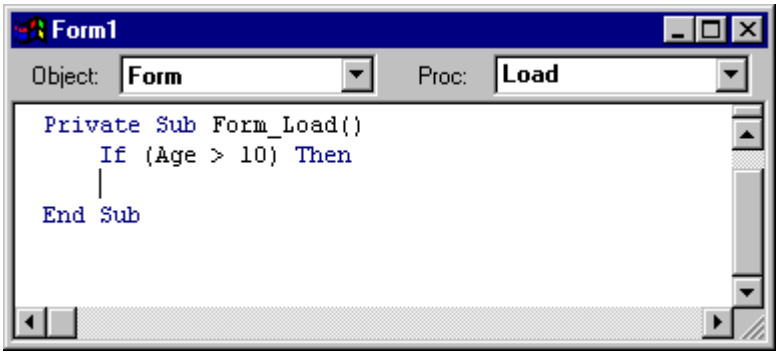
Debugging Your Application

When you enter code in a Code window, Visual Basic checks the syntax as you move off a line. If you've made an error, Visual Basic displays an error message.



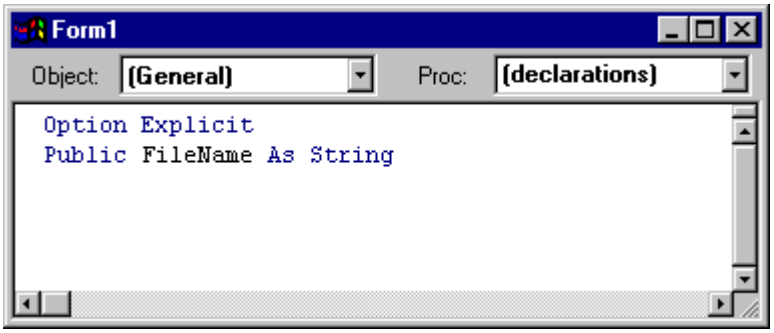
Debugging Your Application

You can either go back and correct your mistake, press ESC and return to the error later, or press F1 to get more information on the error.



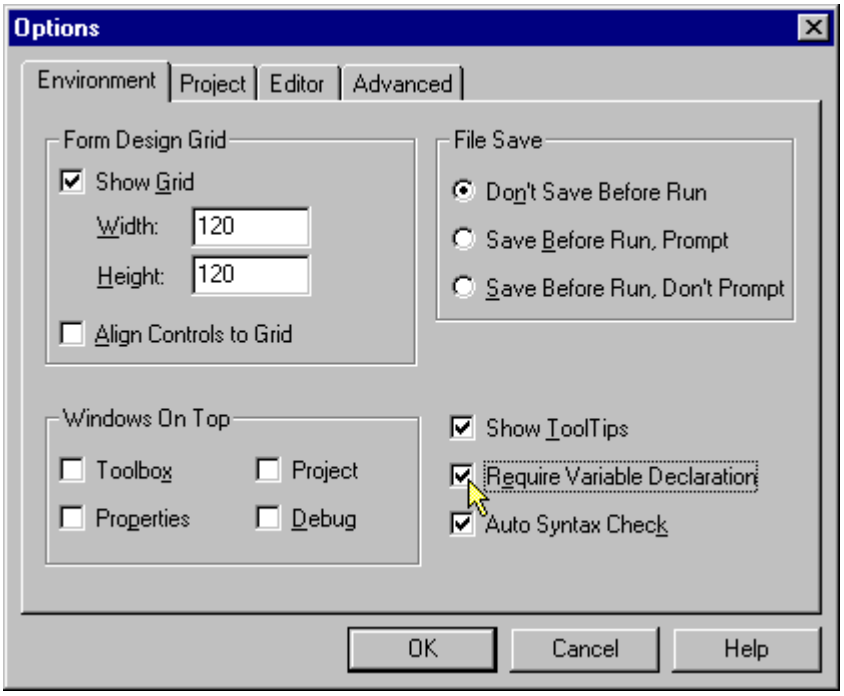
Debugging Your Application

One way to avoid problems caused by mistyped variable names is to use the Option Explicit statement. The Option Explicit statement requires you to declare all variables prior to their use.



Debugging Your Application

You can have Visual Basic place the Option Explicit statement automatically in every module you create by choosing Options from the Tools menu and selecting the Require Variable Declaration option on the Environment tab.



Debugging Your Application

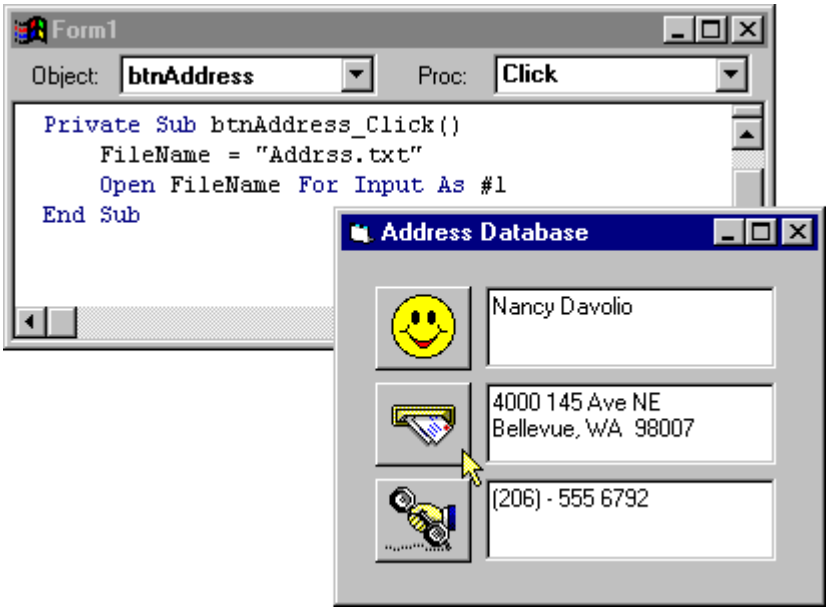
Run-time errors

To find run-time errors, you need to run your application.

Debugging Your Application

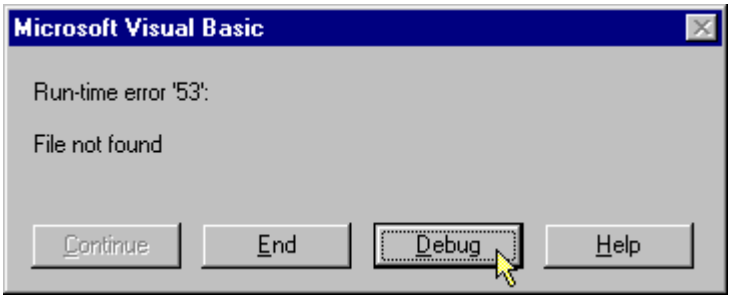
With your application running, you can see how it works.

If Visual Basic detects an error in your code, it halts execution.



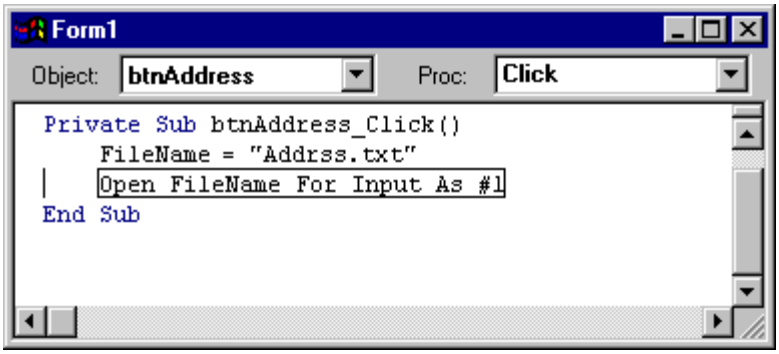
Debugging Your Application

When an error is encountered, an error message displays the type of error encountered. Clicking the Debug button will take you to the Code window where you can view the code that caused the error.



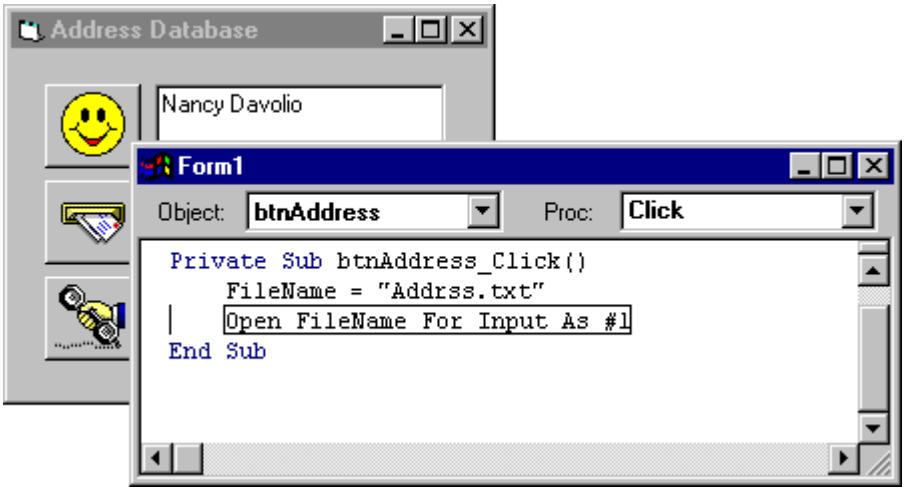
Debugging Your Application

The Debug window displays the code with a box around the line that caused the error.



Debugging Your Application

In this case, the filename was not found because it was not entered correctly. You can fix the error and then continue running the application.



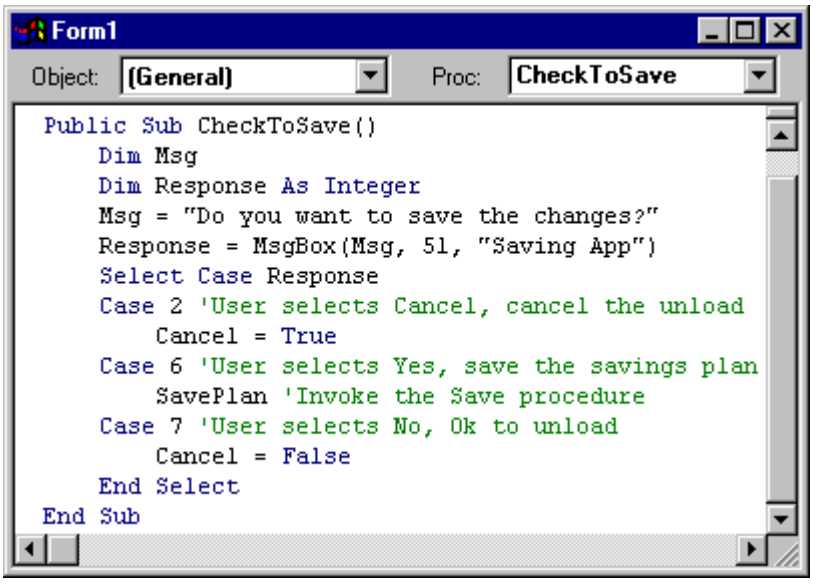
Debugging Your Application

Some changes you make to your application, such as changing a constant or creating a new procedure, will require you to restart the application.

Debugging Your Application

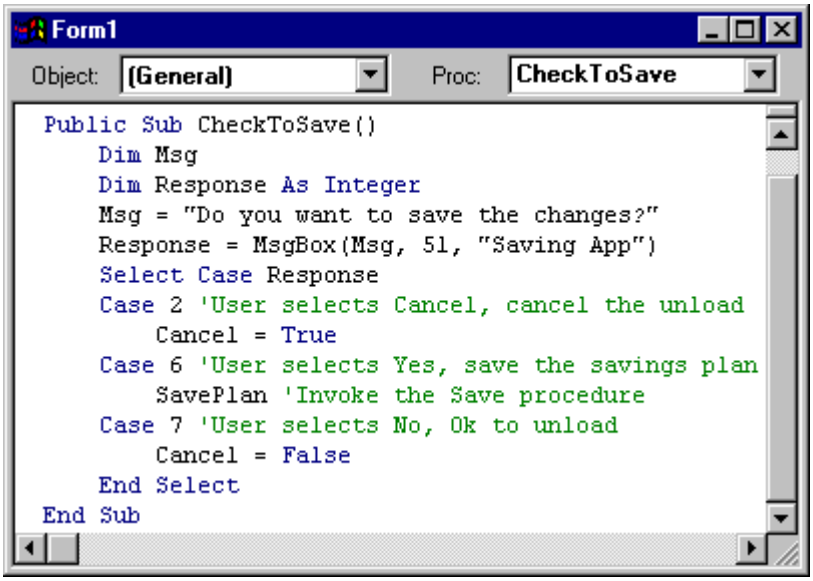
Logic errors

Logic errors can be harder to find. When your application runs, but you get results that aren't what you expect, you've most likely made an error in logic.



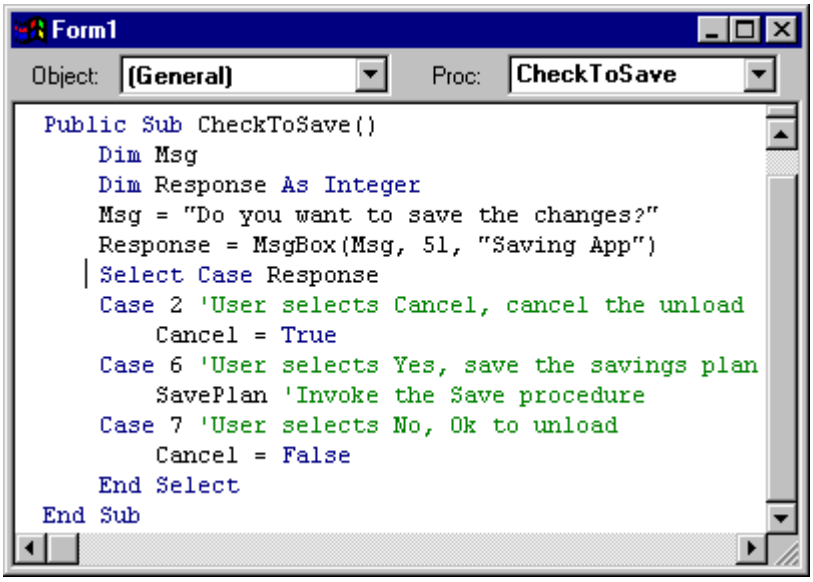
Debugging Your Application

It could be as simple as a mistyped or undeclared variable name, or as complicated as a statement whose syntax is correct but whose outcome isn't what you expect.



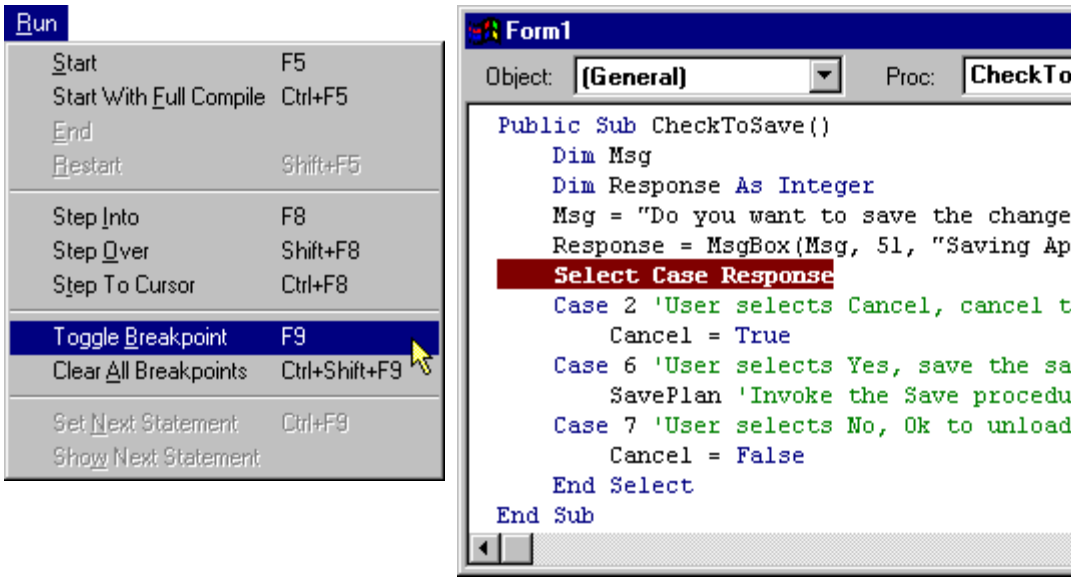
Debugging Your Application

Setting breakpoints to temporarily pause your application so you can step through your code may help you find problems with logic.



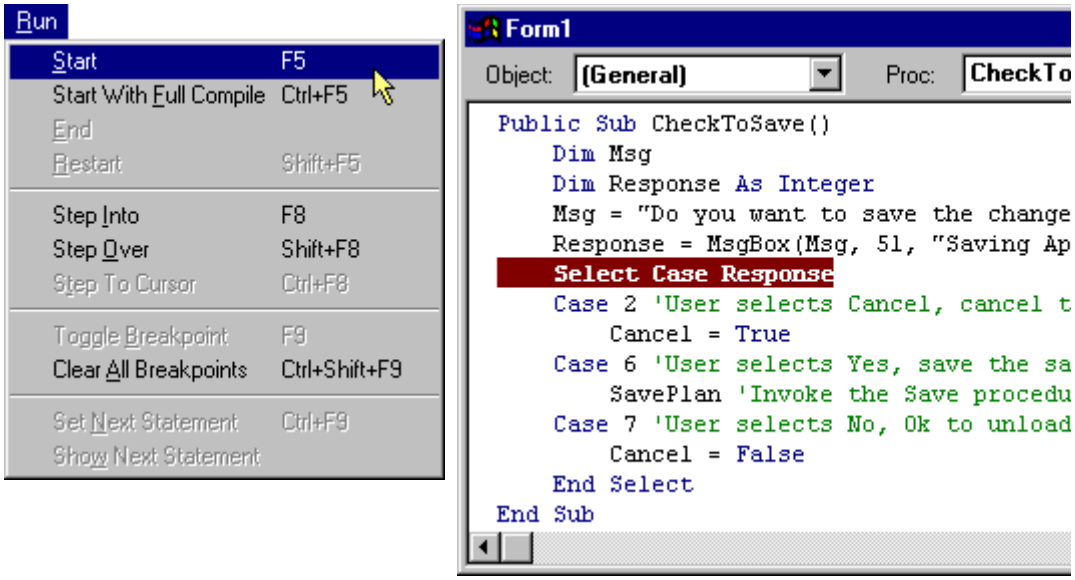
Debugging Your Application

To insert a breakpoint, position the cursor on the line of code where you want execution to stop. Then choose Toggle Breakpoint from the Run menu, or click the Toggle Breakpoint button on the toolbar.



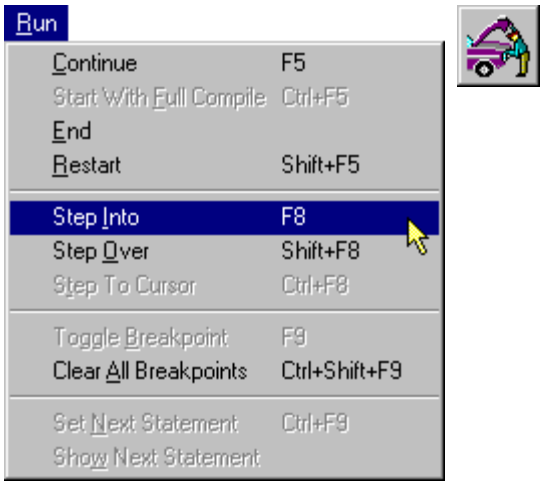
Debugging Your Application

When you start the application, it will run until it reaches the breakpoint.



Debugging Your Application

You can then step through code one statement at a time by choosing Step Into from the Run menu, pressing F8, or clicking the Step Into button on the toolbar to determine where there are problems.



Debugging Your Application

To continue running your code from a breakpoint, choose the Continue command from the Run menu, or click the Continue button on the toolbar.

Debugging Your Application

The Debug window allows you to examine code and watch expressions.
It can be accessed only in break mode.

Debugging Your Application

To ensure that your application is free of bugs, you need to test it in a variety of situations; for example:

- Use large and small values of numbers and strings. Often these reveal limitations in the application.
- Ask other people to work with the application. They may find problems with its design or discover bugs you didn't anticipate.
- If your application stores or retrieves data, check to see that the information is handled correctly.
- Test how your application handles errors.

Debugging Your Application

This lesson discussed three types of errors:

- Compile errors
- Run-time errors
- Logic errors

It also introduced the debugging tools available in Visual Basic.

For more information about debugging, see Chapter 20, "Debugging," in the *Programmer's Guide.*

Debugging Your Application