Using Menus in Your Application

Many simple applications consist of one form and several controls, but you can enhance your Visual Basic applications by adding menus. This section shows you how to create menus and use them in an application.

- Creating Menus with the Menu Editor How to use the Menu Editor.
- Menu Title and Naming Guidelines A discussion of naming conventions.
- Creating Submenus A discussion of submenus.
- Creating a Menu Control Array Using arrays to eliminate redundant menu code.
- Creating and Modifying Menus at Run Time Techniques for working with menus.
- Writing Code for Menu Controls A discussion of coding techniques for menus.
- Displaying Pop-up Menus Techniques for working with pop-up menus.
- Menus in MDI Applications A discussion of MDI menu techniques.

Send feedback to MSDN. Look here for MSDN Online resources.

Creating Menus with the Menu Editor

You can use the Menu Editor to create new menus and menu bars, add new commands to existing menus, replace existing menu commands with your own commands, and change and delete existing menus and menu bars.

To display the Menu Editor

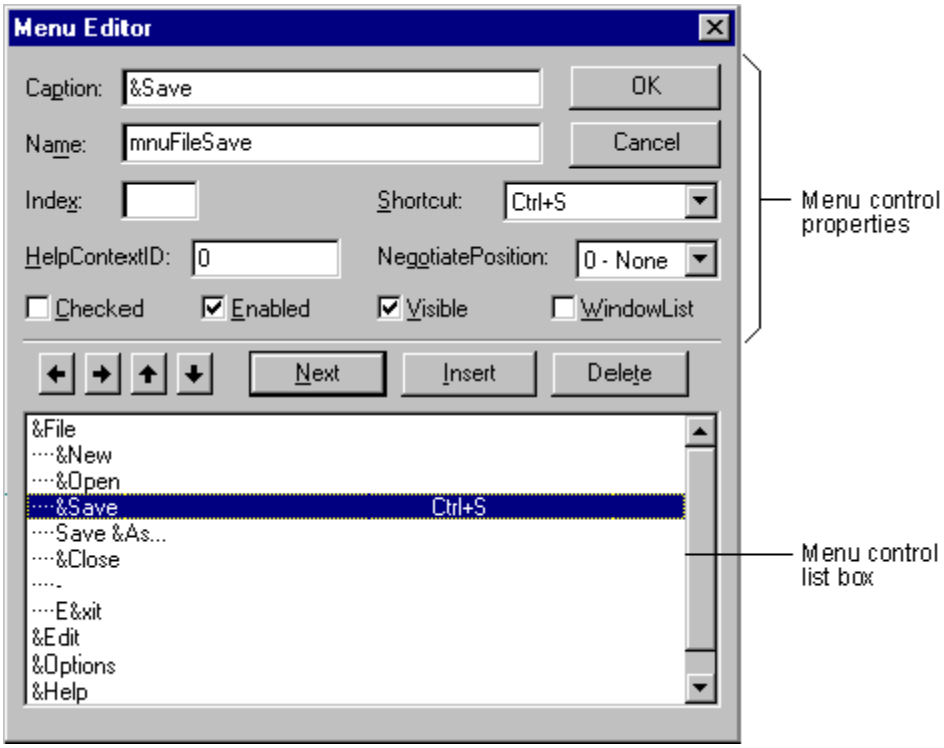
- From the **Tools** menu, choose **Menu Editor**.

  –or–

  Click the **Menu Editor** button on the toolbar.

This opens the Menu Editor, shown in Figure 6.7.

Figure 6.7 The Menu Editor



While most menu control properties can be set using the Menu Editor, all menu properties are available in the Properties window. The two most important properties for menu controls are:

- Name — This is the name you use to reference the menu control from code.
- Caption — This is the text that appears on the control.

Other properties in the Menu Editor, including Index, Checked, and NegotiatePosition, are described later in this chapter.

Using the List Box in the Menu Editor

The menu control list box (the lower portion of the Menu Editor) lists all the menu controls for the current form. When you type a menu item in the Caption text box, that item also appears in the menu control list box. Selecting an existing menu control from the list box allows you to edit the properties for that control.

For example, Figure 6.7 shows the menu controls for a File menu in a typical application. The position of the menu control in the menu control list box determines whether the control is a menu title, menu item, submenu title, or submenu item:

- A menu control that appears flush left in the list box is displayed on the menu bar as a menu title.
- A menu control that is indented once in the list box is displayed on the menu when the user clicks the preceding menu title.
- An indented menu control followed by menu controls that are further indented becomes a submenu title. Menu controls indented below the submenu title become items of that submenu.

- A menu control with a hyphen (-) as its Caption property setting appears as a separator bar. A *separator bar* divides menu items into logical groups.

Note A menu control cannot be a separator bar if it is a menu title, has submenu items, is checked or disabled, or has a shortcut key.

To create menu controls in the Menu Editor

1. Select the form.
2. From the **Tools** menu, choose **Menu Editor**.

   –or–

   Click the **Menu Editor** button on the toolbar.
3. In the **Caption** text box, type the text for the first menu title that you want to appear on the menu bar. Also, place an ampersand (&) before the letter you want to be the access key for that menu item. This letter will automatically be underlined in the menu.

   The menu title text is displayed in the menu control list box.
4. In the **Name** text box, type the name that you will use to refer to the menu control in code. See "Menu Title and Naming Guidelines" later in this chapter.
5. Click the left arrow or right arrow buttons to change the indentation level of the control.
6. Set other properties for the control, if you choose. You can do this in the Menu Editor or later, in the Properties window.
7. Choose **Next** to create another menu control.

   –or–

   Click **Insert** to add a menu control between existing controls.

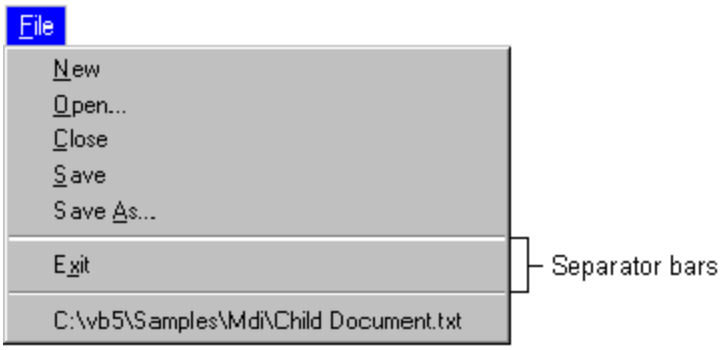
   You can also click the up arrow and down arrow buttons to move the control among the existing menu controls.
8. Choose **OK** to close the Menu Editor when you have created all the menu controls for that form.

   The menu titles you create are displayed on the form. At design time, click a menu title to drop down its corresponding menu items.

Separating Menu Items

A separator bar is displayed as a horizontal line between items on a menu. On a menu with many items, you can use a separator bar to divide items into logical groups. For example, the File menu in Visual Basic uses separator bars to divide its menu items into three groups, as shown in Figure 6.8.

Figure 6.8 Separator bars



To create a separator bar in the Menu Editor

1. If you are adding a separator bar to an existing menu, choose **Insert** to insert a menu control between the menu items you want to separate.

2. If necessary, click the right arrow button to indent the new menu item to the same level as the menu items it will separate.

3. Type a hyphen (-) in the **Caption** text box.

4. Set the **Name** property.

5. Choose **OK** to close the Menu Editor.

Note Although separator bars are created as menu controls, they do not respond to the Click event, and users cannot choose them.

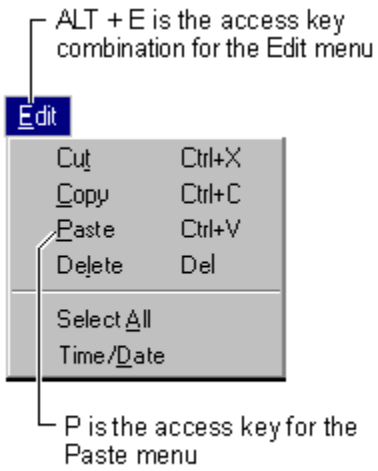
Assigning Access Keys and Shortcut Keys

You can improve keyboard access to menu commands by defining access keys and shortcut keys.

Access Keys

Access keys allow the user to open a menu by pressing the ALT key and typing a designated letter. Once a menu is open, the user can choose a control by pressing the letter (the access key) assigned to it. For example, ALT+E might open the Edit menu, and P might select the Paste menu item. An access-key assignment appears as an underlined letter in the menu control's caption, as shown in Figure 6.9.

Figure 6.9 Access keys



To assign an access key to a menu control in the Menu Editor

1. Select the menu item to which you want to assign an access key.
2. In the **Caption** box, type an ampersand (&) immediately in front of the letter you want to be the access key.

For example, if the Edit menu shown in Figure 6.9 is open, the following Caption property settings respond to the corresponding keys.

Menu control caption	Caption property	Access keys
Cut	Cu&t	t
Copy	C&opy	o
Paste	&Paste	p
Delete	De&lete	l
Select All	Select &All	a
Time/Date	Time/&Date	d

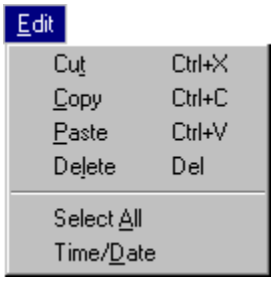
Note Do not use duplicate access keys on menus. If you use the same access key for more than one menu item, the key will not work. For example, if C is the access key for both Cut and Copy, when you select the Edit menu and press C, the Copy command will be selected, but the application will not carry out the command until the user presses ENTER. The Cut command will not be selected at all.

Shortcut Keys

Shortcut keys run a menu item immediately when pressed. Frequently used menu items may be assigned a keyboard shortcut, which provides a single-step method of keyboard access, rather than a three-step method of pressing ALT, a menu title access character, and then a menu item access character. Shortcut key assignments include function key and control key combinations, such as CTRL+F1 or CTRL+A. They appear on the menu to the right of the

corresponding menu item, as shown in Figure 6.10.

Figure 6.10 Shortcut keys



To assign a shortcut key to a menu item

1. Open the **Menu Editor**.
2. Select the menu item.
3. Select a function key or key combination in the **Shortcut** combo box.

   To remove a shortcut key assignment, choose "(none)" from the top of the list.

Note Shortcut keys appear automatically on the menu; therefore, you do not have to enter CTRL+*key* in the Caption box of the Menu Editor.

Send feedback to MSDN. Look here for MSDN Online resources.

Menu Title and Naming Guidelines

To maintain consistency with other applications, it's a good idea to follow established naming guidelines when creating menus.

Setting the Caption Property

When assigning captions for menu items, you should try to follow these guidelines:

- Item names should be unique within a menu, but may be repeated in different menus to represent similar actions.
- Item names may be single, compound, or multiple words.
- Each item name should have a unique mnemonic access character for users who choose commands with keyboards. The access character should be the first letter of the menu title, unless another letter offers a stronger mnemonic link; no two menu titles should use the same access character. For more information about assigning access and shortcut keys, see "Creating Menus with the Menu Editor" earlier in this chapter.

- An ellipsis (…) should follow names of commands that require more information before they can be completed, such as commands that display a dialog (Save As…, Preferences…).

- Keep the item names short. If you are localizing your application, the length of words tends to increase approximately thirty percent in foreign versions, and you may not have enough space to adequately list all of your menu items. For more details on localizing your application, see "International Issues."

Menu Naming Conventions

To make your code more readable and easier to maintain, it's a good idea to follow established naming conventions when setting the Name property in the Menu Editor. Most naming convention guidelines suggest a prefix to identify the object (that is, mnu for a menu control) followed by the name of the top-level menu (for example, File). For submenus, this would be followed by the caption of the submenu (for example, mnuFileOpen).

For More Information For an example of suggested naming conventions, see "Visual Basic Coding Conventions."

Send feedback to MSDN. Look here for MSDN Online resources.

Creating Submenus

Each menu you create can include up to five levels of submenus. A *submenu* branches off another menu to display its own menu items. You may want to use a submenu when:

- The menu bar is full.

- A particular menu control is seldom used.

- You want to emphasize one menu control's relationship to another.

If there is room on the menu bar, however, it's better to create an additional menu title instead of a submenu. That way, all the controls are visible to the user when the menu is dropped down. It's also good programming practice to restrict the use of submenus so users don't get lost trying to navigate your application's menu interface. (Most applications use only one level of submenus.)

In the Menu Editor, any menu control indented below a menu control that is *not* a menu title is a *submenu control*. In general, submenu controls can include submenu items, separator bars, and submenu titles.

To create a submenu

1. Create the menu item that you want to be the submenu title.

2. Create the items that will appear on the new submenu, and indent them by clicking the right arrow button.

   Each indent level is preceded by four dots (....) in the Menu Editor. To remove one level of indentation, click the left arrow button.

Note If you're considering using more than a single level of submenus, think about using a dialog box instead. Dialog boxes allow users to specify several choices in one place. For information on using dialog boxes, see "Dialog Boxes" later in this chapter.

Send feedback to MSDN. Look here for MSDN Online resources.

Creating a Menu Control Array

A *menu control array* is a set of menu items on the same menu that share the same name and event procedures. Use a menu control array to:

- Create a new menu item at run time when it must be a member of a control array. The MDI Notepad sample, for example, uses a menu control array to store a list of recently opened files.
- Simplify code, because common blocks of code can be used for all menu items.

Each menu control array element is identified by a unique index value, indicated in the Index property box on the Menu Editor. When a member of a control array recognizes an event, Visual Basic passes its Index property value to the event procedure as an additional argument. Your event procedure must include code to check the value of the Index property, so you can determine which control you're using.

For More Information For more information on control arrays, see "Working with Control Arrays" in "Using Visual Basic's Standard Controls."

To create a menu control array in the Menu Editor

1. Select the form.
2. From the **Tools** menu, choose **Menu Editor**.

   –or–

   Click the **Menu Editor** button on the toolbar.
3. In the **Caption** text box, type the text for the first menu title that you want to appear on the menu bar.

   The menu title text is displayed in the menu control list box.
4. In the **Name** text box, type the name that you will use to refer to the menu control in code. Leave the **Index** box empty.
5. At the next indentation level, create the menu item that will become the first element in

the array by setting its **Caption** and **Name**.

6. Set the **Index** for the first element in the array to 0.

7. Create a second menu item at the same level of indentation as the first.

8. Set the **Name** of the second element to the same as the first element and set its **Index** to 1.

9. Repeat steps 5 – 8 for subsequent elements of the array.

Important Elements of a menu control array must be contiguous in the menu control list box and must be at the same level of indentation. When you're creating menu control arrays, be sure to include any separator bars that appear on the menu.

Send feedback to MSDN. Look here for MSDN Online resources.

Creating and Modifying Menus at Run Time

The menus you create at design time can also respond dynamically to run-time conditions. For example, if a menu item action becomes inappropriate at some point, you can prevent users from selecting that menu item by *disabling* it. In the MDI NotePad application, for example, if the clipboard doesn't contain any text, the Paste menu item is dimmed on the Edit menu, and users cannot select it.

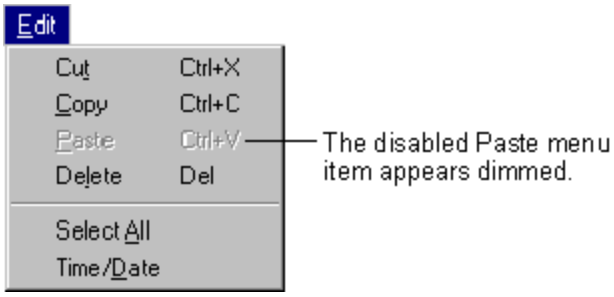
You can also dynamically add menu items, if you have a menu control array. This is described in "Adding Menu Controls at Run Time," later in this topic.

You can also program your application to use a check mark to indicate which of several commands was last selected. For example, the Options, Toolbar menu item from the MDI NotePad application displays a check mark if the toolbar is displayed. Other menu control features described in this section include code that makes a menu item visible or invisible and that adds or deletes menu items.

Enabling and Disabling Menu Commands

All menu controls have an Enabled property, and when this property is set to False, the menu is disabled and does not respond to user actions. Shortcut key access is also disabled when Enabled is set to False. A disabled menu control appears dimmed, like the Paste menu item in Figure 6.11.

Figure 6.11 A disabled menu item



For example, this statement disables the Paste menu item on the Edit menu of the MDI NotePad application:

```
mnuEditPaste.Enabled = False
```

Disabling a menu title in effect disables the entire menu, because the user cannot access any menu item without first clicking the menu title. For example, the following code would disable the Edit menu of the MDI Notepad application:

```
mnuEdit.Enabled = False
```

Displaying a Check Mark on a Menu Control

Using the Checked property, you can place a check mark on a menu to:

- Tell the user the status of an on/off condition. Choosing the menu command alternately adds and removes the check mark.
- Indicate which of several modes is in effect. The Options menu of the MDI Notepad application uses a check mark to indicate the state of the toolbar, as shown in Figure 6.12.

Figure 6.12 A checked menu item

You create check marks in Visual Basic with the Checked property. Set the initial value of the Checked property in the Menu Editor by selecting the check box labeled Checked. To add or remove a check mark from a menu control at run time, set its Checked property from code. For example:

```
Private Sub mnuOptions_Click ()
   ' Set the state of the check mark based on
   ' the Visible property.
   mnuOptionsToolbar.Checked = picToolbar.Visible
End Sub
```

Making Menu Controls Invisible

In the Menu Editor, you set the initial value of the Visible property for a menu control by selecting the check box labeled Visible. To make a menu control visible or invisible at run time,

set its Visible property from code. For example:

```
mnuFileArray(0).Visible = True    ' Make the control
                                  ' visible.

mnuFileArray(0).Visible = False    ' Make the control
                                  ' invisible.
```

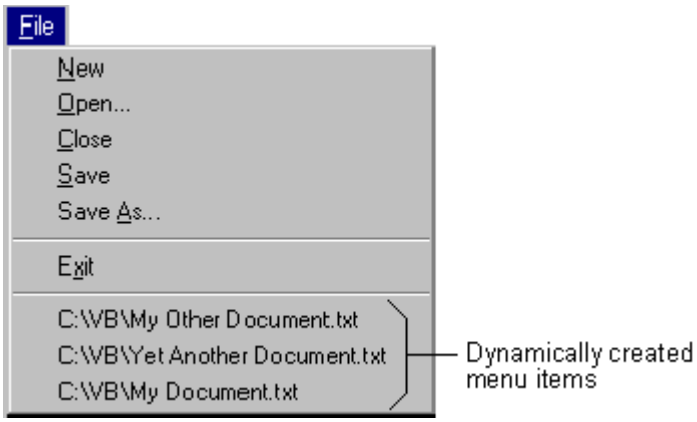
When a menu control is invisible, the rest of the controls in the menu move up to fill the empty space. If the control is on the menu bar, the rest of the controls on the menu bar move left to fill the space.

> **Note** Making a menu control invisible effectively disables it, because the control is inaccessible from the menu, access or shortcut keys. If the menu title is invisible, all the controls on that menu are unavailable.

Adding Menu Controls at Run Time

A menu can grow at run time. In Figure 6.13, for example, as files are opened in the SDI NotePad application, menu items are dynamically created to display the path names of the most recently opened files.

Figure 6.13 Menu control array elements created and displayed at run time



You must use a control array to create a control at run time. Because the mnuRecentFile menu control is assigned a value for the Index property at design time, it automatically becomes an element of a control array — even though no other elements have yet been created.

When you create mnuRecentFile(0), you actually create a separator bar that is invisible at run time. The first time a user saves a file at run time, the separator bar becomes visible, and the first file name is added to the menu. Each time you save a file at run time, additional menu controls are loaded into the array, making the menu grow.

Controls created at run time can be hidden by using the Hide method or by setting the control's Visible property to False. If you want to remove a control in a control array from memory, use the Unload statement.

Send feedback to MSDN. Look here for MSDN Online resources.

Writing Code for Menu Controls

When the user chooses a menu control, a Click event occurs. You need to write a Click event procedure in code for each menu control. All menu controls except separator bars (and disabled or invisible menu controls) recognize the Click event.

The code that you write in a menu event procedure is no different than that which you would write in any other control's event procedure. For example, the code in a File, Close menu's Click event might look like this:

```
Sub mnuFileClose_Click()
   Unload Me
End Sub
```

Visual Basic displays a menu automatically when the menu title is chosen; therefore, it is not necessary to write code for a menu title's Click event procedure unless you want to perform another action, such as disabling certain menu items each time the menu is displayed.

> **Note** At design time, the menus you create are displayed on the form when you close the Menu Editor. Choosing a menu item on the form displays the Click event procedure for that menu control.

Send feedback to MSDN. Look here for MSDN Online resources.

Displaying Pop-up Menus

A *pop-up menu* is a floating menu that is displayed over a form, independent of the menu bar. The items displayed on the pop-up menu depend on where the pointer was located when the right mouse button was pressed; therefore, pop-up menus are also called *context menus*. In Microsoft Windows 95 or later systems, you activate context menus by clicking the right mouse button.

Any menu that has at least one menu item can be displayed at run time as a pop-up menu. To display a pop-up menu, use the PopupMenu method. This method uses the following syntax:

*[object.]***PopupMenu** *menuname [, flags [,x [, y [, boldcommand]]]]*

For example, the following code displays a menu named mnuFile when the user clicks a form with the right mouse button. You can use the MouseUp or MouseDown event to detect when the user clicks the right mouse button, although the standard is to use the MouseUp event:

```
Private Sub Form_MouseUp (Button As Integer, Shift As _
   Integer, X As Single, Y As Single)
   If Button = 2 Then    ' Check if right mouse button
                         ' was clicked.
```

```
      PopupMenu mnuFile    ' Display the File menu as a
                         ' pop-up menu.
   End If
End Sub
```

Any code following a call to the PopupMenu method is not run until the user selects an item in the menu or cancels the menu.

> **Note** Only one pop-up menu can be displayed at a time. While a pop-up menu is displayed, calls to the PopupMenu method are ignored. Calls to the PopupMenu method are also ignored whenever a menu control is active.

Often you want a pop-up menu to access options that are not usually available on the menu bar. To create a menu that will not display on the menu bar, make the top-level menu item invisible at design time (make sure the Visible check box in the Menu Editor is not checked). When Visual Basic displays a pop-up menu, the Visible property of the specified top-level menu is ignored.

The Flags Argument

You use the *flags* argument in the PopupMenu method to further define the location and behavior of a pop-up menu. The following table lists the flags available to describe a pop-up menu's location.

Location constants	Description
vbPopupMenuLeftAlign	Default. The specified *x* location defines the left edge of the pop-up menu.
vbPopupMenuCenterAlign	The pop-up menu is centered around the specified *x* location.
vbPopupMenuRightAlign	The specified *x* location defines the right edge of the pop-up menu.

The following table lists the flags available to describe a pop-up menu's behavior.

Behavior constants	Description
vbPopupMenuLeftButton	Default. The pop-up menu is displayed when the user clicks a menu item with the left mouse button only.
vbPopupMenuRightButton	The pop-up menu is displayed when the user clicks a menu item with either the right or left mouse button.

To specify a flag, you combine one constant from each group using the Or operator. The following code displays a pop-up menu with its top border centered on a form when the user clicks a command button. The pop-up menu triggers Click events for menu items that are clicked with either the right or left mouse button.

```
Private Sub Command1_Click ()
   ' Dimension X and Y variables.
   Dim xloc, yloc
```

```
   ' Set X and Y variables to center of form.
   xloc = ScaleWidth / 2
   yloc = ScaleHeight / 2

   ' Display the pop-up menu.
   PopupMenu mnuEdit, vbPopupMenuCenterAlign Or _
   vbPopupMenuRightButton, xloc, yloc
End Sub
```

The Boldcommand Argument

You use the *boldcommand* argument to specify the name of a menu control in the displayed pop-up menu that you want to appear in bold. Only one menu control in the pop-up menu can be bold.

Send feedback to MSDN. Look here for MSDN Online resources.

Menus in MDI Applications

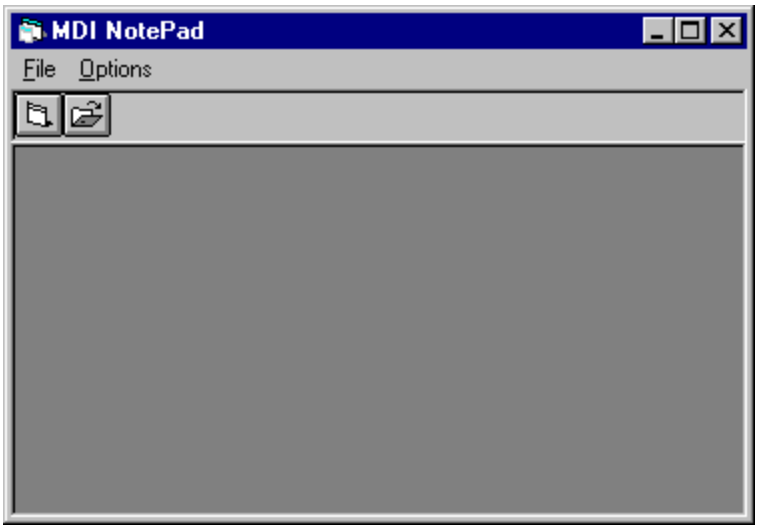
In an MDI application, the menus for each child are displayed on the MDI form, rather than on the child forms themselves. When a child form has the focus, that child's menu (if any) replaces the MDI form's menu on the menu bar. If there are no child forms visible, or if the child with the focus does not have a menu, the MDI form's menu is displayed (see Figures 6.14 and 6.15).

It is common for MDI applications to use several sets of menus. When the user opens a document, the application displays the menu associated with that type of document. Usually, a different menu is displayed when no child forms are visible. For example, when there are no files open, Microsoft Excel displays only the File and Help menus. When the user opens a file, other menus are displayed (File, Edit, View, Insert, Format, Tools, Data, Window, and so on).

Creating Menus for MDI Applications

You can create menus for your Visual Basic application by adding menu controls to the MDI form and to the child forms. One way to manage the menus in your MDI application is to place the menu controls you want displayed all of the time, even when no child forms are visible, on the MDI form. When you run the application, the MDI form's menu is automatically displayed when there are no child forms visible, as shown in Figure 6.14.

Figure 6.14 The MDI form menu is displayed when no child forms are loaded



Place the menu controls that apply to a child form on the child form. At run time, as long as there is at least one child form visible, these menu titles are displayed in the menu bar of the MDI form.

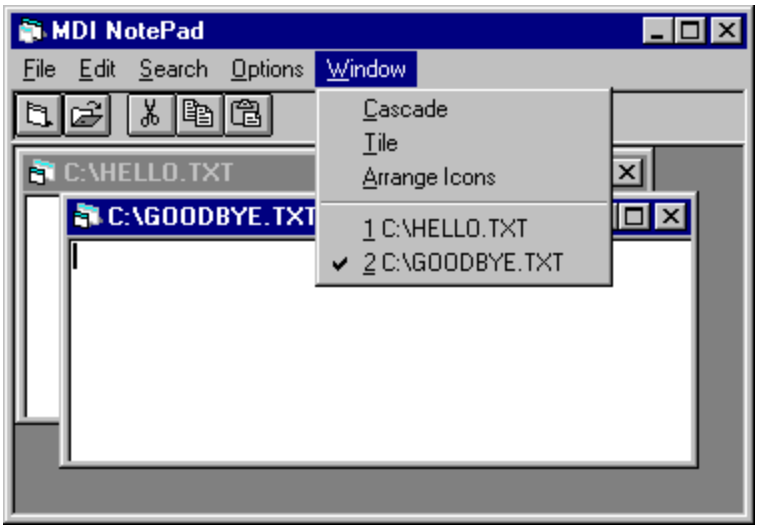
Some applications support more than one type of document. For example, in Microsoft Access, you can open tables, queries, forms, and other document types. To create an application such as this in Visual Basic, use two child forms. Design one child with menus that perform spreadsheet tasks and the other with menus that perform charting tasks.

At run time, when an instance of a spreadsheet form has the focus, the spreadsheet menu is displayed, and when the user selects a chart, that form's menu is displayed. If all the spreadsheets and charts are closed, the MDI form's menu is displayed. For more information on creating menus, see "Using Menus in Your Application" earlier in this chapter.

Creating a Window Menu

Most MDI applications (for example, Microsoft Word for Windows and Microsoft Excel) incorporate a Window menu. This is a special menu that displays the captions of all open child forms, as shown in Figure 6.15. In addition, some applications place commands on this menu that manipulate the child windows, such as Cascade, Tile, and Arrange Icons.

Figure 6.15 The Window menu displays the name of each open child form



Any menu control on an MDI form or MDI child form can be used to display the list of open child forms by setting the WindowList property for that menu control to True. At run time, Visual Basic automatically manages and displays the list of captions and displays a check mark next to the one that had the focus most recently. In addition, a separator bar is automatically placed above the list of windows.

To set the WindowList property

1. Select the form where you want the menu to appear, and from the **Tools** menu, choose **Menu Editor**.

   Note The WindowList property applies only to MDI forms and MDI child forms. It has no effect on standard (non-MDI) forms.

2. In the **Menu Editor** list box, select the menu where you want the list of open child forms to display.

3. Select the **WindowList** check box.

At run time, this menu displays the list of open child forms. In addition, the WindowList property for this menu control returns as True.

For More Information See "WindowList Property" in the *Language Reference*.

Arranging Child Forms

As was mentioned earlier, some applications list actions such as Tile, Cascade, and Arrange Icons on a menu, along with the list of open child forms. Use the Arrange method to rearrange child forms in the MDI form. You can display child forms as cascading, as horizontally tiled, or as child form icons arranged along the lower portion of the MDI form. The following example shows the Click event procedures for the Cascade, Tile, and Arrange Icons menu controls.

```
Private Sub mnuWCascade_Click ()
   ' Cascade child forms.
   frmMDI.Arrange vbCascade
End Sub

Private Sub mnuWTile_Click ()
```

```
   ' Tile child forms (horizontal).
   frmMDI.Arrange vbTileHorizontal
End Sub

Private Sub mnuWArrange_Click ()
   ' Arrange all child form icons.
   frmMDI.Arrange vbArrangeIcons
End Sub
```

Note The intrinsic constants vbCascade, vbTileHorizontal, and vbArrangeIcons are listed in the Visual Basic (VB) object library of the Object Browser.

When you tile or cascade child forms that have a fixed border style, each child form is positioned as if it had a sizable border. This can cause child forms to overlap.

Send feedback to MSDN. Look here for MSDN Online resources.