Applications with Multiple Forms

After studying Appendix D, you should be able to:

- Explain the difference between SDI, MDI, and TDI applications
- Create an SDI application that contains more than one form
- Include Font and Color dialog boxes in an application
- Print the user interface during run time
- Create an MDI application
- Code the FormClosing event
- Create a quasi TDI application using the TabControl tool

**SDI, MDI, and TDI**

Applications can be classified as single-document interface (SDI), multiple-document interface (MDI), or tabbed-document interface (TDI). In an SDI application, each window of the application holds a single document. If you need to open another document, you must open a new window. Examples of SDI applications include Windows Notepad and Microsoft Word (version 2000 and later). As shown in Figures D-1 and D-2, each document in an SDI application appears in its own window. Notice that each window has its own menu bar and toolbar.
Unlike an SDI application, an MDI application consists of a parent window and one or more child windows that are contained within the parent window. The predecessors of Microsoft Word 2000 are examples of MDI applications. Although current versions of Microsoft Word use SDI, you can switch to MDI. If you are using Office 2007, you do this by clicking the Microsoft Office button, clicking Word Options, clicking Advanced, deselecting (in the Display section) the Show all windows in the Taskbar check box, and then clicking the OK button. (If you are using Office 2010, you would need to click Options on the File tab.) Figure D-3 shows the result of changing Microsoft Office 2007 from SDI to MDI. Notice that each document appears in a separate child window within the Microsoft Word parent window. Also notice that a single menu bar and toolbar are shared between both child windows.
SDI, MDI, and TDI

Each document resides in its own child window within the parent window.

Both child windows share the parent’s menu bar and toolbars.

Figure D-3  Microsoft Word 2007 (as an MDI application)

MDI applications are not as popular as they once were, because many users find the management of child windows to be either difficult or confusing. One of the biggest problems cited is the inability to quickly determine which documents are open. This is especially true when one of the child windows is maximized, because the other child windows are hidden behind the maximized one. Most MDI applications provide a menu (usually named Window or View) that lists the open documents.

TDI is an offshoot of MDI. Like an MDI application, a TDI application allows more than one child window to be open in a parent window. Unlike an MDI application, however, a TDI application uses tabs to show the windows that are open in an application. Visual Studio 2010 is an example of a TDI application. See Figure D-4.

Figure D-4  Visual Studio (TDI application)
All of the applications you created in the chapters in this book were SDI applications that contained one form. In this appendix, you will learn how to create SDI applications that contain more than one form. The Country Charm Inn application, for example, contains a main form and a splash screen form. The Font and Color application contains a main form and two forms that represent dialog boxes. The Zappet application contains three main forms. You also will learn how to create an MDI application named Jotpad, which contains a parent form and multiple child forms. The creation of a TDI application is beyond the scope of this book; however, you will create a quasi TDI application (named Currency Converter) using the TabControl tool in the toolbox. In addition to learning how to create SDI, MDI, and quasi TDI applications, you also will learn how to print an interface during run time, as well as code a form’s FormClosing event procedure.

**Country Charm Inn Application**

The simplest multiple-form application is one composed of a main form and a splash screen. As you learned in Chapter 1, a splash screen is the first image that appears when an application is started. It is used to introduce the application and to hold the user’s attention while the application is being read into the computer’s internal memory.

To add a splash screen to the Country Charm Inn application:

1. If necessary, start Visual Studio 2010 or Visual Basic 2010 Express. Open the Country Solution (Country Solution.sln) file contained in the VB2010\AppD\Country Solution folder. If necessary, open the designer window. The application’s interface appears on the screen. See Figure D-5.

   ![Figure D-5](image)

   **Figure D-5**  User interface for the Country Charm Inn application

2. Now, you will add a splash screen to the application. Click Project on the menu bar and then click Add Windows Form to open the Add New Item dialog box. The dialog box provides a template that is already configured as a splash screen. Scroll the middle column of the
dialog box until you see Splash Screen, and then click **Splash Screen**. Change the splash screen’s name to **Splash Form.vb** and then click the **Add** button. Figure D-6 shows the form created by the Splash Screen template during design time. The form contains five controls: two table layout panels and three labels.

![Figure D-6 Splash screen added to the project](image)

3. You can modify the form by adding controls to it or deleting controls from it. You also can use the Properties window to change the properties of the form and its controls. In the Properties window, change the splash screen’s name to **frmSplash**.

In Chapter 1 you learned that the computer automatically displays an application’s startup form each time the application is started. It also automatically displays an application’s splash screen; however, you first must specify the splash screen’s name in the Project Designer window. When the application is started, the splash screen will appear first. After a few seconds, the splash screen will disappear automatically and the startup form will appear.

**To specify the name of the splash screen and then start the application:**

1. Right-click **My Project** in the Solution Explorer window, and then click **Open** to open the Project Designer window. Click the **Splash screen** list arrow in the Application pane, and then click **frmSplash** in the list.

2. Click the **Assembly Information** button in the Application pane to open the Assembly Information dialog box. Modify the Title, Company, and Copyright boxes as shown in Figure D-7.
3. Click the **OK** button to close the Assembly Information dialog box. Save the solution and then close the Project Designer window.

4. Start the application. The splash screen shown in Figure D-8 appears first.

5. After a few seconds have elapsed, the splash screen disappears and the startup form (frmMain) appears. Click the **Exit** button and then close the solution.

**Font and Color Application**

Most Windows applications consist of at least one main window (often referred to as a primary window) and one or more secondary windows, called dialog boxes. The primary viewing and editing of your application’s
data take place in a primary window. The primary window shown in Figure D-9, for example, allows you to view and edit documents created using the Notepad application. Dialog boxes, on the other hand, support and supplement a user’s activities in a primary window. For instance, you can use the Font dialog box in Figure D-9 to specify the font of the text selected in the primary window.

![Figure D-9 Primary window and Font dialog box in Notepad](image)

Visual Basic provides several tools for creating commonly-used dialog boxes, such as the Color, Font, and Save As dialog boxes. The tools are located in the Dialogs section of the toolbox. When you drag one of these tools to a form, its instantiated control does not appear on the form. Instead, the control is placed in the component tray in the IDE. You use the ShowDialog method to show (in other words, open) the dialog box during run time. The method’s syntax is `dialogControlName.ShowDialog()`.

The dialog boxes created by the dialog tools in the toolbox are modal, which means they remain on the screen until the user closes them. While a modal dialog box is on the screen, no input from the keyboard or mouse can occur in the application’s primary window; however, you can access other applications. You close a modal dialog box by selecting either its OK button or its Cancel button, or by clicking the Close button on its title bar. You will use both the **Font dialog box** and the **Color dialog box** in the Font and Color application.
To add Font and Color dialog boxes to the Font and Color application:

1. Open the Font Color Solution (Font Color Solution.sln) file contained in the VB2010\AppD\Font Color Solution folder. If necessary, open the designer window. The application's interface appears on the screen. See Figure D-10.

   ![Figure D-10](image)
   **Figure D-10** User interface for the Font and Color application

2. Click `FontDialog` in the Dialogs section of the toolbox, and then drag your mouse pointer to the form. When you release the mouse button, a font dialog control appears in the component tray. Change the control's name to `dlgFont`.

3. Click `ColorDialog` in the Dialogs section of the toolbox, and then drag your mouse pointer to the form. When you release the mouse button, a color dialog control appears in the component tray. Change the control's name to `dlgColor`. See Figure D-11.

   ![Figure D-11](image)
   **Figure D-11** Dialog box controls shown in the component tray

The Font button in the interface should allow the user to change the Font property values (name, style, size, and so on) for the `lblMessage` control. Similarly, the Color button should allow the user to change the value in the `lblMessage` control's ForeColor property.
To code the Font and Color buttons:

1. Open the Code Editor window. In the comments, replace <your name> and <current date> with your name and the current date, respectively.

2. Open the code template for the btnFont control’s Click event procedure. First, you will assign the lblMessage control’s Font property value to the dlgFont control’s Font property. By doing this, the label’s values will be selected in the Font dialog box when the dialog box appears on the screen. Enter the following statement:

   ```vba
dlgFont.Font = lblMessage.Font
```

3. Now, you will use the ShowDialog method to tell the computer to open the dialog box. Enter the following statement:

   ```vba
dlgFont.ShowDialog()
```

4. Finally, you will assign the values selected in the Font dialog box to the label’s Font property. Enter the following statement:

   ```vba
lblMessage.Font = dlgFont.Font
```

5. Next, open the code template for the btnColor control’s Click event procedure. The procedure will begin by assigning the current value in the lblMessage control’s ForeColor property to the dlgColor control’s Color property. Enter the following statement:

   ```vba
dlgColor.Color = lblMessage.ForeColor
```

6. Here too, you will use the ShowDialog method to tell the computer to open the dialog box. Enter the following statement:

   ```vba
dlgColor.ShowDialog()
```

7. Finally, you will assign the color selected in the Color dialog box to the label’s ForeColor property. Enter the following statement:

   ```vba
lblMessage.ForeColor = dlgColor.Color
```

8. Save the solution. Figure D-12 shows the code entered in the Click event procedures for the Font and Color buttons.

![Figure D-12](image)

Figure D-12  Click event procedures for the Font and Color buttons
9. Save the solution and then start the application. Click the **Color** button to open the Color dialog box. See Figure D-13.

![Figure D-13  Color dialog box](image)

10. Click any **red square** and then click the **OK** button. The color of the text in the label control changes to red.

11. Click the **Font** button to open the Font dialog box. See Figure D-14.

![Figure D-14  Font dialog box](image)

12. Click **Bold Italic** in the list of font styles and then click the **OK** button. The label control’s text is now boldfaced and italicized. See Figure D-15.
13. Click the **Exit** button.

In the next section, you will learn how to print the Font and Color application's interface during run time.

**Printing an Interface During Run Time**

Visual Basic provides the **PrintForm tool** for printing an interface during run time. The tool is contained in the Visual Basic PowerPacks section of the toolbox. When you drag the PrintForm tool to a form, the instantiated print form control appears in the component tray. You can use the control to send the printout either to the Print preview window or directly to the printer. The syntax for printing the interface during run time is shown in Figure D-16 along with examples of using the syntax. As the figure indicates, the printing task requires two statements. The first statement specifies the output destination, and the second statement tells the computer to start the print operation.

```
Printing the interface during run time

**Syntax**

object.PrintAction = Printing.PrintAction.destination
object.Print()

<table>
<thead>
<tr>
<th>destination</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>PrintToPreview</td>
<td>sends the printout to the Print preview window</td>
</tr>
<tr>
<td>PrintToPrinter</td>
<td>sends the printout to the printer</td>
</tr>
</tbody>
</table>

**Example 1**

```vbnet
PrintForm1.PrintAction = Printing.PrintAction.PrintToPreview
PrintForm1.Print()
```

sends the output to the Print preview window

**Example 2**

```vbnet
PrintForm1.PrintAction = Printing.PrintAction.PrintToPrinter
PrintForm1.Print()
```

sends the output directly to the printer

---

**Figure D-15**  Status of the lblMessage control's text

**Figure D-16**  Syntax and examples of printing the interface during run time
To add a print form control to the application:

1. Return to the designer window. Click PrintForm in the Visual Basic PowerPacks section of the toolbox, and then drag your mouse pointer to the form. When you release the mouse button, a print form control appears in the component tray.

2. Return to the Code Editor window. Open the code template for the btnPrint control's Click event procedure. You will send the printout to the Print preview window. Enter the statements shown in Figure D-17.

```vbnet
Private Sub btnPrint_Click(ByVal sender As Object, ByVal e As System.Drawing.Printing.PrintEventArgs)
    PrintForm.PrintAction = Printing.PrintAction.PrintToPreview
    PrintForm.Print()
End Sub
```

Figure D-17  Click event procedure for the Print button

3. Save the solution and then start the application. Click the Print button. A printout of the interface appears in the Print preview window. (It may take a few seconds for the window to open,) Click the Zoom button list arrow and then click 75%. See Figure D-18.

4. If your computer is connected to a printer, click the Print button (the printer) on the Print preview window's toolbar to send the output to the printer.

5. Click the Close button on the Print preview window's toolbar. Click the Exit button and then close the solution.
Custom Dialog Boxes

You can also create your own dialog boxes in Visual Basic. You do this using either the Dialog template or the Windows Form template; both templates are available in the Add New Item dialog box. The Dialog template creates a Windows form that is already configured for use as a dialog box. The Windows Form template, on the other hand, adds a blank Windows form to the project. To make the form a dialog box, you would need to set its FormBorderStyle property to FixedDialog. The FixedDialog setting draws a fixed border around the form, and it removes the Control menu box from the form’s title bar. The Windows standard is that dialog boxes contain only a Close button and, in some cases, a Help button. Therefore, you also would need to remove the Minimize and Maximize buttons from the title bar by setting the form’s MinimizeBox and MaximizeBox properties to False.

Zappet Application

Some Windows applications contain more than one primary form. The Zappet application, for example, has three primary forms. The first form gets the customer’s billing information, the second gets the customer’s shipping information, and the third displays the billing and shipping information for the customer to review.

To view the forms contained in the Zappet application:

1. Open the Zappet Solution (Zappet Solution.sln) file contained in the VB2010\AppD\Zappet Solution folder. If the Zappet - Billing form does not appear in the designer window, double-click Billing Form.vb in the Solution Explorer window. The form, which is named frmBilling, provides text boxes for the user to enter a customer’s billing information. See Figure D-19.

![Figure D-19](Zappet - Billing form)

2. Double-click Shipping Form.vb in the Solution Explorer window. The Zappet - Shipping form, which is named frmShipping, appears in the designer window. The form provides text boxes for the user to enter a customer’s shipping information. See Figure D-20.
3. Close the Zappet - Shipping form and then double-click Review Form.vb in the Solution Explorer window. The Zappet - Billing/Shipping Review form, which is named frmReview, appears in the designer window. The form uses labels to display the billing and shipping information entered in the frmBilling and frmShipping forms, respectively. See Figure D-21.


In order to display the billing information on the frmReview form, you will need to save the information in variables. You will declare the variables in the frmBilling form’s Declarations section, using the keyword Friend. Variables declared using the Friend keyword are recognized by any form within the current solution.

To begin coding the frmBilling form:

1. Open the frmBilling form’s Code Editor window. Notice that the btnExit control’s Click event procedure contains the Me.Close() statement.

2. In the comments, replace <your name> and <current date> with your name and the current date, respectively.
3. Click the **blank line** below the ' declare Friend variables' comment. Enter the following five declaration statements:

```vbnet
Friend strBillName As String
Friend strBillAddress As String
Friend strBillCity As String
Friend strBillState As String
Friend strBillZip As String
```

4. When the user clicks the Continue button on the frmBilling form, the button's Click event procedure will save the billing information in the Friend variables. Open the code template for the btnContinue control's Click event procedure. Type the following comment and then press **Enter** twice:

' continue to the shipping form

5. Now enter the following comment and assignment statements. Press **Enter** twice after typing the last assignment statement.

```vbnet
' save billing information
strBillName = txtName.Text
strBillAddress = txtAddress.Text
strBillCity = txtCity.Text
strBillState = txtState.Text
strBillZip = txtZip.Text
```

After saving the billing information, the Continue button's Click event procedure will hide the current form and then show the frmShipping form. You hide the current form using the form's **Hide method**. The method's syntax is `Me.Hide()`, where `Me` refers to the current form. To show a form—in other words, to open the form—you use the form's **Show method**; the method's syntax is `formName.Show()`.

**To finish coding the frmBilling form:**

1. Enter the following comments and statements:

```vbnet
' hide the billing form and then
' show the shipping form
Me.Hide()
frmShipping.Show()
```

2. Save the solution. Close the Code Editor window and then close the frmBilling designer window.

You will code the shipping form next.

**Coding the Shipping Form**

For many customer orders, the shipping information is the same as the billing information. Therefore, when the shipping form is loaded into memory, its Load event procedure will display the billing information in the text boxes on the form. In cases where the shipping information is different from the billing information, the user can make the appropriate changes in the text boxes.
To begin coding the frmShipping form:

1. Double-click **Shipping Form.vb** in the Solution Explorer window to open the frmShipping form in the designer window, and then open the form's Code Editor window. Notice that the btnExit control's Click event procedure contains the `Me.Close()` statement.

2. In the comments, replace `<your name>` and `<current date>` with your name and the current date, respectively.

3. As mentioned earlier, the shipping form's Load event procedure will display the billing information in the text boxes on the form. Open the code template for the form's Load event procedure. Type the following comment and then press **Enter** twice:

   ```
   ' display billing information
   ```

Recall that the billing information is stored in the Friend variables declared in the frmBilling form. To refer to a Friend variable declared in another form, you use the syntax `formName.variableName`, where `formName` is the name of the form in which the Friend variable is declared.

To continue coding the frmShipping form:

1. Enter the following five assignment statements:

   ```
   txtName.Text = frmBilling.strBillName
   txtAddress.Text = frmBilling.strBillAddress
   txtCity.Text = frmBilling.strBillCity
   txtState.Text = frmBilling.strBillState
   txtZip.Text = frmBilling.strBillZip
   ```

2. In order to display the shipping information on the frmReview form, you will need to save the information in variables. Here too, you will declare the variables in the form's Declarations section, using the keyword **Friend**. Click the blank line below the `Friend variables` comment. Enter the following five declaration statements:

   ```
   Friend strShipName As String
   Friend strShipAddress As String
   Friend strShipCity As String
   Friend strShipState As String
   Friend strShipZip As String
   ```

3. When the user clicks the Continue button on the frmShipping form, the button's Click event procedure will save the shipping information in the form's Friend variables. Open the code template for the btnContinue control's Click event procedure. Type the following comment and then press **Enter** twice:

   ```
   ' continue to the review form
   ```

4. Now enter the following comment and assignment statements. Press **Enter** twice after typing the last assignment statement.

   ```
   ' save shipping information
   strShipName = txtName.Text
   strShipAddress = txtAddress.Text
   ```
strShipCity = txtCity.Text
strShipState = txtState.Text
strShipZip = txtZip.Text

5. Finally, the procedure will hide the current form and then show the frmReview form. Enter the following comments and statements:

' hide the shipping form and then
' show the review form
Me.Hide()
frmReview.Show()

6. Save the solution and then start the application. The billing form appears first, because it is the startup form.

7. Enter any name, address, city, state, and ZIP code in the text boxes, and then click the **Continue** button. The shipping form opens and displays the billing information. The text boxes on the shipping form allow the user to change the shipping information, if necessary.

8. Click the **Exit** button on the shipping form. Look closely at the Microsoft Visual Studio (or Microsoft Visual Basic Express) title bar. See Figure D-22. The “(Running)” text in the title bar indicates that the application is still running. This is because the `Me.Close()` statement in the button’s Click event procedure closes only the current form, which is the frmShipping form. It does not close the frmBilling form, which is hidden but still resides in the computer’s internal memory. You will fix this problem in the next section.

![Figure D-22 Result of clicking the Exit button](image)

9. Click **Debug** on the menu bar and then click **Stop Debugging** to end the application.

**FormClosing Event**

To fix the problem encountered in Step 8 in the previous set of steps, you will need to close the billing form using the statement `frmBilling.Close()`. You will place the statement in the shipping form’s FormClosing event procedure. A form’s **FormClosing event** occurs immediately before the form is closed.
To finish coding the frmShipping form:

1. Open the form's FormClosing event procedure and then enter the following statement:
   
   ```vbnet
   frmBilling.Close()
   ```

2. Save the solution and then start the application. The billing form appears first. Enter any name, address, city, state, and ZIP code in the text boxes, and then click the Continue button. The shipping form opens and displays the billing information.

3. Click the Exit button on the shipping form. The “(Running)” text no longer appears in the Microsoft Visual Studio (or Microsoft Visual Basic Express) title bar, indicating that the application has ended.

4. Close the Code Editor window and then close the frmShipping designer window.

Finally, you will code the review form.

Coding the Review Form

When the review form is loaded into memory, its Load event procedure will display the billing and shipping information entered on the previous forms.

To code the frmReview form:

1. Double-click Review Form.vb in the Solution Explorer window to open the frmReview form in the designer window, and then open the form's Code Editor window. Notice that the btnExit control's Click event procedure contains the Me.Close() statement.

2. In the comments, replace <your name> and <current date> with your name and the current date, respectively.

3. Open the code template for the form's Load event procedure. Type the following comment and then press Enter twice:
   
   ```vbnet
   ' display billing and shipping information
   ```

4. Enter the following 10 assignment statements:
   
   ```vbnet
   lblBillName.Text = frmBilling.strBillName
   lblBillAddress.Text = frmBilling.strBillAddress
   lblBillCity.Text = frmBilling.strBillCity
   lblBillState.Text = frmBilling.strBillState
   lblBillZip.Text = frmBilling.strBillZip
   lblShipName.Text = frmShipping.strShipName
   lblShipAddress.Text = frmShipping.strShipAddress
   lblShipCity.Text = frmShipping.strShipCity
   lblShipState.Text = frmShipping.strShipState
   lblShipZip.Text = frmShipping.strShipZip
   ```

5. Before the frmReview form closes, you will need to specifically close the frmBilling and frmShipping forms, which will be hidden but will still
remain in the computer's internal memory during run time. Open the form's FormClosing event procedure and then enter the following statements:

```csharp
frmBilling.Close()
frmShipping.Close()
```

6. Save the solution. Close the Code Editor window and then close the frmReview designer window.

Figure D-23 shows the Zappet application's code.
Public Class frmShipping
    ' declare Friend variables
    Friend strShipName As String
    Friend strShipAddress As String
    Friend strShipCity As String
    Friend strShipState As String
    Friend strShipZip As String

    Private Sub btnExit_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles btnExit.Click
        Me.Close()
    End Sub

    Private Sub frmShipping_FormClosing(ByVal sender As Object, ByVal e As System.Windows.Forms.FormClosingEventArgs) Handles Me.FormClosing
        frmBilling.Close()
    End Sub

    Private Sub frmShipping_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Load
        ' display billing information
        txtName.Text = frmBilling.strBillName
        txtAddress.Text = frmBilling.strBillAddress
        txtCity.Text = frmBilling.strBillCity
        txtState.Text = frmBilling.strBillState
        txtZip.Text = frmBilling.strBillZip
    End Sub

    Private Sub btnContinue_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles btnContinue.Click
        ' continue to the review form
        ' save shipping information
        strShipName = txtName.Text
        strShipAddress = txtAddress.Text
        strShipCity = txtCity.Text
        strShipState = txtState.Text
        strShipZip = txtZip.Text
        ' hide the shipping form and then
        ' show the review form
        Me.Hide()
        frmReview.Show()
    End Sub
End Class
To test the Zappet application:

1. Double-click **Billing Form.vb** in the Solution Explorer window to open the frmBilling form in the designer window.

2. Start the application. The billing form appears first. Enter any name, address, city, state, and ZIP code in the text boxes on the billing form, and then click the **Continue** button. The shipping form opens and displays the billing information.

3. Enter any name, address, city, state, and ZIP code in the text boxes on the shipping form, and then click the **Continue** button. The review form appears and displays the billing and shipping information.

4. Click the **Exit** button on the review form to end the application, and then close the solution.
Jotpad Application

In this section, you will create an MDI application named Jotpad. The application will allow the user to both save information to and read information from a sequential access file.

To open the Jotpad application:

1. Open the Jotpad Solution (Jotpad Solution.sln) file contained in the VB2010\AppD\Jotpad Solution folder. The solution contains two forms named frmChild and frmParent.

2. If necessary, double-click Child Form.vb and Parent Form.vb in the Solution Explorer window to open each form’s designer window.

3. If necessary, click the Child Form.vb [Design] tab to view the frmChild form. The form contains a text box named txtNote. The text box fills the interior of the form, because its Dock property is set to Fill. See Figure D-24.

4. Click the Parent Form.vb [Design] tab to view the frmParent form. The form contains a File menu and a Window menu. Click File on the form’s menu bar to open the File menu. The File menu contains five menu items. See Figure D-25.
5. Click **Window** on the form’s menu bar. Notice that the menu does not contain any menu items. Click the form’s **title bar** to close the Window menu.

Recall that MDI applications contain a parent form and one or more child forms. Currently, neither form in the Jotpad application is the parent or child. You specify that a form is the parent form by setting its **IsMdiContainer property** to True.

**To specify the parent form:**

1. Set the frmParent form’s IsMdiContainer property to **True**. The color of the form’s interior changes to a dark gray, and the interior appears indented within the form’s borders. See Figure D-26.

   ![Figure D-25](image1)  
   **Figure D-25**  
   frmParent form

   ![Figure D-26](image2)  
   **Figure D-26**  
   Result of setting the IsMdiContainer property to True

The Window menu on the parent form should list the open child forms. You accomplish this task by setting the menu strip control’s **MdiWindowListItem property** to the name of the Window menu.

**To have the Window menu list the open child forms:**

1. Click the **MenuStrip1** control in the component tray. Click **MdiWindowListItem** in the Properties window and then set the property to **mnuWindow**.

2. Save the solution.
Now that the interface is complete, you can begin coding the application. When you open the Code Editor window, you will find that most of the code has already been entered for you. To complete the application, you need to code the Click event procedures for the File menu’s New Document and Close Document menu items. You also need to finish coding the Click event procedures for the File menu’s Open Existing Document and Save Document menu items.

To begin coding the New Document item’s Click event procedure:

1. Open the Code Editor window. The window contains the code for the mnuFileExit object’s Click event procedure. It also contains most of the code for the mnuFileOpen and mnuFileSave objects’ Click event procedures.

2. Replace <your name> and <current date> in the comments with your name and the current date, respectively.

3. Open the code template for the mnuFileNew object’s Click event procedure. Type the following comment and then press **Enter** twice:

   ' displays a new child window

4. The Click event procedure should create a new child window and then display the window within the parent window. Enter the following statements. The first statement creates a new frmChild form named frmNewChild. The second statement makes frmNewChild a child of the current form, which is the frmParent form. The third statement displays the child form within the parent form.

   ```vba
   Dim frmNewChild As New frmChild
   frmNewChild.MdiParent = Me
   frmNewChild.Show()
   ```

5. Save the solution and then start the application. Click **File** on the parent window’s menu bar and then click **New Document**. A new child window appears within the parent window.

6. Click **File** on the parent window’s menu bar and then click **New Document**. Another child window appears within the parent window. See Figure D-27.

   ![Figure D-27 Parent window and two child windows](Image)

   **Figure D-27** Parent window and two child windows

7. Click **File** on the parent window’s menu bar and then click **Exit**.
To keep track of the different child windows, you will display the message “DocumentX” (where X is the number of the child window) in each child window’s title bar.

To display a message in each child window’s title bar:

1. Insert a blank line below the Dim statement in the mnuFileNew object’s Click event procedure, and then enter the following statement:
   ```vbs```
   Static intNum As Integer
   ```vbs```
2. Insert a blank line below the frmNewChild.MdiParent = Me statement, and then enter the following assignment statements:
   ```vbs```
   intNum += 1
   frmNewChild.Text = "Document" & intNum.ToString
   ```vbs```
3. Save the solution and then start the application. Click File on the parent window’s menu bar and then click New Document. The Document1 child window appears within the parent window. Click File on the parent window’s menu bar and then click New Document. The Document2 child window appears within the parent window. See Figure D-28.

   ![Figure D-28 Document1 and Document2 windows within the parent window](image)

4. Click File on the parent window’s menu bar and then click Exit.

Next, you will code the File menu’s Close Document option. The option’s Click event procedure should close the current (or active) child window. You can use the parent form’s ActiveMdiChild property to determine the active child window (if any). If no child window is present, the property contains the keyword Nothing.

To code the Close Document object’s Click event procedure:

1. Open the code template for the mnuFileClose object’s Click event procedure. Type the following comment and then press Enter twice:
   ```vbs```
   ' closes the current (active) child window
   ```vbs```
2. Enter the following selection structure:
   ```vbs```
   If ActiveMdiChild IsNot Nothing Then
       ActiveMdiChild.Close()
   End If
   ```vbs```
3. Save the solution and then start the application. Use the New Document option on the File menu to open two child windows.
4. You will close the Document1 window first. Click the Document1 window’s title bar to make it the active window. Click File and then click Close Document. Now, you will close the Document2 window, which is the active window. Click File and then click Close Document.

5. Click File on the parent window’s menu bar and then click Exit.

Now, you will finish coding the File menu’s Save Document option, which allows the user to save information to a sequential access file. The option’s Click event procedure should prompt the user to enter a filename. After using the filename to open a sequential access file for output, the procedure should save the contents of the txtNote control, which is on the frmChild form, in the file. You can refer to a control on a child form using the syntax ActiveMdiChild.Controls("controlName"). For example, to refer to the txtNote control on the active child form, you use ActiveMdiChild.Controls("txtNote"). To refer to the Text property of the txtNote control, you use ActiveMdiChild.Controls("txtNote").Text.

To finish coding the Save Document option’s Click event procedure:

1. Locate the code template for the mnuFileSave object’s Click event procedure. Study the existing code.

2. Click the blank line below the ' write the text to the file comment. Enter the following statement:

   outFile.WriteLine(ActiveMdiChild.Controls("txtNote").Text)

3. Save the solution and then start the application. Press Ctrl+n (the New Document option’s shortcut keys) to open a child window. Type Visual Basic in the text box and then press Enter.

4. Click File on the parent window’s menu bar and then click Save Document. Type First Note.txt as the filename and then press Enter. The “Saved to First Note.txt” message appears in a message box. See Figure D-29.

   Figure D-29  Message indicating that the document was saved
5. Close the message box.

6. Change the text entered in the text box to **Have a great day!**. Use the Save Document option on the File menu to save the text in the Second Note.txt file. Close the message box.

7. Click **File** on the parent window’s menu bar and then click **Exit**.

Finally, you will finish coding the File menu’s Open Existing Document option, which allows the user to open an existing sequential access file. The option should prompt the user to enter a filename. After determining whether the file exists, the procedure should use the filename to open the sequential access file for input. It then should read the contents of the file, displaying the contents in the txtNote control on the active child form. If no child form is present, the procedure should invoke the New Document option’s Click event procedure.

**To finish coding the Open Existing Document option’s Click event procedure:**

1. Locate the code template for the mnuFileOpen object’s Click event procedure. Study the existing code.
2. Click the **blank line** below the `text box on the child form` comment. Enter the following selection structure and assignment statement:

   ```vba
   If ActiveMdiChild Is Nothing Then
     mnuFileNew.PerformClick()
   End If
   ActiveMdiChild.Controls("txtNote").Text = inFile.ReadLine
   ```
3. Save the solution and then start the application. Click **File** on the parent window’s menu bar and then click **Open Existing Document**. Type **Second Note.txt** as the filename and then press **Enter**. The Document1 window opens and displays the contents of the file, which is “Have a great day!”.
4. Press **Ctrl+n** to open another child window. Use the File menu’s Open Existing Document option to open the First Note.txt file. The words “Visual Basic” appear in the Document2 window.
5. Click **Window** on the parent window’s menu bar. The Window menu lists the open windows. See Figure D-30.

![Figure D-30](image)

**Figure D-30** Open Window menu
6. Click **Document1** on the Window menu to make the Document1 window the active window.

7. Click **File** on the parent window's menu bar and then click **Exit**. Close the Code Editor window and then close the solution.

Figure D-31 shows the Jotpad application's code. All of the code is entered in the parent form's Code Editor window.

```vbnet
Public Class frmParent
    Private Sub mnuFileExit_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles mnuFileExit.Click
        Me.Close()
    End Sub

    Private Sub mnuFileOpen_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles mnuFileOpen.Click
        ' reads information from a sequential access file
        Dim inFile As IO.StreamReader
        Dim strFileName As String

        ' get the filename
        strFileName = InputBox("Filename", "Jotpad")

        If IO.File.Exists(strFileName) Then
            ' open the file for input
            inFile = IO.File.OpenText(strFileName)
            ' assign the file's contents to the
            ' text box on the child form
            If ActiveMdiChild Is Nothing Then
                mnuFileNew.PerformClick()
            End If
            ActiveMdiChild.Controls("txtNote").Text = inFile.ReadLine
        Else
            MessageBox.Show("Can't locate " & strFileName, "Jotpad", MessageBoxButtons.OK, MessageBoxIcon.Information)
        End If
    End Sub

    Private Sub mnuFileSave_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles mnuFileSave.Click
        ' close the file
        inFile.Close()
    End Sub

    Private Sub mnuFileNew_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles mnuFileNew.Click
End Sub
```

**Figure D-31**  Jotpad application's code (continues)
(continued)

```vbnet
' saves information to a sequential access file
Dim outFile As IO.StreamWriter
Dim strFileName As String

' get the filename
strFileName = InputBox("Filename", "Jotpad")

If strFileName <> String.Empty Then
    ' open the file for output
    outFile = IO.File.CreateText(strFileName)
    ' write the text to the file
    outFile.WriteLine(ActiveMdiChild.Controls("txtNote").Text)
    MessageBox.Show("Saved to " & strFileName, "Jotpad", MessageBoxButtons.OK, MessageBoxIcon.Information)
    ' close the file
    outFile.Close()
End If

Private Sub mnuFileNew_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles mnuFileNew.Click
' displays a new child window
Dim frmNewChild As New frmChild
Static intNum As Integer
frmNewChild.MdiParent = Me
intNum += 1
frmNewChild.Text = "Document" & intNum.ToString
frmNewChild.Show()
End Sub

Private Sub mnuFileClose_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles mnuFileClose.Click
' closes the current (active) child window
If ActiveMdiChild IsNot Nothing Then
    ActiveMdiChild.Close()
End If
End Sub
```

Figure D-31  Jotpad application's code

Currency Converter Application

In this section, you will create a quasi TDI application using the **TabControl** tool in the toolbox. The application, named Currency Converter, will allow the user to convert American dollars into either British pounds or Mexican pesos.
To add a tab control to the Currency Converter application:

1. Open the Currency Converter Solution (Currency Converter Solution.sln) file contained in the VB2010\AppD\Currency Converter Solution folder. If necessary, open the designer window. The File menu in the interface contains one option: Exit.

2. Click the TabControl tool in the Containers section of the toolbox, and then drag the mouse pointer to the form. Release the mouse button. A tab control appears on the form. The control contains two tab pages.

3. Click Dock in the Properties window. Click the down arrow in the Settings box and then click the middle button to change the property’s setting to Fill. See Figure D-32. You use the tab control’s task box to display its task list, which contains commands for adding and removing tab pages.

4. Click TabPages in the Properties window and then click the ... (ellipsis) button in the Settings box. The TabPage Collection Editor dialog box opens. You can use the dialog box to both add tab pages to and remove tab pages from the control. You also can use it to set the properties of existing tab pages.

5. Click the Alphabetical button to display the properties in alphabetical order. Change TabPage1’s Text property to To Pounds, and change its Name property to tabpgPounds.
6. Click TabPage2 in the Members list. Change TabPage2’s Name property to tabpgPesos, and change its Text property to To Pesos. Figure D-33 shows the completed TabPage Collection Editor dialog box.

7. Click the OK button to close the dialog box.

Each tab page will provide a text box for entering the number of American dollars. It also will provide a button for converting the dollars to the appropriate currency (either pounds or pesos), as well as a label for displaying the converted value.

**To add controls to both tab pages:**

1. Add three labels, a text box, and a button to the To Pounds tab page.
2. Change Label1’s Text property to **American dollars:**.
3. Change Label2’s Text property to **British pounds:**.
4. Change Label3’s Name, AutoSize, BorderStyle, and TextAlign properties to lblPounds, False, FixedSingle, and MiddleCenter, respectively. Also, delete the contents of its Text property.
5. Change the text box’s Name property to txtDollars.
6. Change the button’s Name and Text properties to btnConvertToPounds and **Convert**, respectively.
7. Size and position the controls as shown in Figure D-34.
8. Click **View** on the menu bar and then click **Tab Order**. Use the information shown in Figure D-35 to set the tab order.

![Figure D-35 Tab order for the To Pounds tab page](image)

9. Press **Esc** to remove the tab order boxes.

10. Now, you will copy the controls from the To Pounds tab page to the To Pesos tab page. Select the five controls on the To Pounds tab page and then press **Ctrl+c**. Click the **To Pesos** tab twice to make the To Pesos tab page the active page. Click the **blank area** below the tab and then press **Ctrl+v**. Now, click the **To Pesos** tab to deselect the controls.

11. Change the text box’s name to **txtAmerican**. (The names of controls on the tab pages must be unique. Therefore, you can’t use the name **txtDollars** for this text box because it is used on the To Pounds tab page.)

12. Change the British pounds: text to **Mexican pesos**: Change the name of the label that displays the number of pesos to **lblPesos**. Change the button’s Name property to **btnConvertToPesos**.

13. Click the **To Pesos** tab. Click **View** on the menu bar and then click **Tab Order**. Use the information shown in Figure D-36 to set the tab order.
14. Press Esc to remove the tab order boxes. Lock the controls on the To Pesos tab page and then save the solution. (Locking the controls on one tab page also locks them on all of the tab pages.)

Now that the interface is complete, you can code the Click event procedures for the two Convert buttons on the tab pages.

**To code the Convert buttons:**

1. Open the Code Editor window. The window contains the code for the mnuFileExit object’s Click event procedure. Replace <your name> and <current date> in the comments with your name and the current date, respectively.

2. Open the code template for the btnConvertToPounds control’s Click event procedure. Type the following comment and then press Enter twice:

   ' converts dollars to pounds

3. Next, enter the following declaration statements. Press Enter twice after typing the last statement.

   Const dblPOUND_RATE As Double = .64
   Dim dblDollars As Double
   Dim dblPounds As Double

4. Now, enter the following lines of code:

   Dim dblDollars As Double
   dblDollars = Double.TryParse(txtDollars.Text, dblDollars)
   dblPounds = dblDollars * dblPOUND_RATE
   lblPounds.Text = dblPounds.ToString("N2")

5. Open the code template for the btnConvertToPesos control’s Click event procedure. Type the following comment and then press Enter twice:

   ' converts dollars to pesos

6. Next, enter the following declaration statements. Press Enter twice after typing the last statement.

   Const dblPESO_RATE As Double = 12.36
   Dim dblDollars As Double
   Dim dblPesos As Double
7. Now, enter the following lines of code:

```
Double.TryParse(txtAmerican.Text, dblDollars)
dblPesos = dblDollars * dblPESO_RATE
lblPesos.Text = dblPesos.ToString("N2")
```

Figure D-37 shows the code entered in the Convert buttons’ Click event procedures.

```
Private Sub btnConvertToPounds_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles btnConvertToPounds.Click
    ' converts dollars to pounds
    Const dblPOUND_RATE As Double = 0.64
    Dim dblDollars As Double
    Dim dblPounds As Double
    Double.TryParse(txtDollars.Text, dblDollars)
    dblPounds = dblDollars * dblPOUND_RATE
    lblPounds.Text = dblPounds.ToString("N2")
End Sub
```

```
Private Sub btnConvertToPesos_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles btnConvertToPesos.Click
    ' converts dollars to pesos
    Const dblPESO_RATE As Double = 12.36
    Dim dblDollars As Double
    Dim dblPesos As Double
    Double.TryParse(txtAmerican.Text, dblDollars)
    dblPesos = dblDollars * dblPESO_RATE
    lblPesos.Text = dblPesos.ToString("N2")
End Sub
```

To test the Currency Converter application:

1. Save the solution and then start the application. Click the American dollars box on the To Pounds tab page. Type 100 and then click the Convert button. The number of British pounds appears in the British pounds box. See Figure D-38.
2. Click the To Pesos tab and then click the American dollars box. Type 100 and then click the Convert button. The number of Mexican pesos (1,236.00) appears in the Mexican pesos box.

3. Click File on the application’s menu bar and then click Exit. Close the Code Editor window and then close the solution.

**Key Terms**

**ActiveMdiChild property**—a property of a parent form; used to determine the active child window (if any)

**Color dialog box**—one of the standard dialog boxes available in Visual Basic; created using the ColorDialog tool, which is contained in the Dialogs section of the toolbox

**Font dialog box**—one of the standard dialog boxes available in Visual Basic; created using the FontDialog tool, which is contained in the Dialogs section of the toolbox

**FormClosing event**—occurs immediately before a form is closed

**Friend keyword**—can be used to declare variables that are recognized by any form within the current solution

**Hide method**—used to hide a form

**IsMdiContainer property**—a property of a form; used to specify that the form is a parent form

**MDI**—an acronym for multiple-document interface

**MDI application**—an application that consists of a parent window and one or more child windows that are contained within the parent window

**MdiWindowListItem property**—a property of a menu strip control; specifies the menu that lists the open child windows in an MDI application

**Multiple-document interface**—the interface used in an MDI application

**Multiple-form application**—an application that contains more than one form

**PrintForm tool**—contained in the Visual Basic PowerPacks section of the toolbox; allows you to print an interface during run time

**SDI**—an acronym for single-document interface

**SDI application**—an application in which each window holds a single document

**Show method**—used to show a form

**Single-document interface**—the interface used in an SDI application

**Tabbed-document interface**—the interface used in a TDI application

**TabControl tool**—contained in the Containers section of the toolbox; used to display a tab control that contains tab pages

**TDI**—an acronym for tabbed-document interface

**TDI application**—an offshoot of MDI; uses tabs to show the opened child windows