APPENDIX E

Locating Syntax and Logic Errors

After studying Appendix E, you should be able to:

- Locate and correct syntax and logic errors
- Use the Debug menu's Step Into option
- Set and remove a breakpoint

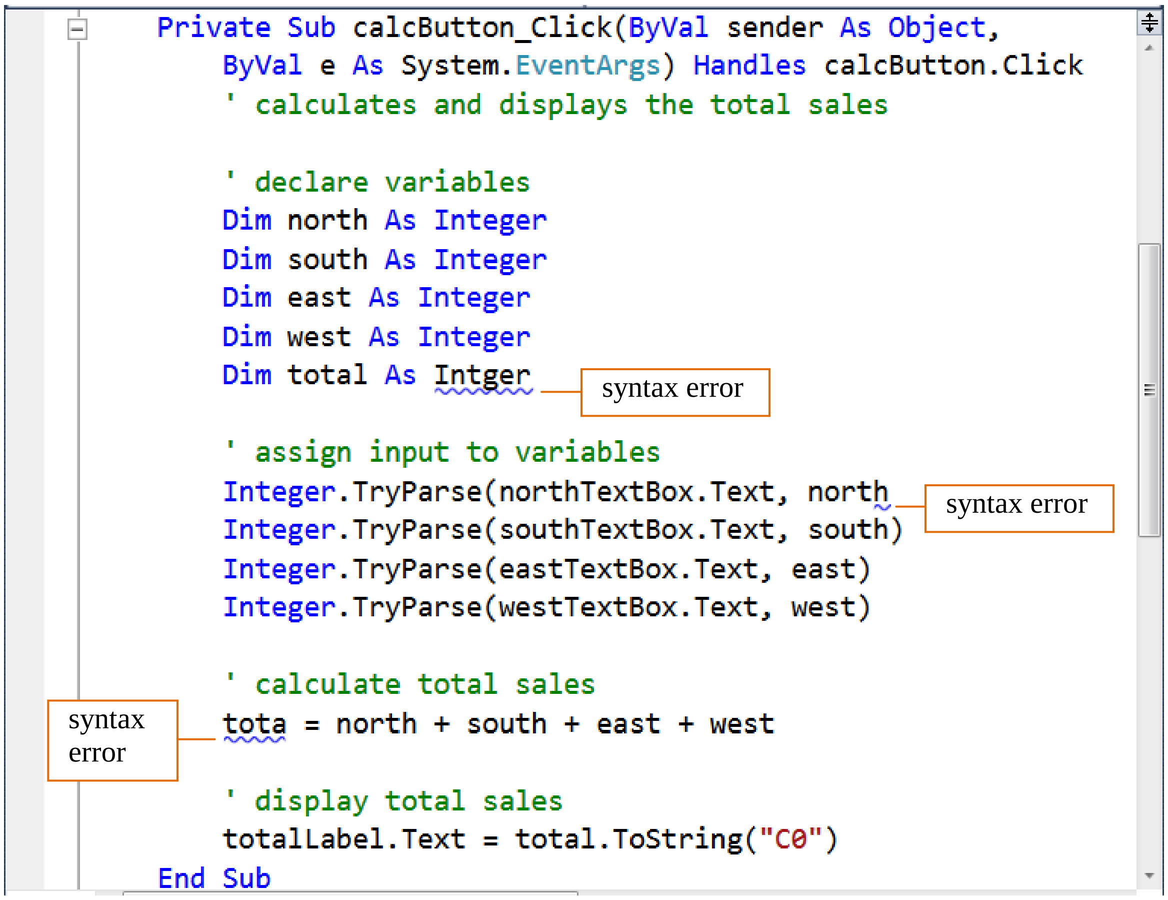
Finding Syntax Errors

As you learned in Chapter 1, a syntax error occurs when you break one of a programming language's rules. Most syntax errors are a result of typing errors that occur when entering instructions, such as typing `Me.Clse()` instead of `Me.Close()`. The Code Editor detects most syntax errors as you enter the instructions. However, if you are not paying close attention to your computer screen, you may not notice the errors. In the following set of steps, you will observe what happens when you try to start an application that contains a syntax error.

To debug the Total Sales Calculator application:

1. Start Visual Studio or the Express Edition of Visual Basic. Open the **Total Sales Solution** (**Total Sales Solution.sln**) file, which is contained in the VbReloaded2010\AppE\Total Sales Solution folder. If necessary, open the designer window. The application calculates and displays the total of the sales amounts entered by the user.

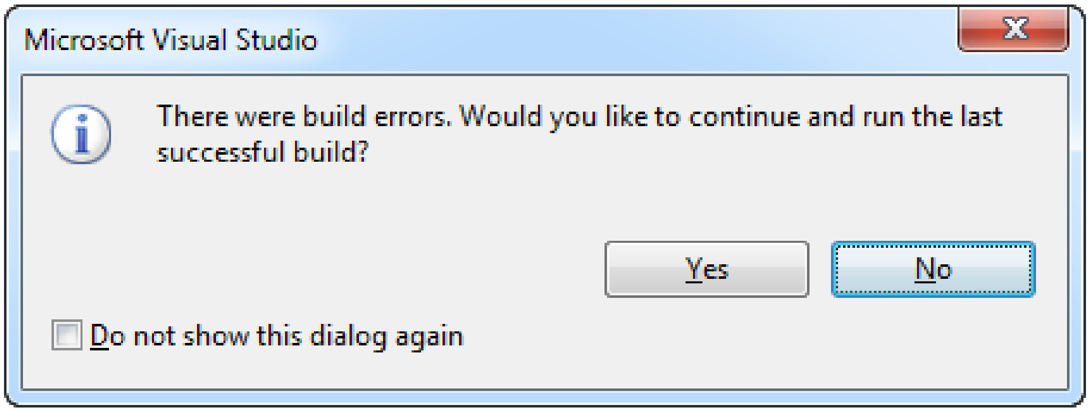
2. Open the Code Editor window.
3. In the comments in the General Declarations section, replace <your name> and <current date> with your name and the current date. Figure E-1 shows the code entered in the calcButton's Click event procedure. The jagged lines alert you that three lines of code contain a syntax error. However, you may fail to notice the jagged lines if you are not paying really close attention to the code.



```
Private Sub calcButton_Click(ByVal sender As Object,
    ByVal e As System.EventArgs) Handles calcButton.Click
    ' calculates and displays the total sales

    ' declare variables
    Dim north As Integer
    Dim south As Integer
    Dim east As Integer
    Dim west As Integer
    Dim total As Intger

    ' assign input to variables
    Integer.TryParse(northTextBox.Text, north
    Integer.TryParse(southTextBox.Text, south)
    Integer.TryParse(eastTextBox.Text, east)
    Integer.TryParse(westTextBox.Text, west)

    ' calculate total sales
    tota = north + south + east + west

    ' display total sales
    totalLabel.Text = total.ToString("C0")
End Sub
```

Figure E-1 calcButton's Click event procedure

4. Start the application. If the dialog box shown in Figure E-2 appears, click the **No** button.

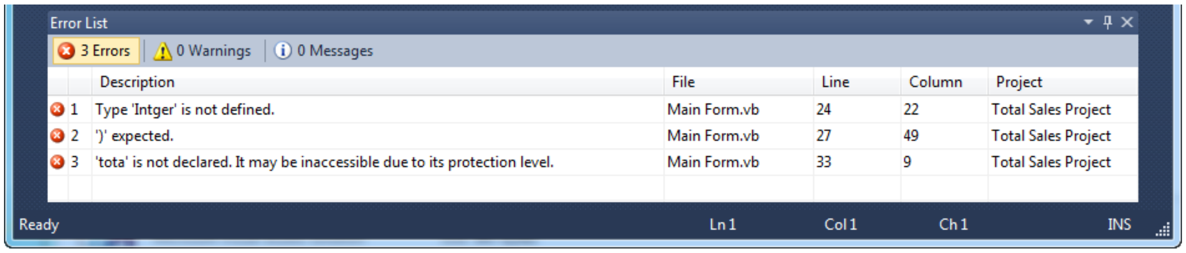


Microsoft Visual Studio

There were build errors. Would you like to continue and run the last successful build?

Figure E-2 Dialog box

5. The Error List window shown in Figure E-3 opens at the bottom of the IDE. The Error List window indicates that the code contains three errors. The window provides a description of each error and the location of each error in the code. If you want to change the size of the Error List window, position your mouse pointer on the window's top border until the mouse pointer becomes a sizing pointer. Then press and hold down the left mouse button while you drag the border either up or down.



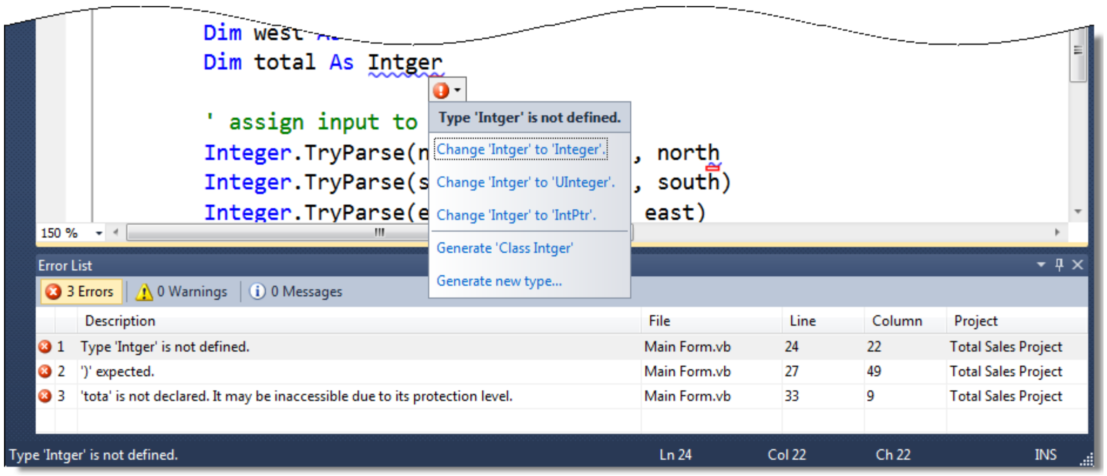
Error List

3 Errors | 0 Warnings | 0 Messages

	Description	File	Line	Column	Project
1	Type 'Intger' is not defined.	Main Form.vb	24	22	Total Sales Project
2	')' expected.	Main Form.vb	27	49	Total Sales Project
3	'tota' is not declared. It may be inaccessible due to its protection level.	Main Form.vb	33	9	Total Sales Project

Figure E-3 Error List window

6. Double-click **the first error message** in the Error List window. A list of suggestions for fixing the error appears in a box. See Figure E-4.



```
Dim west As
Dim total As Intger

' assign input to
Integer.TryParse(n                    , north
Integer.TryParse(s                    , south)
Integer.TryParse(e                    east)
```

Type 'Intger' is not defined.
Change 'Intger' to 'Integer'.
Change 'Intger' to 'UInteger'.
Change 'Intger' to 'IntPtr'.
Generate 'Class Intger'
Generate new type...

Error List — 3 Errors | 0 Warnings | 0 Messages

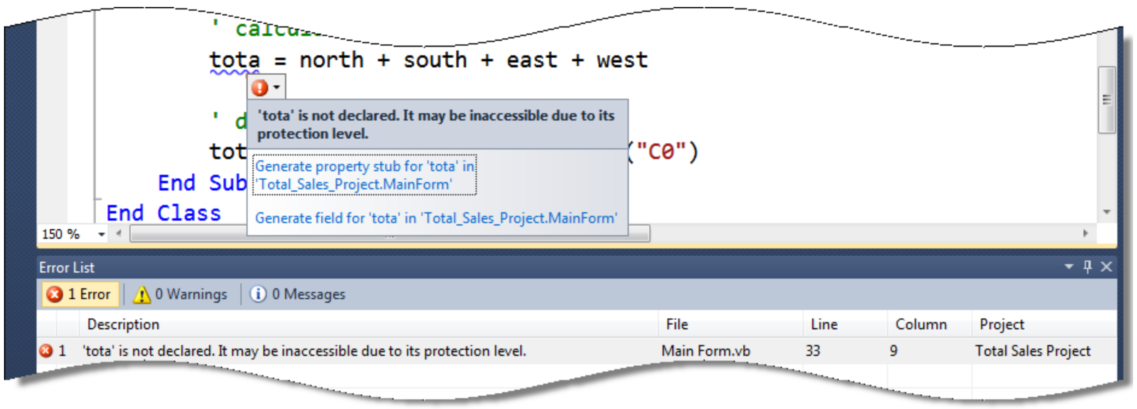
	Description	File	Line	Column	Project
1	Type 'Intger' is not defined.	Main Form.vb	24	22	Total Sales Project
2	')' expected.	Main Form.vb	27	49	Total Sales Project
3	'tota' is not declared. It may be inaccessible due to its protection level.	Main Form.vb	33	9	Total Sales Project

Type 'Intger' is not defined. Ln 24 Col 22 Ch 22 INS

Figure E-4 Result of double-clicking the first error message

HELP: If the suggestion box does not appear, hover your mouse pointer over the thin red box that appears below the letter r. An Error icon (a white exclamation point in a red circle) appears along with a down arrow. (If you don't see the down arrow, hover your mouse pointer over the Error icon until the down arrow appears.) Click the down arrow.

7. The first error is nothing more than a typing error. In this case, the programmer meant to type `Integer`. Click the **Change 'Intger' to 'Integer'.** suggestion. The Code Editor changes `Intger` to `Integer` in the Dim statement and then removes the error from the Error List window.
8. Double-click **')' expected.** in the Error List window. Click **Insert the missing ')'.** in the list of suggestions. The Code Editor inserts the missing parenthesis and then removes the error message from the Error List window.
9. Double-click **the remaining error message** in the Error List window. The Code Editor offers two suggestions for fixing the error, as shown in Figure E-5.


```
' calcu
tota = north + south + east + west

' d
tot                                   ("C0")
End Sub
End Class
```

'tota' is not declared. It may be inaccessible due to its protection level.

Generate property stub for 'tota' in 'Total_Sales_Project.MainForm'

Generate field for 'tota' in 'Total_Sales_Project.MainForm'

	Description	File	Line	Column	Project
1	'tota' is not declared. It may be inaccessible due to its protection level.	Main Form.vb	33	9	Total Sales Project

Figure E-5 Result of double-clicking the remaining error message

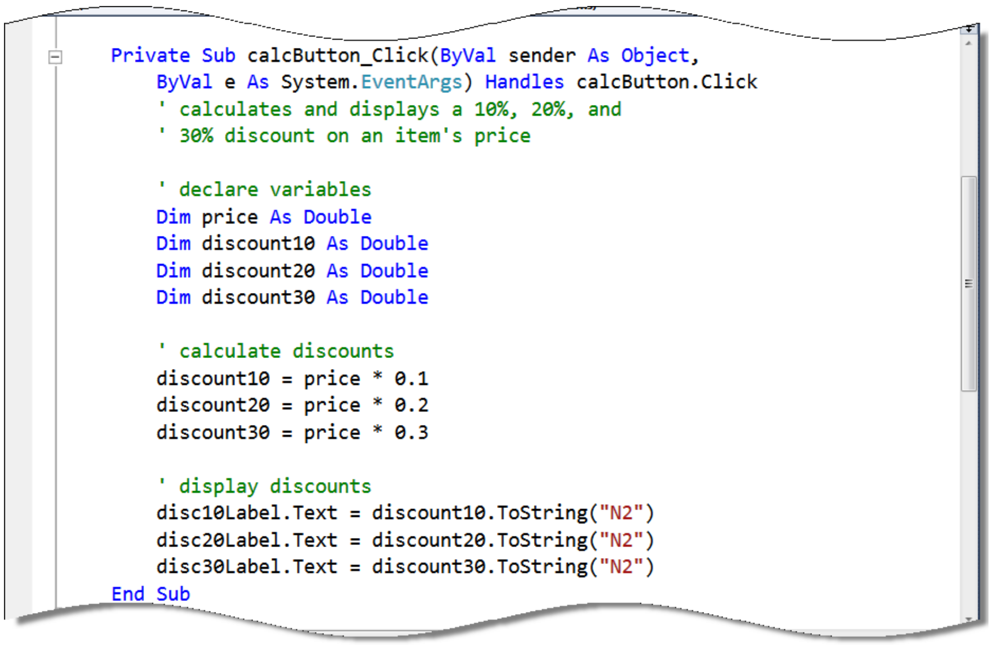
10. Neither suggestion shown in Figure E-5 is appropriate in this case. The error's description indicates that the Code Editor does not recognize the name `tota`. The unrecognized name appears on the left side of an assignment statement, so it belongs to something that can store information: either the property of a control or a variable. It's not the name of either a control or a property, so it must be the name of a variable. Looking at the variable declarations at the beginning of the procedure, you will notice that the procedure declares a variable named `total`. Obviously, the programmer mistyped the variable's name. Change `tota` to **total** in the assignment statement, and then move the insertion point to another line in the Code Editor window. When you do this, the Code Editor removes the error message from the Error List window.
11. Close the Error List window. Save the solution and then start the application. Test the application using **2000** as the North sales, **3000** as the South sales, **1200** as the East sales, and **1800** as the West sales. Click the **Calculate** button. The total sales are $8,000. Click the **Exit** button to end the application. Close the Code Editor window and then close the solution.

Locating Logic Errors

As you observed in the previous section, the Code Editor makes syntax errors easy to find and correct. A much more difficult type of error to locate, and one that the Code Editor cannot detect, is a logic error. A logic error can occur for a variety of reasons, such as forgetting to enter an instruction or entering the instructions in the wrong order. Some logic errors occur as a result of calculation statements that are correct syntactically, but incorrect mathematically. An example of this is the `radiusSquared = radius + radius` statement. The statement's syntax is correct, but it is incorrect mathematically: you square a value by multiplying it by itself, not by adding it to itself. In the remainder of this appendix, you will debug two applications that contain logic errors.

To debug the Discount Calculator application:

1. Open the **Discount Solution** (**Discount Solution.sln**) file, which is contained in the VbReloaded2010\AppE\Discount Solution folder. If necessary, open the designer window. The application calculates and displays three discount amounts, which are based on the price entered by the user.
2. Open the Code Editor window. In the comments in the General Declarations section, replace <your name> and <current date> with your name and the current date. Figure E-6 shows the code entered in the calcButton's Click event procedure.

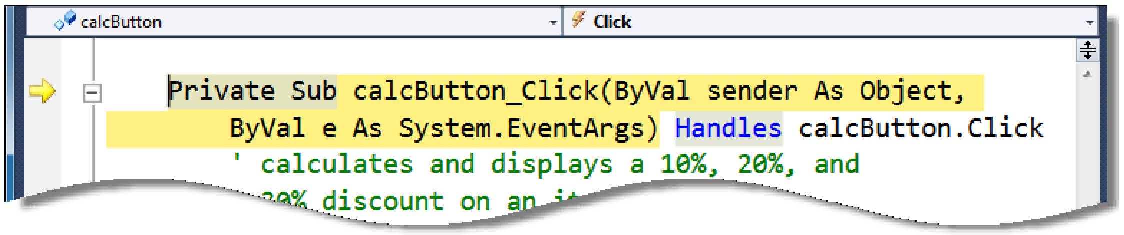


```
Private Sub calcButton_Click(ByVal sender As Object,
    ByVal e As System.EventArgs) Handles calcButton.Click
    ' calculates and displays a 10%, 20%, and
    ' 30% discount on an item's price

    ' declare variables
    Dim price As Double
    Dim discount10 As Double
    Dim discount20 As Double
    Dim discount30 As Double

    ' calculate discounts
    discount10 = price * 0.1
    discount20 = price * 0.2
    discount30 = price * 0.3

    ' display discounts
    disc10Label.Text = discount10.ToString("N2")
    disc20Label.Text = discount20.ToString("N2")
    disc30Label.Text = discount30.ToString("N2")
End Sub
```

Figure E-6 Code entered in the calcButton's Click event procedure

3. Start the application. Type **100** in the Price box and then click the **Calculate** button. The interface shows that each discount is 0.00, which is incorrect. Click the **Exit** button to stop the application.

4. You'll use the Debug menu to run the Visual Basic debugger, which is a tool that helps you locate the logic errors in your code. Click **Debug** on the menu bar. The menu's Step Into option will start your application and allow you to step through your code. It does this by executing the code one statement at a time, pausing immediately before each statement is executed. Click **Step Into**. Type **100** in the Price box and then click the **Calculate** button. The debugger highlights the first instruction to be executed. In this case, it highlights the calcButton_Click procedure header. In addition, an arrow points to the instruction, as shown in Figure E-7, and the code's execution is paused. (If the interface still appears on the screen, click the Code Editor window's title bar.)

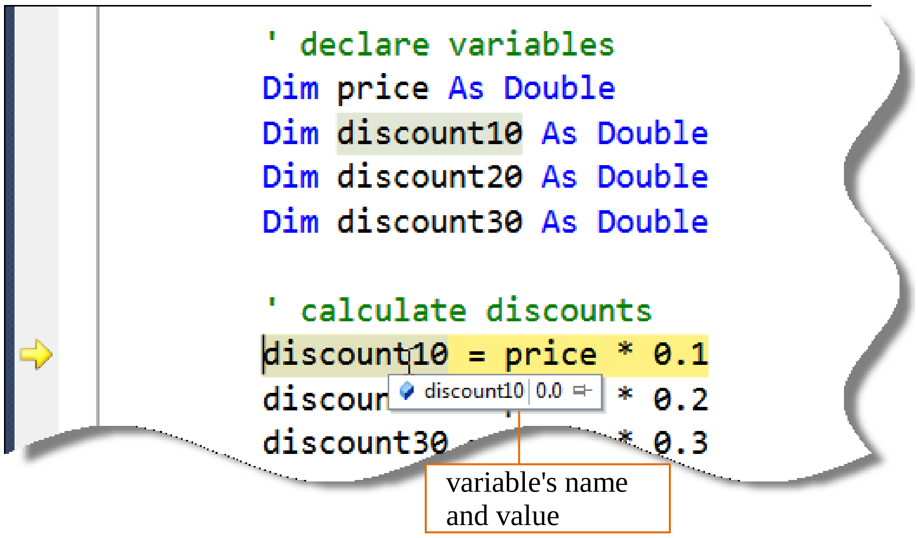


```
    Private Sub calcButton_Click(ByVal sender As Object,
        ByVal e As System.EventArgs) Handles calcButton.Click
        ' calculates and displays a 10%, 20%, and
        '30% discount on an it
```

Figure E-7 Procedure header highlighted

5. To execute the highlighted instruction, you can use either the Debug menu's Step Into option or the F8 key on your keyboard. Press the **F8** key. After the computer processes the procedure header, the debugger highlights the next statement to be processed—in this case, the `discount10 = price * 0.1` statement—and then pauses execution of the code. (The Dim statements are skipped over because they are not considered executable by the debugger.)

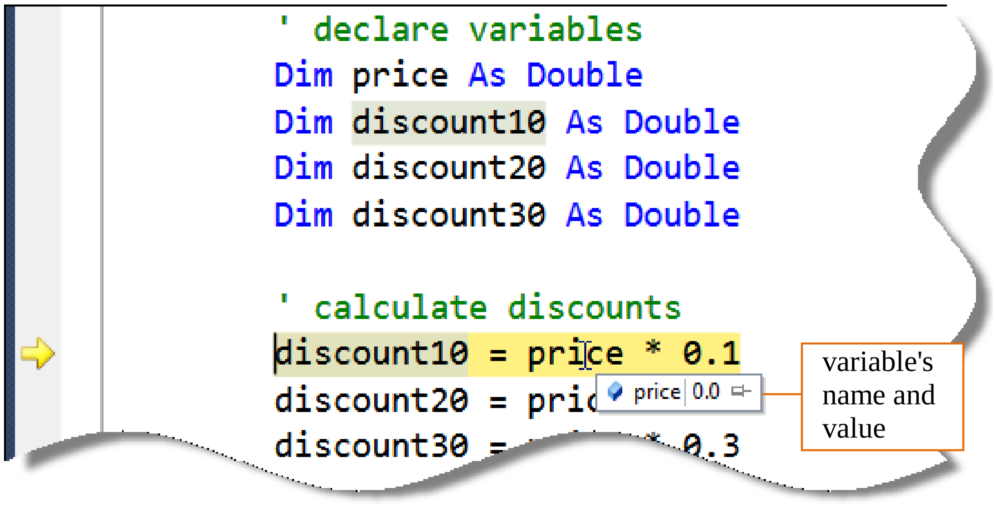
6. While the execution of a procedure's code is paused, you can view the contents of properties and variables that appear in the highlighted statement, as well as in the statements above it in the procedure. Before you view the contents of a property or variable, however, you should consider the value you expect to find. Before the `discount10 = price * 0.1` statement is processed, the `discount10` variable should contain its initial value, 0. (Recall that the Dim statement initializes numeric variables to 0.) Place your mouse pointer on `discount10` in the highlighted statement. The variable's name and current value appear in a small box, as shown in Figure E-8. (The .0 indicates that the value's data type is Double.) At this point, the `discount10` variable's value is correct.



```
' declare variables
Dim price As Double
Dim discount10 As Double
Dim discount20 As Double
Dim discount30 As Double

' calculate discounts
discount10 = price * 0.1
discour                  * 0.2
discount30             * 0.3
```

Figure E-8 Value stored in the `discount10` variable before the highlighted statement is executed

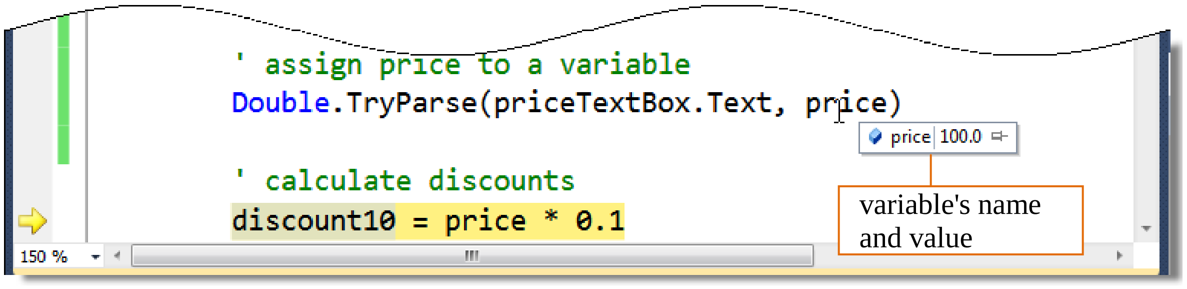
7. Now consider the value you expect the `price` variable to contain. Before the highlighted statement is processed, the `price` variable should contain the number 100, which is the value you entered in the Price box. Place your mouse pointer on `price` in the highlighted statement. As Figure E-9 shows, the `price` variable contains 0.0, which is its initial value. Consider why the variable's value is incorrect. In this case, the value is incorrect because no statement above the highlighted statement assigns the Price box's value to the `price` variable. In other words, a statement is missing from the procedure.



```
' declare variables
Dim price As Double
Dim discount10 As Double
Dim discount20 As Double
Dim discount30 As Double

' calculate discounts
discount10 = price * 0.1
discount20 = pric
discount30 =          0.3
```

Figure E-9 Value stored in the `price` variable before the highlighted statement is executed

8. Click **Debug** on the menu bar and then click **Stop Debugging** to stop the debugger. Click the **blank line** below the last Dim statement and then press **Enter** to insert another blank line. Enter the following comment and TryParse method.
   ‘ assign price to a variable
   Double.TryParse(priceTextBox.Text, price)
9. Save the solution. Click **Debug** on the menu bar and then click **Step Into**. Type **100** in the Price box and then click the **Calculate** button. (If the interface still appears on the screen, click the Code Editor window’s title bar.) Press **F8** to process the procedure header. The debugger highlights the `Double.TryParse(priceTextBox.Text, price)` statement and pauses execution of the code.
10. Before the highlighted statement is processed, the priceTextBox’s Text property should contain 100, which is the value you entered in the control. Place your mouse pointer on priceTextBox.Text in the highlighted statement. The box shows that the Text property contains the expected value. The 100 is enclosed in quotation marks because it is considered a string.
11. The `price` variable should contain its initial value, 0.0. Place your mouse pointer on `price` in the highlighted statement. The box shows that the variable contains the expected value.
12. Press **F8** to process the TryParse method. The debugger highlights the `discount10 = price * 0.1` statement before pausing execution of the code. Place your mouse pointer on `price` in the TryParse method, as shown in Figure E-10. Notice that after the method is processed by the computer, the `price` variable contains the number 100.0.



```
' assign price to a variable
Double.TryParse(priceTextBox.Text, price)

' calculate discounts
discount10 = price * 0.1
```

Figure E-10 Value stored in the `price` variable after the TryParse method is executed

13. Before the highlighted statement is executed, the `discount10` variable should contain its initial value, and the `price` variable should contain the value assigned to it by the TryParse method. Place your mouse pointer on `discount10` in the highlighted statement. The box shows that the variable contains 0.0, which is correct. Place your mouse pointer on `price` in

the highlighted statement. The box shows that the variable contains 100.0, which also is correct.

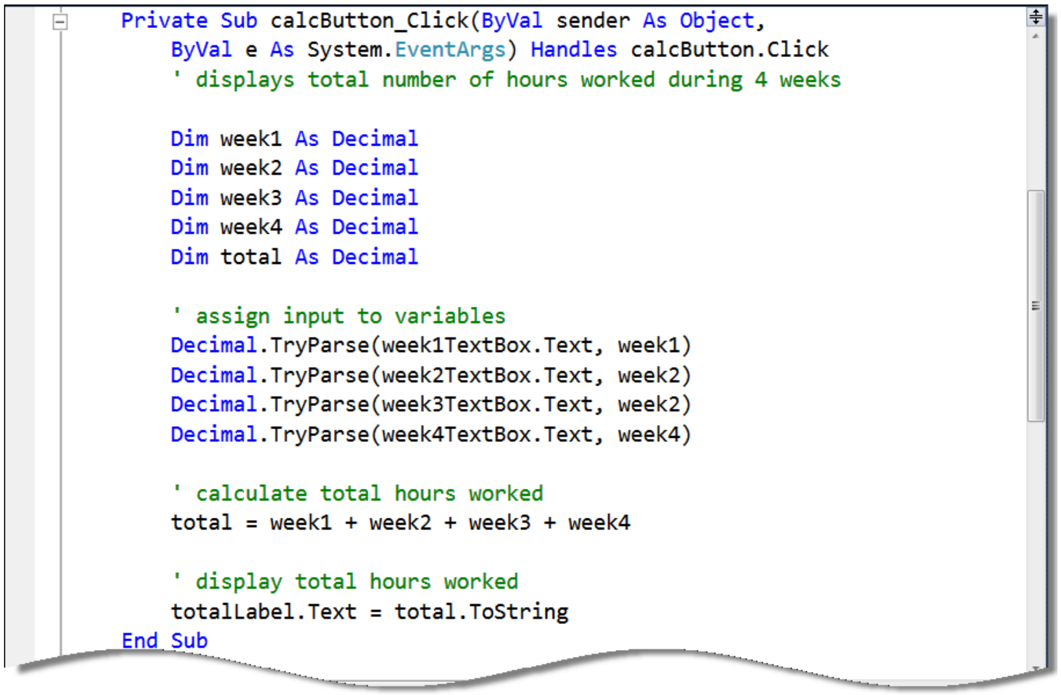
14. After the highlighted statement is processed, the `price` variable should still contain 100.0. However, the `discount10` variable should contain 10.0, which is 10% of 100.0. Press **F8** to execute the `discount10 = price * 0.1` statement, and then place your mouse pointer on `discount10` in the statement. The box shows that the variable contains the expected value. On your own, verify that the `price` variable in the statement contains the appropriate value.
15. To continue program execution without the debugger, click **Debug** on the menu bar and then click **Continue**. This time, the correct discount amounts (10.00, 20.00, and 30.00) appear in the interface. Click the **Exit** button to end the application. Close the Code Editor window and then close the solution.

Setting Breakpoints

Stepping through code one line at a time is not the only way to search for logic errors. You also can use a breakpoint to pause execution at a specific line in the code. You will learn how to set a breakpoint in the following set of steps.

To debug the Hours Worked application:

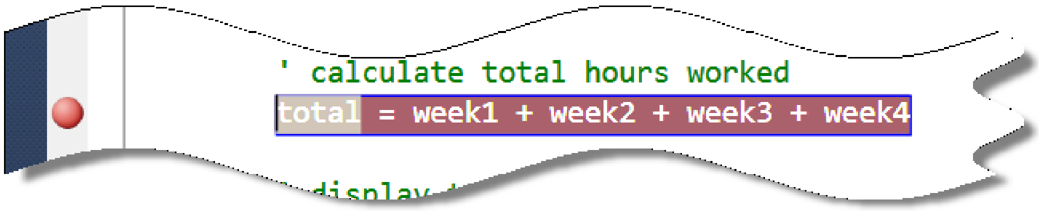
1. Open the **Hours Worked Solution** (**Hours Worked Solution.sln**) file, which is contained in the VbReloaded2010\AppE\Hours Worked Solution folder. If necessary, open the designer window. The application calculates and displays the total number of hours worked during four weeks.
2. Open the Code Editor window. In the comments in the General Declarations section, replace <your name> and <current date> with your name and the current date. Figure E-11 shows the code entered in the calcButton's Click event procedure.



```
Private Sub calcButton_Click(ByVal sender As Object,
    ByVal e As System.EventArgs) Handles calcButton.Click
    ' displays total number of hours worked during 4 weeks

    Dim week1 As Decimal
    Dim week2 As Decimal
    Dim week3 As Decimal
    Dim week4 As Decimal
    Dim total As Decimal

    ' assign input to variables
    Decimal.TryParse(week1TextBox.Text, week1)
    Decimal.TryParse(week2TextBox.Text, week2)
    Decimal.TryParse(week3TextBox.Text, week2)
    Decimal.TryParse(week4TextBox.Text, week4)

    ' calculate total hours worked
    total = week1 + week2 + week3 + week4

    ' display total hours worked
    totalLabel.Text = total.ToString
End Sub
```

Figure E-11 Click event procedure for the calcButton

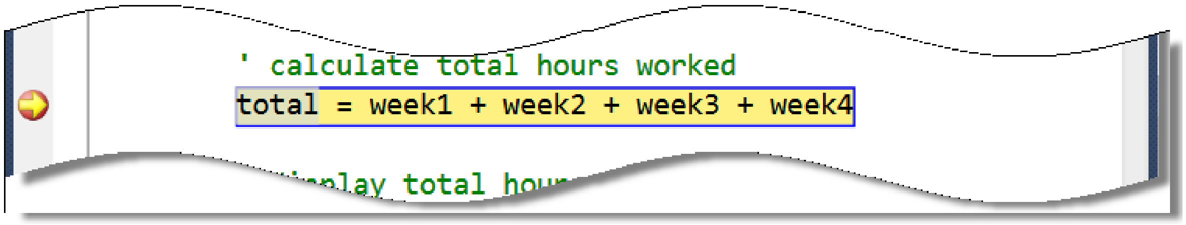
3. Start the application. Type the number **1** in each of the four text boxes and then click the **Calculate** button. The interface shows that the total number of hours is 3, which is incorrect; it should be 4. Click the **Exit** button to stop the application.
4. Obviously, something is wrong with the statement that calculates the total number of hours worked. Rather than having the computer pause before processing each line of code in the procedure, you will have it pause only before processing the calculation statement. You do this by setting a breakpoint on the statement. Right-click the **calculation statement**, point to **Breakpoint**, and then click **Insert Breakpoint**. (You also can set a breakpoint by clicking the statement and then using the Toggle Breakpoint option on the Debug menu. Or, you can simply click in the gray margin next to the statement.) The debugger highlights the statement and places a circle next to it, as shown in Figure E-12.



```
' calculate total hours worked
total = week1 + week2 + week3 + week4
```

Figure E-12 Breakpoint set in the procedure

5. Start the application. Type the number **1** in each of the four text boxes and then click the **Calculate** button. The computer begins processing the code contained in the button's Click event procedure. It stops processing when it reaches the calculation statement, which it highlights. The highlighting indicates that the statement is the next one to be processed. See Figure E-13.



```
' calculate total hours worked
total = week1 + week2 + week3 + week4
```

Figure E-13 Result of the computer reaching the breakpoint

6. Before viewing the values contained in each variable in the highlighted statement, consider the values you expect to find. Before the calculation statement is processed, the `total` variable should contain its initial value (0). The other four variables should contain the number 1, which is the value you entered in each text box. Place your mouse pointer on `total` in the highlighted statement. The box shows that the variable's value is 0D, which is correct. (You can verify the variable's initial value by placing your mouse pointer on `total` in its declaration statement.) Don't be concerned that 0D appears rather than 0. As you learned in Chapter 3, the letter D is one of the literal type characters in Visual Basic; it indicates that the value's data type is Decimal.
7. On your own, view the values contained in the `week1`, `week2`, `week3`, and `week4` variables. Notice that three of the variables contain 1D, which is correct. However, the `week3` variable contains its initial value (0D), which is incorrect.
8. One of the TryParse methods is responsible for assigning a new value to the `week3` variable. Looking closely at the four TryParse methods in the procedure, you will notice that the third one is incorrect. After converting the contents of the week3TextBox to a number, the method should assign the number to the `week3` variable rather than to the `week2` variable. Click **Debug** on the menu bar and then click **Stop Debugging**.
9. Change `week2` in the third TryParse method to **week3**.

10. Now you can remove the breakpoint. Right-click the **statement containing the breakpoint**, point to **Breakpoint**, then click **Delete Breakpoint**. (Or, you can simply click the breakpoint circle.)
11. Save the solution and then start the application. Enter the number **1** in each of the four text boxes and then click the **Calculate** button. The interface shows that the total number of hours is 4.
12. On your own, test the application using other values for the hours worked in each week. When you are finished testing, click the **Exit** button to end the application. Close the Code Editor window and then close the solution.

You have completed Appendix E.