8.2 StreamReaders, StreamWriters, and Structured Exception Handling

So far we have accessed the data in a text file by filling an array with the lines of the file and then accessing the array. However, sometimes we want to read or write a text file directly one line at a time without using arrays as intermediaries.

VideoNote

StreamReaders and StreamWriters

Reading a Text File with a StreamReader

Lines of a text file can be read in order and assigned to variables with the following steps:

1. Execute a statement of the form

```
Dim srVar As IO.StreamReader
```

A StreamReader is a class from the Input/Output namespace that can read a stream of characters coming from a disk. The Dim statement declares the variable *srVar* to be of type StreamReader.

2. Execute a statement of the form

```
srVar = IO.File.OpenText(filespec)
```

where *filespec* identifies the text file to be read. If *filespec* consists only of a filename (that is, if no path is given), Visual Basic will look for the file in the program's *bin\Debug* folder. This statement, which establishes a communications link between the computer and the disk drive for reading data from the disk, is said to **open a file for input**. Data then can be input from the specified file and assigned to variables in the program.

Just as with other variables, the declaration and assignment statements in Steps 1 and 2 can be combined into the single statement

```
Dim srVar As IO.StreamReader = IO.File.OpenText(filespec)
```

3. Read lines in order, one at a time, from the file with the ReadLine method. Each line is retrieved as a string. A statement of the form

```
strVar = srVar.ReadLine
```

causes the program to look in the file for the next unread line of data and assign it to the variable *strVar*.

The OpenText method sets a pointer to the first line in the specified file. Each time a ReadLine method is executed, the line pointed to is read, and the pointer is then moved to the next line. After all lines have been read from the file, the value of

```
srVar.EndOfStream
```

will be True. The EndOfStream property can be used in the condition of a Do loop to cycle through every line of a text file. Such a loop might begin with the statement `Do Until srVar.EndOfStream`.

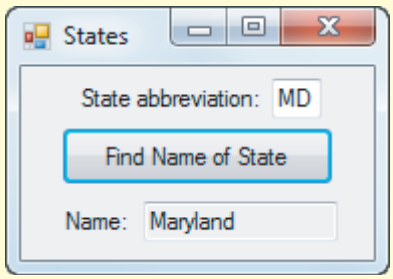
4. After the desired lines have been read from the file, terminate the communications link set in Step 2 with the statement

```
srVar.Close()
```

Example 1 The following program uses a StreamReader to carry out the same task as the program in Example 7 of the previous section. That is, it finds the name of the state whose abbreviation is given in a masked text box.

```
Private Sub btnFind_Click(...) Handles btnFind.Click
  Dim sr As IO.StreamReader = IO.File.OpenText("USStates.txt")
  Dim abbr As String = mtbAbbr.Text.ToUpper   'mask is LL
  Dim line As String
  Dim foundFlag As Boolean = False
  Do Until foundFlag Or sr.EndOfStream
    line = sr.ReadLine
    If line.Split(","c)(1) = abbr Then
      txtName.Text = line.Split(","c)(0)
      foundFlag = True
    End If
  Loop
  If Not foundFlag Then
    Dim str As String = " is not a valid state abbreviation."
    MessageBox.Show(mtbAbbr.Text.ToUpper & str, "Error")
    mtbAbbr.Clear()
    mtbAbbr.Focus()
  End If
End Sub
```

[Run, enter a state abbreviation into the masked text box, and click on the button.]



Creating a Text File with a StreamWriter

In Section 8.1 we created text files by copying the contents of an array into the files with the WriteAllLines method. However, sometimes we want to write to a text file directly one line at a time without using an array. The following steps create a new text file and write data to it.

1. Choose a *filespec*.
2. Execute a statement of the form

   ```
   Dim swVar As IO.StreamWriter = IO.File.CreateText(filespec)
   ```

   where *swVar* is a variable name. This process is said to **open a file for output**. It establishes a communications link between the program and the disk drive for storing data onto the disk. It allows data to be output from the program and recorded in the specified file. If *filespec* consists only of a filename (that is, if no path is given), Visual Basic will place the file in the program's *bin\Debug* folder. ***Caution:*** If an existing file is opened for output, Visual Basic will replace the file with a new empty file.

3. Place data into the file with the WriteLine method. If *info* is a literal, variable, or expression of any data type, then the statement

   ```
   swVar.WriteLine(info)
   ```

   writes the information into a new line of the file.

4. After all the data have been recorded in the file, execute

   ```
   swVar.Close()
   ```

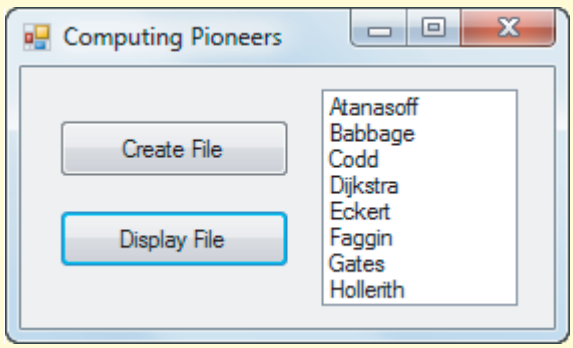
   This statement is very important because the WriteLine method actually places data into a temporary buffer, and the Close method transfers the data to the disk. Therefore, if you omit the statement `swVar.Close()`, some data might be lost. The statement also breaks the communications link with the file. Therefore its omission might prevent other procedures from accessing the file.

Example 2 The following program creates a text file consisting of several last names of computer pioneers and then displays the entire contents of the file in a list box.

```
Private Sub btnCreateFile_Click(...) Handles btnCreateFile.Click
  'Create the file Pioneers.txt
  Dim sw As IO.StreamWriter = IO.File.CreateText("Pioneers.txt")
  sw.WriteLine("Atanasoff")
  sw.WriteLine("Babbage")
  sw.WriteLine("Codd")
  sw.WriteLine("Dijkstra")
  sw.WriteLine("Eckert")
  sw.WriteLine("Faggin")
  sw.WriteLine("Gates")
  sw.WriteLine("Hollerith")
  sw.Close()   'If this line is omitted, the file will be empty.
  MessageBox.Show("Names recorded in file", "File Status")
End Sub

Private Sub btnDisplayFile_Click(...) Handles btnDisplayFile.Click
  'Display the contents of the file Pioneers.txt in a list box
  Dim sr As IO.StreamReader = IO.File.OpenText("Pioneers.txt")
  lstNames.Items.Clear()
  Do Until sr.EndOfStream
    lstNames.Items.Add(sr.ReadLine)
  Loop
End Sub
```

[Run, click on the first button, and then click on the second button.]



Adding Items to a Text File

Data can be added to the end of an existing text file with the following steps.

1. Execute the statement

   ```
   Dim swVar As IO.StreamWriter = IO.File.AppendText(filespec)
   ```

   where *swVar* is a variable name and *filespec* identifies the file. This process is said to **open a file for append**. It allows data to be output and recorded at the end of the specified file. If *filespec* consists only of a filename (that is, if no path is given), Visual Basic will look for the file in the program's *bin\Debug* folder.
2. Place data into the file with the WriteLine method.
3. After all the data have been recorded into the file, close the file with the statement

   ```
   swVar.Close()
   ```

The IO.File.AppendText option is used to add data to an existing file. However, it also can be used to create a new file. If the file does not exist, then the IO.File.AppendText option creates the file.

The three states "open for input," "open for output," and "open for append" are referred to as **modes**. A file should not be open in two modes at the same time. For instance, after a file has been opened for output and data have been written to it, the file should be closed before being opened for input. Had the statement `swVar.Close()` in Example 2 been omitted, then the program would have crashed when the second button was clicked on.

An attempt to open a nonexistent file for input terminates the program with a FileNotFoundException message box, stating that the file could not be found. There is a method that tells us whether a certain file already exists. If the value of

```
IO.File.Exists(filespec)
```

is True, then the specified file exists. Therefore, prudence dictates that files be opened for input with code such as

```
Dim sr As IO.StreamReader
If IO.File.Exists(filespec) Then
  sr = IO.File.OpenText(filespec)
Else
  message = "Either no file has yet been created or the file "
  message &= filespec & " is not where expected."
  MessageBox.Show(message, "File Not Found")
End If
```

There is one file-management operation that we have yet to discuss: changing or deleting an item of information from a text file. An individual item of a file cannot be changed or deleted directly. A new file must be created by reading each item from the original file and recording it, with the single item changed or deleted, into the new file. The old file is then erased, and the new file is renamed with the name of the original file. Regarding these last two tasks, the Visual Basic statement

```
IO.File.Delete(filespec)
```

removes the specified file from the disk, and the statement

```
IO.File.Move(oldfilespec, newfilespec)
```

changes the filespec of a file. ***Note 1:*** The IO.File.Delete and IO.File.Move methods cannot be used with open files; doing so generates an exception. ***Note 2:*** Nothing happens if the file

referenced in an IO.File.Delete method doesn't exist. However, a nonexistent *oldfilespec* in an IO.File.Move method generates an exception.

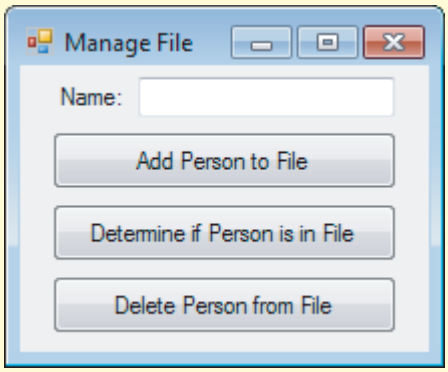
System.IO Namespace

Creating a program that has extensive file handling can be simplified by placing the statement

```
Imports System.IO
```

at the top of the Code Editor, before the `Class frmName` statement. Then there is no need to insert the prefix "IO." before the words StreamReader, StreamWriter, and File.

Example 3 The following program manages Names.txt, a file of names. The Boolean function IsInFile returns the value True if the file Names.txt exists and the name entered in the text box is in the file.



OBJECT	PROPERTY	SETTING
frmNames	Text	Manage File
lblName	Text	Name
txtName		
btnAdd	Text	Add Person to File
btnDetermine	Text	Determine if Person is in File
btnDelete	Text	Delete Person from File

```
Imports System.IO     'Appears at top of Code Editor

Private Sub btnAdd_Click(...) Handles btnAdd.Click
  'Add a person's name to the file
  Dim person As String = txtName.Text
  If person <> "" Then
    If IsInFile(person) Then
      MessageBox.Show(person & " is already in the file.", "Alert")
    Else
      Dim sw As StreamWriter = File.AppendText("Names.txt")
      sw.WriteLine(person)
      sw.Close()
      MessageBox.Show(person & " added to file.", "Name Added")
      txtName.Clear()
      txtName.Focus()
    End If
  Else
    MessageBox.Show("You must enter a name.", "Information Incomplete")
  End If
End Sub

Private Sub btnDetermine_Click(...) Handles btnDetermine.Click
  'Determine if a person is in the file
  Dim person As String = txtName.Text
```

```
    If person <> "" Then
      If IsInFile(person) Then
        MessageBox.Show(person & " is in the file.", "Yes")
      Else
        MessageBox.Show(person & " is not in the file.", "No")
      End If
    Else
      MessageBox.Show("You must enter a name.", "Information Incomplete")
    End If
    txtName.Clear()
    txtName.Focus()
  End Sub

  Private Sub btnDelete_Click(...) Handles btnDelete.Click
    'Remove the person in text box from the file
    Dim person As String = txtName.Text
    If person <> "" Then
      If IsInFile(person) Then
        'code to remove person
        Dim sr As StreamReader = File.OpenText("Names.txt")
        Dim sw As StreamWriter = File.CreateText("Temp.txt")
        Dim individual As String
        Do Until sr.EndOfStream
          individual = sr.ReadLine
          If individual <> person Then
            sw.WriteLine(individual)
          End If
        Loop
        sr.Close()
        sw.Close()
        File.Delete("Names.txt")
        File.Move("Temp.txt", "Names.txt")
        MessageBox.Show(person & " removed from file.", "Name Removed")
      Else
        MessageBox.Show(person & " is not in the file.", "Name Not Found")
      End If
    Else
      MessageBox.Show("You must enter a name.", "Information Incomplete")
    End If
    txtName.Clear()
    txtName.Focus()
  End Sub

  Function IsInFile(ByVal person As String) As Boolean
    'Determine if person is currently in a file named Names.txt
    If File.Exists("Names.txt") Then
      Dim sr As StreamReader = File.OpenText("Names.txt")
      Dim individual As String
      Do Until sr.EndOfStream
        individual = sr.ReadLine
        If individual = person Then
          sr.Close()
          Return True
        End If
      Loop
```

```
    sr.Close()
  End If
  Return False
End Function
```

[Run, add, delete some names, or search for some names. After terminating the program, click on the *Refresh* button in the Solution Explorer window, click on the *View All Files* button, and look at Names.txt in the *bin**Debug* subfolder.]

Structured Exception Handling

There are two categories of problems that a software program might encounter when it executes. The first is a *logic error*, which is caused by code that does not perform as intended. Common examples of logic errors are typos, using the wrong formula, and accessing the wrong property value. The second category is an *exception*, which typically occurs due to circumstances beyond the program's control. Two situations where exceptions occur are when invalid data are input and when a file cannot be accessed. (In Chapter 4, we showed another way to prevent invalid-data exceptions from occurring.) For example, if a user enters a word when the program prompts for a number, an exception is generated and the program terminates abruptly. In this situation, the programmer did not employ faulty logic or mistype. If the user had followed the directions, no problem would have occurred. Even though the user is at fault, however, it is still the programmer's responsibility to anticipate exceptions and to include code to work around their occurrence.

The Visual Studio environment contains powerful tools that programmers can use to find and correct bugs. These debugging tools are discussed extensively in Appendix D. This section describes techniques used to anticipate and deal with exceptions (in programming terms, this is called "handling exceptions").

An unexpected problem causes Visual Basic to raise an exception. If the programmer does not explicitly include exception-handling code in the program, then Visual Basic handles an exception with a default handler. This handler terminates execution, displays the exception's message in a window and highlights the line of code where the exception occurred. Consider a program that contains the following code:

```
Dim taxCredit As Double

Private Sub btnCompute_Click(...) Handles btnCompute.Click
  Dim numDependents As Integer
  numDependents = CInt(InputBox("How many dependents?"))
  taxCredit = 1000 * numDependents
End Sub
```

A user with no dependents might just leave the input dialog box blank and click on the *OK* button. If so, Visual Basic terminates the program and displays the box shown in Fig. 8.5. (The problem was caused by the fact that the default value in an input dialog box, the empty string, cannot be converted to an integer. Text boxes also have the empty string as their default value.) It also highlights the fourth line of code, since the exception was thrown while executing the CInt function. The program also would have crashed had the user typed in an answer like "TWO".

A more robust program explicitly handles the previous exception by protecting the code in a Try-Catch-Finally block. This allows the program to continue regardless of whether or not an exception was thrown. The computer tries to execute the code in the Try block. As soon as an exception occurs, execution jumps to the code in the Catch block. Regardless of whether

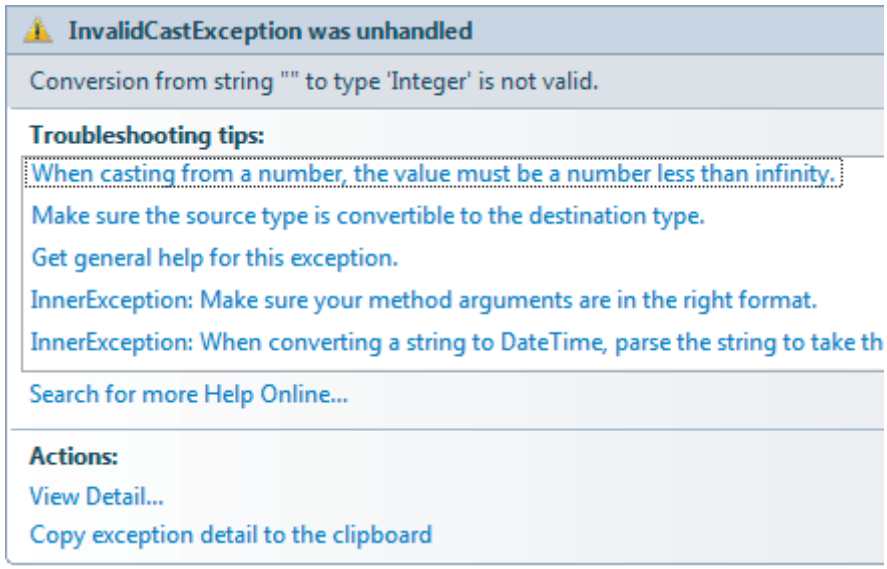


FIGURE 8.5 Exception handled by Visual Basic.

an exception occurred, the code in the Finally block is then executed. The following code is illustrative:

```
Dim taxCredit As Double

Private Sub btnCompute_Click(...) Handles btnCompute.Click
  Dim numDependents As Integer, message As String
  Try
    numDependents = CInt(InputBox("How many dependents?"))
  Catch
    message = "You did not answer the question with an " &
              "integer value. We will assume your answer is zero."
    MessageBox.Show(message, "Improper Response")
    numDependents = 0
  Finally
    taxCredit = 1000 * numDependents
  End Try
End Sub
```

This type of exception handling is known as **data validation**. It catches situations where invalid data cannot be converted to a particular type. An exception is thrown if the user enters data that cannot be converted to an integer using the CInt function. Table 8.1 on the next page lists several exceptions and descriptions of why they are thrown.

The Catch block above will be executed when *any* exception occurs. Visual Basic also allows Try-Catch-Finally blocks to have one or more specialized Catch clauses that handle only a specific type of exception. The general form of a specialized Catch clause is

```
Catch exc As ExceptionType
```

where the variable *exc* will be assigned the name of the exception. The code in this block will be executed only when the specified exception occurs.

The general form of a Try-Catch-Finally block is

```
Try
  normal code
Catch exc1 As FirstException
  exception-handling code for FirstException
```

TABLE 8.1 Some common exceptions.

Exception Name	Description and Example
ArgumentOutOfRangeException	An argument to a method is out of range. `str = "Goodbye".Substring(12,3)`

IndexOutOfRangeException	An array's subscript is out of range.
	`Dim arr(3) As Integer`
	`arr(5) = 2`

InvalidCastException	A value cannot be converted to another type. `Dim num As Integer = CInt("one")`

NullReferenceException	A method is called on a variable that is set to Nothing.
	`Dim str As String, len As Integer`
	`len = str.Length`

OverflowException	A number too big for the data type is assigned.
	`Dim num As Integer = 2000000000`
	`num = 2 * num`

IO.DirectoryNotFoundException	A file within a missing folder is accessed.
	`Dim sr As IO.StreamReader =`
	`IO.File.OpenText("C:\BadDir\File.txt")`

IO.FileNotFoundException	A missing file is accessed.
	`Dim sr As IO.StreamReader =`
	`IO.File.OpenText("Missing.txt")`

IO.IOException	Any file-handling exception, including those mentioned above. For instance, an attempt is made to delete or rename an open file, to change the name of a closed file to an already used name, or when a disk drive specified contains no disk. ***Note:*** If a series of IO exceptions is being tested with Catch clauses, this exception should be the last one tested. `IO.File.Move(filespec, AlreadyExistingName)`

```
Catch exc2 As SecondException
  exception-handling code for SecondException
.
.
Catch
  exception-handling code for any remaining exceptions
Finally
  clean-up code
End Try
```

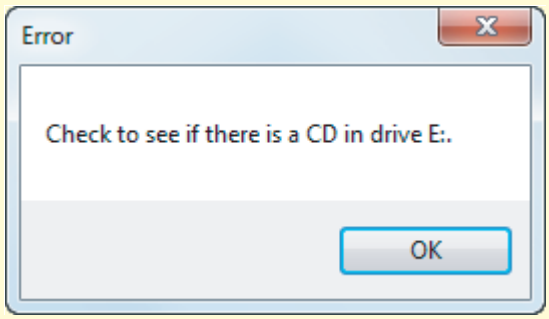
The *normal code* is the code that you want to monitor for exceptions. If an exception occurs during execution of any of the statements in this section, Visual Basic transfers control to the code in one of the Catch blocks. As with a Select Case block, the Catch clauses are considered one at a time until the first matching exception is located. The last Catch clause in the preceding code functions like the Case Else clause. The *clean-up code* in the Finally block always executes last regardless of whether any *exception-handling code* has executed. In most situations, this code *cleans up* any resources such as files that were opened during the *normal code*. If clean-up is not necessary, then the Finally block can be omitted. However, to complete a Try block, a Catch block or a Finally block must appear.

In addition to data validation, a popular use of exception handling is to account for errors when accessing files. Visual Basic has the capability to access files stored on remote servers via the Internet. An exception is thrown if a desired file is missing (IO.FileNotFoundException) or if the file cannot be read because the Internet connection between the computer and the server is broken (IO.IOException).

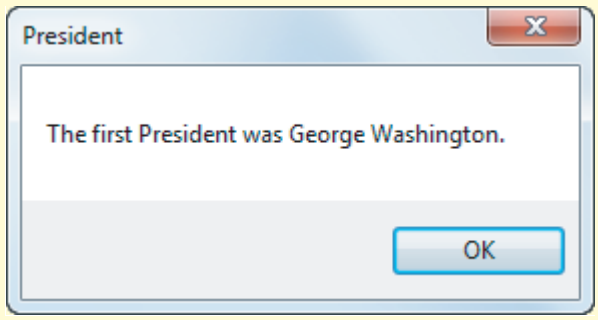
Example 4 The following program reads the first line from a file on a CD. The program expects the file to reside in the folder DataFiles of the CD. Note that the clean-up code, `sr.Close()`, in the Finally block is enclosed in a Try-Catch block of its own. This protects the Close method from any exceptions that might occur.

```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim sr As IO.StreamReader
  Dim message As String
  Try
    sr = IO.File.OpenText("E:\DataFiles\USPres.txt")
    message = "The first President was " & sr.ReadLine & "."
    MessageBox.Show(message, "President")
  Catch exp As IO.DirectoryNotFoundException
    message = "The requested folder is not on the CD."
    MessageBox.Show(message, "Error")
  Catch exp As IO.FileNotFoundException
    message = "The file is not in the specified folder of the CD."
    MessageBox.Show(message, "Error")
  Catch exp As IO.IOException
    message = "Check to see if there is a CD in drive E:."
    MessageBox.Show(message, "Error")
  Finally
    Try
      sr.Close()
    Catch
      'Disregard any exceptions during the Close() method
    End Try
  End Try
End Sub
```

[Remove the CD from the E: drive, run the program, and then click on the button.]



[Insert the CD containing the file USPres.txt (in the folder DataFiles) into the E: drive and then click on the button.]



Comments

1. Any variable declared within a Try-Catch-Finally block has block-level scope. That is, it will not be available after the block terminates.
2. Text files are also called ***sequential files*** because a SteamReader reads the records one at a time in sequence.

Practice Problems 8.2

1. Give three different ways to display the last record of the file USStates.txt in a message box.
2. Consider the following event procedure.

```
Private Sub btnCreate_Click(...) Handles btnCreate.Click
  Dim sw As IO.StreamWriter = IO.File.CreateText("ABC.txt")
  sw.WriteLine("abc")
End Sub
```

Describe the file ABC.txt after the button is pressed.

EXERCISES 8.2

In Exercises 1 through 10, determine the output displayed in the text box when the button is clicked.

1.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim salutation As String
  Dim sw As IO.StreamWriter = IO.File.CreateText("Greetings.txt")
  sw.WriteLine("Hello")
  sw.WriteLine("Aloha")
  sw.Close()
  Dim sr As IO.StreamReader = IO.File.OpenText("Greetings.txt")
  salutation = sr.ReadLine
  txtOutput.Text = salutation
  sr.Close()
End Sub
```

2.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim salutation, welcome As String
  Dim sw As IO.StreamWriter = IO.File.CreateText("Greetings.txt")
  sw.WriteLine("Hello")
  sw.WriteLine("Aloha")
  sw.Close()
  Dim sr As IO.StreamReader = IO.File.OpenText("Greetings.txt")
  salutation = sr.ReadLine
  welcome = sr.ReadLine
  txtOutput.Text = welcome
  sr.Close()
End Sub
```

3.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim salutation As String
  Dim sw As IO.StreamWriter = IO.File.CreateText("Greetings.txt")
  sw.WriteLine("Hello")
  sw.WriteLine("Aloha")
  sw.WriteLine("Bon Jour")
  sw.Close()
  Dim sr As IO.StreamReader = IO.File.OpenText("Greetings.txt")
  Do Until sr.EndOfStream
    salutation = sr.ReadLine
    txtOutput.Text = salutation
  Loop
  sr.Close()
End Sub
```

4. Assume that the contents of the file Greetings.txt are as shown in Figure 8.6.

```
Hello
Aloha
Bon Jour
```

FIGURE 8.6 Contents of the file Greetings.txt.

```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim file, welcome As String
  file = "Greetings.txt"
  Dim sw As IO.StreamWriter = IO.File.AppendText(file)
  sw.WriteLine("Buenos Dias")
  sw.Close()
  Dim sr As IO.StreamReader = IO.File.OpenText(file)
  For i As Integer = 1 To 4
    welcome = sr.ReadLine
    txtOutput.Text = welcome
  Next
  sr.Close()
End Sub
```

5.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim num As Integer
  'Assume that txtBox is empty
```

```
  Try
    num = CInt(txtBox.Text)
    txtOutput.Text = "Your number is " & num
  Catch
    txtOutput.Text = "You must enter a number."
  End Try
End Sub
```

6.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim nafta() As String = {"Canada", "United States", "Mexico"}
  Try
    txtOutput.Text = "The third member of NAFTA is " & nafta(3)
  Catch exc As IndexOutOfRangeException
    txtOutput.Text = "Error occurred."
  End Try
End Sub
```

7.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Try
    Dim usPop As Integer = 304000000    'Approx population of U.S.
    Dim worldPop As Integer
    worldPop = 21 * usPop
    txtOutput.Text = CStr(worldPop)
  Catch exc As ArgumentOutOfRangeException
    txtOutput.Text = "Oops"
  Catch exc As OverflowException
    txtOutput.Text = "Error occurred."
  End Try
End Sub
```

8.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim flower As String = "Bougainvillaea", lastLetter As String
  Try
    lastLetter = flower.Substring(14, 1)
    txtOutput.Text = lastLetter
  Catch exc As InvalidCastException
    txtOutput.Text = "Oops"
  Catch exc As ArgumentOutOfRangeException
    txtOutput.Text = "Error occurred."
  End Try
End Sub
```

9. Assume that the file Ages.txt is located in the *Debug* subfolder of the folder *bin* and the first line of the file is "Twenty-one".

```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim sr As IO.StreamReader
  Dim age As Integer
  Try
    sr = IO.File.OpenText("Ages.txt")   'FileNotFound if fails
    age = CInt(sr.ReadLine)           'InvalidCast if fails
    txtOutput.Text = "Age is " & age
  Catch exc As IO.FileNotFoundException
    txtOutput.Text = "File Ages.txt not found"
```

```
  Catch exc As InvalidCastException
    txtOutput.Text = "File Ages.txt contains an invalid age."
  Finally
    Try
      sr.Close() 'This code executes no matter what happens above
    Catch
     'Disregard any exceptions thrown during the Close() method
    End Try
  End Try
End Sub
```

10. Redo Exercise 9 with the assumption that the file Ages.txt is not located in the *Debug* subfolder of the folder *bin*.

11. Assume that the contents of the file Greetings.txt are as shown in Figure 8.3 (on page 356). What is the effect of the following program?

```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim g As String
  Dim sr As IO.StreamReader = IO.File.OpenText("Greetings.txt")
  Dim sw As IO.StreamWriter = IO.File.CreateText("Welcome.txt")
  Do Until sr.EndOfStream
    g = sr.ReadLine
    If (g <> "Aloha") Then
      sw.WriteLine(g)
    End If
  Loop
  sr.Close()
  sw.Close()
End Sub
```

12. Assume that the file Names.txt contains a list of names in alphabetical order. What is the effect of the Mystery function?

```
Private Sub btnFind_Click(...) Handles btnFind.Click
  MessageBox.Show(CStr(Mystery("Laura")))
End Sub

Function Mystery(ByVal name As String) As Boolean
  Dim sr As IO.StreamReader = IO.File.OpenText("Names.txt")
  Dim inputName As String
  Dim missingFlag As Boolean = False
  Do Until sr.EndOfStream
    inputName = sr.ReadLine
    If inputName = name Then
      Return True
    ElseIf inputName > name Then
      Return False
    End If
  Loop
  Return False
End Function
```

In Exercises 13 through 18, identify any errors. Assume that the contents of the file Greetings.txt is as shown in Fig. 8.6.

13.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim sw As IO.StreamWriter = IO.File.AppendText(Greetings.txt)
  sw.WriteLine("Guten Tag")
  sw.Close()
End Sub
```

14.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim term As String
  Dim sw As IO.StreamWriter = IO.File.CreateText("Greetings.txt")
  term = sw.Readline
  txtOutput.Text = term
  sw.Close()
End Sub
```

15.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  'Copy the contents of the file Greetings.txt into the file NewGreet.txt
  Dim name, greeting As String
  Dim sr As IO.StreamReader = IO.File.OpenText("Greetings.txt")
  name = "NewGreet.txt"
  Dim sw As IO.StreamWriter = IO.File.CreateText("name")
  Do Until sr.EndOfStream
    greeting = sr.ReadLine
    sw.WriteLine(greeting)
  Loop
  sr.Close()
  sw.Close()
End Sub
```

16.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim sw As IO.StreamReader = IO.File.CreateText("Greetings.txt")
  "Greetings.txt".Close()
End Sub
```

17.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Try
    Dim age As Integer
    age = CInt(InputBox("Enter your age."))
  Catch
    MessageBox.Show("Invalid age.")
  End Try
  MessageBox.Show("You are " & age & " years old")
End Sub
```

18.
```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim sw As IO.StreamWriter
  Try
    sw = IO.File.CreateFile("E:\Lakes.txt")
  Catch IO.IOException
    MessageBox.Show("Is there a CD in the E: drive?")
  End Try
```

```
    sw.Close()
End Sub
```

Exercises 19 through 25 are related and use the data in Table 8.2. The file created in Exercise 19 should be used in Exercises 20 through 25.

19. Write a program to create the text file Cowboy.txt containing the information in Table 8.2.

TABLE 8.2 Prices paid by cowboys for certain items in mid-1800s.

Colt Peacemaker	12.20
Holster	2.00
Levi Strauss jeans	1.35
Saddle	40.00
Stetson	10.00

20. Suppose the price of saddles is reduced by 20%. Use the file Cowboy.txt to create a file, Cowboy2.txt, containing the new price list.

21. Write a program to add the data Winchester Rifle, 20.50 to the end of the file Cowboy.txt.

22. Suppose an order is placed for 3 Colt Peacemakers, 2 Holsters, 10 pairs of Levi Strauss jeans, 1 Saddle, and 4 Stetsons. Write a program to perform the following tasks:

(a) Create the file Order.txt to hold the numbers 3, 2, 10, 1, and 4.
(b) Use the files Cowboy.txt and Order.txt to display a sales receipt with three columns giving the quantity, name, and cost for each item ordered.
(c) Compute the total cost of the items and display it at the end of the sales receipt.

23. Write a program to request an additional item and price from the user. Then create a file called Cowboy2.txt containing all the information in the file Cowboy.txt with the additional item (and price) inserted in its proper alphabetical sequence. Run the program for both of the following data items: Boots, 20.00 and Horse, 35.00.

24. Write a program to allow additional items and prices to be input by the user and added to the end of the file Cowboy.txt. Include a method to terminate the process.

25. Write a program using a StreamReader that displays the contents of the file Cowboy.txt in a DataGridView control.

26. Visual Basic cannot delete a file that is open. Attempting to do so generates an exception. Write a short program that uses structured exception handling to handle such an exception.

In Exercises 27 through 32, write a program to carry out the task without using arrays or LINQ. Assume that the file Numbers.txt contains a list of integers.

27. Display the number of numbers in the file Numbers.txt.

28. Display the largest number in the file Numbers.txt.

29. Display the smallest number in the file Numbers.txt.

30. Display the sum of the numbers in the file Numbers.txt.

31. Display the average of the numbers in the file Numbers.txt.

32. Display the last number in the file Numbers.txt.

Solutions to Practice Problems 8.2

1. *First:*

```
Dim sr As IO.StreamReader = IO.File.OpenText("USStates.txt")
Dim temp As String = ""
```

```
Do Until sr.EndOfStream
  temp = sr.ReadLine
Loop
MessageBox.Show(temp)
sr.Close()
```

Second:
```
Dim states() As String = IO.File.ReadAllLines("USStates.txt")
MessageBox.Show(states.Last)
```

Third:
```
Dim states() As String = IO.File.ReadAllLines("USStates.txt")
MessageBox.Show(states(states.Count - 1))
```

2. The file ABC.txt will be present in the program's *bin**Debug* folder. However, the file will be empty. In order for the string *abc* to be placed in the file, the statement **`sw.Close()`** must be executed.

8.3 XML

As we have seen, CSV files are quite useful. However, the expansion of the Internet mandated a standard text file format for transmitting data. The World Wide Web Consortium recommends a format called XML (eXtensible Markup Language) to be that standard.

Consider the CSV file USStates.txt. Let's look at a text file consisting of the first two lines of USStates.txt. That is, the two lines of the new file are

```
Delaware,DE,1954,759000
Pennsylvania,PA,44817,12296000
```

We are familiar with these records and know what each field represents. However, if we showed this file to someone else, they might not know how to interpret the information. One way to present it in great detail and in an organized format would be as follows:

```
U.S. States
  state
    name: Delaware
    abbreviation: DE
    area: 1954
    population: 749000
  state
    name: Pennsylvania
    abbreviation: PA
    area: 44817
    population: 1229600
```

VideoNote
XML

Format of XML Files

The XML format for the state data, which has the look and feel of the presentation above, is as follows:

```
<?xml version='1.0'?>
<!-- This file contains data on two of the 50 U.S. states.-->
<us_states>
  <state>
    <name>Delaware</name>
    <abbreviation>DE</abbreviation>
    <area>1954</area>
    <population>749000</population>
  </state>
```