File Input and Output

File I/O (Input and Output) is covered in chapter 9. We will only cover sequential text input and output, which is described in the first half of chapter 9.

Reading Data from a Text File

If you have stored data in a text file, using a program such as Notepad, you can also read it with VB.NET. Reading from a text file is useful to load the program with large amounts of data that would otherwise be tedious to type.

Data can be stored in files and accessed with a StreamReader object.

The steps to use the StreamReader object are as follows:

1. Execute a statement of the form

```
Dim readerVar As IO.StreamReader
```

A StreamReader is an object from the Input/Output class that can read a stream of characters coming from a disk or coming over the Internet. The Dim statement declares the variable readerVar to be of type StreamReader.

2. Execute a statement of the form

```
readerVar = IO.File.OpenText(filepath)
```

where filepath identifies the file to be read. This statement establishes a communications link between the computer and the disk drive for reading data from the disk. Data then can be input from the specified file and assigned to variables in the program. This assignment statement is said to "open the file for input."

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

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

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

```
strVar = readerVar.ReadLine
```

causes the program to look in the file for the next unread line of data and assign it to the variable strVar. The data can be assigned to a numeric variable if it is first converted to a numeric type with a statement such as

numVar = CDbl(readerVar.ReadLine)

Note: If all the data in a file have been read by ReadLine statements and another item is requested by a ReadLine statement, the item retrieved will have the value Nothing.

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

readerVar.Close()

As an example, a list of foods my 6 year old will eat is stored in the file C:\FOODS.TXT and it contains the following:

400
300
pizza
cheeseburgers
spaghetti
chicken nuggets
rice with chili
corn
apples
grapes

This says that we would like our program to be displayed with the upper left corner window at coordinate 400,300. Next is a list of foods he will eat. We would like a program that reads in this data, positions the window at 400,300, and then outputs all the foods he will eat into a listbox.

Inside the Form_Load event:

```
Dim foodFile As IO.StreamReader =
     IO.File.OpenText("C:\foods.txt")
Dim x, y As Integer
Dim s As String

s = foodFile.ReadLine()         ' Read left X coordinate
x = CInt(s)
y = CInt(foodFile.ReadLine())   ' Directly read top Y coordinate

' Set the window position to x,y
Me.Top = y
Me.Left = x

' Read in all the foods
s = foodFile.ReadLine()          ' Pizza
ListBox1.Items.Add(s)            ' Add to listbox
s = foodFile.ReadLine()          ' cheeseburger
ListBox1.Items.Add(s)            ' Add to listbox
s = foodFile.ReadLine()          ' spaghetti
```

```
ListBox1.Items.Add(s)            ' Add to listbox
s = foodFile.ReadLine()          ' chicken nuggets
ListBox1.Items.Add(s)            ' Add to listbox
s = foodFile.ReadLine()          ' rice with chili
ListBox1.Items.Add(s)            ' Add to listbox
s = foodFile.ReadLine()          ' corn
ListBox1.Items.Add(s)            ' Add to listbox
s = foodFile.ReadLine()          ' applets
ListBox1.Items.Add(s)            ' Add to listbox
s = foodFile.ReadLine()          ' grapes
ListBox1.Items.Add(s)            ' Add to listbox

foodFile.Close()
```

The output is:

Note that there is a lot of repeat code. We repeat the same two lines of code to read in the name of each item and add it to the listbox. We could improve this code by using a loop that continued until we reached the end of the file. The function:

filevar.Peek

returns -1 when we have reached the end of the file, and the ASCII code of the next character otherwise. Here is the simpler code:

```
Dim foodFile As IO.StreamReader =
          IO.File.OpenText("C:\foods.txt")
Dim x, y As Integer
Dim s As String

s = foodFile.ReadLine()        ' Read left X coordinate
x = CInt(s)
y = CInt(foodFile.ReadLine()) ' Directly read top Y coordinate

' Set the window position to x,y
Me.Top = y
Me.Left = x

' Read in all the foods
Do While (foodFile.Peek <> -1)
    s = foodFile.ReadLine()          ' Read a food
    ListBox1.Items.Add(s)            ' Add it to the listbox
Loop
foodFile.Close()
```

Here is another example that reads in words from a dictionary file to solve this word puzzle: "Name a common word, besides tremendous, stupendous and horrendous, that ends in dous."

We can solve this problem by loading up a file of words and checking each one to see if:

1) It is more than 4 letters long
2) The word contains “dous” at the end

Assume we have a file of English words located at C:\WORDS.TXT

```
Dim wordFile As IO.StreamReader = IO.File.OpenText("C:\WORDS.TXT")
Dim s As String

Do While (wordFile.Peek <> -1)
    s = wordFile.ReadLine()
    If s.Length > 4 Then
        If s.EndsWith("dous") Then
            Console.WriteLine(s)
        End If
    End If
Loop
wordFile.Close()                    ' Close the file
```

Writing To Sequential Text Files

Here we'll cover just the very basics of how to write and create a text file from your program.

Creating a text file is a lot like opening a file for reading, except we open it for creation instead. The steps to create a new text file and write data to it are:

1. Create an IO.StreamWriter object:

   Dim swriter As IO.StreamWriter = IO.File.CreateText(pathOnDisk)

   This will create a blank file with the given pathname. If the file already exists, it will be destroyed! (There is a separate function, IO.File.AppendText, that will open a file but append to the end of it instead of destroying the file). Note the similarities to opening a file, which was IO.File.OpenText(pathOnDisk)

   If we don't specify a full path then by default the file is placed in the current working directory (the bin directory of the project, if running from visual studio)

2. To place data in the file, use WriteLine, as we have used to write data to the console, except precede it by the Stream Writer variable:

   swriter.WriteLine(data)

3. When you are done recording data to the file, close it:

   swriter.Close()

The close statement breaks the link with the file on disk and frees up space in memory. Note that in most cases output will not actually be stored into the file until the Close statement is executed, so if you are checking the file contents to see what it is in, make sure the file is closed before you look at it.

Here is an example that would "hello there" and the number 42 to disk:

```
Dim sw As IO.StreamWriter = IO.File.CreateText("c:\test.txt")
sw.WriteLine("hello there")
sw.WriteLine(40 + 2)
sw.Close()
```