Unit 7 File Handling

Structure:

7.1 Introduction

In the previous unit we discussed the attributes in VB .NET its role, and also to create the custom attributes. We also discussed the delegates that are important to hold the reference of the methods.

In this unit we will explore the different file handling techniques available in VB.NET. File is generally defined as a named collection of related bytes of data stored on a secondary device - example disk. When your application accesses a file, it must assume whether the bytes represent characters, data records, integers, strings, and so on. The file's access type will inform the application that what type of data that the application will be handling.

The file access type we use, depends on the kind of data that the file contains. Visual Basic provides three types of file access:

- Sequential, for reading and writing text files in continuous blocks.
- Random, for reading and writing text or binary files need not be continuous order.
- Binary, for reading and writing binary data.

This unit is also focusing on StreamReader and StreamWriter classes, and various other methods supported by these classes. We will also discuss the BinaryReader and BinaryWriter class which supports to read the binary data from the file. Finally this unit explores the details of file and directory classes, various methods that support the operation on files and directories like create, move, copy, delete etc.

Objectives:

After studying this unit, you will be able to:

- explain the role of FileStream in file handling
- discuss the StreamReader support for file reading
- list and explain the methods of StreamWriter class
- discuss to read and write binary data using BinaryReader and BinaryWriter
- list the methods that supports file and directory operations.

7.2 Introduction to FileStream

File can be defined as “a collection of data stored in a disk with a specific name and a directory path”. When a file is opened for reading or writing, it becomes a stream.

In earlier versions you would have studied about the concept of reading or writing a data from the file as a single character or as a string. In this unit we are going to discuss the concept of file streams. When a group of or a sequence of bytes passes through the communication path is called streams. Input stream and output stream are the two major file streams. As the name indicates the input stream is use to read a data or information from the file and the output stream is used to write the data or information in to the file.

Various file operation are supported by the *Sysem.IO* name space. It consist of class that supports operations like reading and writing data from and to the file, creating and deleting files and opening and closing files etc.

The following table 7.1 lists few commonly used *System.IO* name space with its descriptions.

Table 7.1: I/O classes in System.IO namespace.

I/O Class	Description
BinaryReader	Reads primitive data from a binary stream.
BinaryWriter	Writes primitive data in binary format.
BufferedStream	A temporary storage for a stream of bytes.
Directory	Helps in manipulating a directory structure.
DirectoryInfo	Used for performing operations on directories.
DriveInfo	Provides information for the drives.
File	Helps in manipulating files.
FileInfo	Used for performing operations on files.
FileStream	Used to read from and write to any location in a file.
MemoryStream	Used for random access to streamed data stored in memory.
Path	Performs operations on path information.
StreamReader	Used for reading characters from a byte stream.
StreamWriter	Is used for writing characters to a stream.
StringReader	Is used for reading from a string buffer.
StringWriter	Is used for writing into a string buffer.

The FileStream class that belongs to the *System.IO* namespace is used for the various operations like read, write and close the files. FileStream class is derived from abstract class called Stream. To use these stream objects, first we need to create the FileStream object to open an existing or create a new file. Below is the syntax to create a FileStream object

Dim <object_name> As FileStream = New FileStream(<file_name>, <FileMode Enumerator>, <FileAccess Enumerator>, <FileShare Enumerator>)

If you want to create an object say obj1 to read a file “file1.txt” the syntax will be

Dim obj1 FileStream=New FileStream(“file.txt”, FileMode.OpenOrCreate, FileAccess.ReadWrite)

Table 7.2 explains the FileMode, FileAccess and FileShare parameters that are used in the above syntax.

Table 7.2: File reading method parameter with its description

Parameter	Description
FileMode	The **FileMode** enumerator defines various methods for opening files. The members of the FileMode enumerator are: • **Append**: It opens an existing file and puts cursor at the end of file, or creates the file, if the file does not exist.
	• **Create**: It creates a new file. • **CreateNew**: It specifies to the operating system, that it should create a new file. • **Open**: It opens an existing file. • **OpenOrCreate**: It specifies to the operating system that it should open a file if it exists, otherwise it should create a new file. • **Truncate**: It opens an existing file and truncates its size to zero bytes.
FileAccess	**FileAccess** enumerators have members: **Read**, **ReadWrite** and **Write**.
FileShare	**FileShare** enumerators have the following members: • **Inheritable**: It allows a file handle to pass inheritance to the child processes • **None**: It declines sharing of the current file • **Read**: It allows opening the file for reading • **ReadWrite**: It allows opening the file for reading and writing • **Write**: It allows opening the file for writing

The *FileStream* class gives you access to files. You start working with a file on disk by opening it or creating it. You can use the members of the *FileAccess, FileMode, and FileShare* enumerations with the constructors of the FileStream class to determine how the file is created, opened, and shared. In addition, the FileStream class can open a file in one of two modes, either synchronously or asynchronously, which can have significant performance differences in different circumstances. *FileStream* defaults to opening files synchronously, but also has a constructor to open files asynchronously. After you have opened or created a file, you can pass its *FileStream* object to the BinaryReader, BinaryWriter, StreamReader, and StreamWriter classes to actually work with the data in the file, which we will discuss in later sections.

FileStream Seek method can be used to move to various locations in a file. This is called moving the read/write position or the read/write pointer. This allows you to break a file up into *records*, each of the same length. For example, if you're keeping track of 2,000 employees, you can create 2,000 records in a file, each with data on the corresponding employee.

Because you know the length of each record, it's easy to move to the beginning of a specific record and read it or overwrite it with new data. This

record-based process, where you can move around in a file and select the data you want, is called *random access*. The other form of file access, where you just read or write data to a file one item after the other is called *sequential access*. The only difference between these types of access from our point of view is that for random access, you use the *Seek* method. The following example helps you understand the use of file stream.

```
Imports System.IO Imports System.Text
Public Class Form1
  Private Sub Button1_Click(ByVal sender As System.Object, _ ByVal e As
      System.EventArgs) Handles Button1.Click
    Try
      Dim file1 As System.IO.FileStream Dim byteData() As
      Byte
      byteData = Encoding.ASCII.GetBytes("FileStream Test1") file1 = New
      FileStream("streamtest.txt", FileMode.Append) file1.Write(byteData, 0,
      byteData.Length)
      file1.Close()
    Catch ex As IOException MsgBox(ex.ToString)
    End Try End Sub
End Class
```

You can observe from the above program that the file1 is declared as a file stream object. It is been set to the file called “streamtest.txt” opened as a append mode. Stream of bytes will be written in to this file using the write () method. In the above example the 3 parameters of write methods are buffer, offset and count respectively.

- buffer (byteData) - Array of bytes copied from buffer to current stream
- offset (0) - position from where to start copying bytes to the current stream
- count (byteData.Length) – number of bytes to be written on the current stream

After completion of writing process the file is been closed with close () function.

7.3 StreamReader and StreamWriter Classes

If you want to work with text data files, to do the operations of read, write and store text data in the file it is recommend to use either StreamWriter or StreamReader classes. Basically the TextWriter is the base class for the StreamWriter class and the StreamReader class is derived from the TextReader base class. Both these classes are used to read and write the stream of bytes from and to the file.

StreamReader class

As already we discussed it is extracted from the abstract base class TextReader. Table 7.3; list the methods that are commonly used from StreamReader class with its purpose.

Table 7.3: StreamReader class methods

Method Name	Description
lose()	Closes the StreamReader object and release the resource availed.
Peek()	Returns the next available character
Read()	Reads the next character from the input stream and advance character position by one.

```
Imports System.IO Module fileProg
```

```
Sub Main () Try
    Using sr1 As StreamReader = New StreamReader ("Streamread.txt")
      Dim line1 As String line1 = sr1.ReadLine ()
      While (line1 <> Nothing)
        Console.WriteLine(line1) line1 = sr1.ReadLine
        ()
      End While End Using
  Catch e1 As Exception
    Console.WriteLine("The file could not be read:")
    Console.WriteLine(e1.Message)
  End Try Console.ReadKey()
 End Sub
End Module
```

The above given program is to explain the StreamReader class. Here we have created an instance sr1 for the class StreamReader to read the data from the file "Streamread.txt". Variable line1 is declared as string variable. This program starts reading the data from the file line by line and display on the screen, until it reaches the end of the file. Here *using* statement is used to close the file. In this program we also have the code for error handling. The catch block catches the error thrown from the try block and prints the error message if any on the screen. In the above program when error encountered while opening the file the error will be thrown from the Try block.

StreamWriter class

As already we discussed the StreamWriter class is inherited from the abstract TextWriter base class. This writes the series of bytes of information to the file. Following table 7.4 lists the methods that are commonly used with this class.

Table: 7.4 StreamWriter class

Method Name	Description
Close()	Close the current StreamWriter object
Flush()	Clears all buffers for the current writer
Write(value as Boolean)	Writes the text representation of the boolean value to the stream
Write(value as Char)	Writes the character to the stream
Write(Value as Decimal)	Writes the text representation of decimal value
Write(Value as String)	Writes String to the Stream
WriteLine()	Writes a line terminator to the text string

Now we will discuss an example that writes a stream of data into the file using StreamWriter class.

```
Imports System.IO Module fileProg1
  Sub Main()
    Dim namelist As String() = New String() {"C Language", _ "OOPS Concepts", "Java
      Programming", "System Software"}
    Dim s1 As String
    Using sw1 As StreamWriter = New StreamWriter("namelist.txt") For Each s1 In
      namelist
        Sw1.WriteLine (s1) Next s1
    End Using

    Dim line1 As String
    Using sr1 As StreamReader = New StreamReader("namelist.txt") Line1 =
      sr1.ReadLine()
      While (line1 <> Nothing)
        Console.WriteLine(line1) Line1 =
        sr1.ReadLine()
      End While End Using
    Console.ReadKey() End Sub
End Module
```

The above program creates an instance for the class StreamWrite and writes the data using WriteLine () function. The same file is reopened and displays the content that was written earlier.

The output of the program will be
C Language
OOPS Concepts
Java Programming
System Software

Now We will see one more example that uses classes named StreamWriterReader; this example will write text to a file, “file.txt”, and then
read that text back, and display the same in the text box. We are going to start this program by importing the System.IO namespace, and by creating file.txt and connecting a FileStream object to it. Next, we are going to create StreamWriter object and use various methods to move around and write text to the file.

```
Public Class Form1
  Inherits System.Windows.Forms.Form
  Private Sub Button1_Click(ByVal sender As System.Object, _ ByVal e As
    System.EventArgs) Handles Button1.Click
    Dim fs As New System.IO.FileStream("file.txt", FileMode.Create,
FileAccess.Write)
    Dim w As New StreamWriter(fs) w.BaseStream.Seek(0,
    SeekOrigin.End) w.WriteLine("Here is the file's text.")
    w.Write("Here is more file text." & ControlChars.CrLf) w.WriteLine("And
    that's about it.")
    w.Flush()
```

```
        w.Close()
        fs = New System.IO.FileStream("file.txt", FileMode.Open, _ FileAccess.Read)
        Dim r As New StreamReader(fs) r.BaseStream.Seek(0,
        SeekOrigin.Begin) While r.Peek() > -1
            TextBox1.Text &= r.ReadLine() & ControlChars.CrLf End While
        r.Close() End Sub
    End Class
```

In this case, we use the Seek method to move the pointer to the beginning of the file. This is not applicable in newly created or opened files because when you open a file, you start at the beginning of the file since the file is empty. Then write a line of text to the file with the WriteLine method, and then write some text to the file with the *Write* method. The WriteLine method is the same as Write, except it adds a carriage-return/linefeed pair at the end of the text it writes.

We are using Flush method because the file handling is buffered in Visual Basic, nothing will be written to disk until the buffer is flushed. This happens automatically when the buffer is full or when you close a file or when the associated stream object goes out of scope. But you also can use the Flush method to explicitly flush data to the disk. Flush works automatically, in this program but we are using Flush to know the role of this method. Finally, we are closing the file with the Close method; this closes the file on disk, which finishes our work with the file and makes it available to other programs:

You can see the results of this code in Figure 7.1, depicting the creation of new file, filled it with text, and read that text back in.

Fig. 7.1: Writing and reading a text file.

7.4 BinaryReader and BinaryWriter Classes

BinaryReader and BinaryWriter classes are used to read and write the content in the binary files.

BinaryReader class

As the name indicates this class is used to read a binary data from the file. Here FileStream object will be passed through the constructor to create the

BinaryReader object. Following table 7.5 lists the methods of BinaryReader class.

Table 7.5: BinaryReader methods

Method Name	Description
Close()	Closes the BinaryReader object
Read()	Reads the character from the stream
ReadBoolean()	Reads boolean value and advance the pointer by one byte
ReadByte()	Reads next byte from current stream and advance the pointer by one byte
ReadChar()	Reads next character from current stream and advance the pointer by one byte
ReadDouble()	Reads 8 byte floating point value from current stream and advance the pointer by one byte
ReadInt32()	Reads 4 byte signed integer value from current stream and advance the pointer by one byte
ReadString()	Reads String from current stream of fixed length encoded as seven bits at a time.

BinaryReader Object works at lower level of Streams. BinaryReader is used for read premitive types as binary values in a specific encoding stream. Binaryreader Object works with Stream Objects that provide access to the underlying bytes. For creating a BinaryReader Object, you have to first create a FileStream Object and then pass BinaryReader to the constructor method.

```
Imports System.IO
Public Class Form1
Private Sub Button1_Click(ByVal sender As System.Object, _ ByVal e As
System.EventArgs) Handles Button1.Click
    Dim readStream As FileStream Dim msg As String
    Try
readStream = New FileStream("c:\testBinary.dat", FileMode.Open) Dim readBinary As
      New BinaryReader(readStream)
       msg = readBinary.ReadString() MsgBox(msg)
       readStream.Close()
```

```
        Catch ex As Exception
            MsgBox(ex.ToString)
        End Try
    End Sub
End Class
```

BinaryWriter class

BinaryWriter creates a binary file. This file contains a certain layout of bytes. With BinaryWriter, we write individual bytes, integers or strings to a file location. This makes possible the creation of files in certain formats. The BinaryWriter Object works at lower level of Streams. BinaryWriter is used for write primitive types as binary values in a specific encoding stream. BinaryWriter Object works with Stream Objects that provide access to the underlying bytes. For creating a BinaryWriter Object, you have to first create a FileStream Object and then pass BinaryWriter to the constructor method.

The main advantage of Binary information is that it is not easily human readable and stores files as Binary format is the best practice of space utilization. Table 7.6 lists the methods of BinaryWriter class.

Table 7.6: Methods of BinaryWriter class

Method Name	Description
Close	closes the BinaryReader object
Read	Reads the characters from the current stream and advances pointer position
ReadBoolean	Reads a Boolean value from the current stream and advances pointer position
ReadByte	Reads the next byte from the current stream and advances pointer position
ReadBytes (count As Integer)	Reads the specified number of bytes from the current stream
ReadChar	Reads the next character from the current stream advances the current position
ReadChars (count As Integer)	Reads the specified number of characters from the current stream
ReadDouble	Reads an 8-byte floating point value from the current stream advances the current position
ReadInt32	Reads a 4-byte signed integer from the current stream
ReadString	Reads a string from the current stream. The string is prefixed with the length

Following program explain how BinaryWriter can be used to write the data in the file through binary stream.

```
Imports System.IO
Public Class Form1
Private Sub Button1_Click(ByVal sender As System.Object, _ ByVal e As
```

```
System.EventArgs) Handles Button1.Click
    Dim writeStream As FileStream Try
writeStream = New FileStream("c:\testBinary.dat", FileMode.Create) Dim writeBinay As
        New BinaryWriter(writeStream) writeBinay.Write("This is a test for BinaryWriter
        !") writeBinay.Close()
    Catch ex As Exception MsgBox(ex.ToString)
    End Try End Sub
End Class
```

7.5 File and Directory Classes

Now we are going to discuss the other important classes apart from what we discussed above, namely File and Directory.

File class

The File class supports in various file operations like open, move and copy a file to the different location and so on. Similarly the Directory class supports in creating, renaming and deleting the directories. Here the methods that we are going to use are class methods, so you are free from creating an object to do these operations.

Following are the list of syntaxes that supports various file operations.

File Creation: First we will start with the method that does the file creation.

```
File.Create (Path and file name)
```

Here the parameter requires for the file create method is name of the file along with the path where you wanted to create that file.

Check for File Existence: You must have observed in the above discussed examples, that before we create an object for the file we check for the availability of the particular file.

```
File.Exists (Path and file name) as Boolean
```

This particular file check method for its existence it require single parameter, that indicates the path of the file and returns the status of the file with the Boolean expression. If it returns True means the file is existing and false state indicates the absence of the file.

```
Imorts System.IO Public Class Form1
  Private Sub Button1_Click(ByVal sender As System.Object, _ ByVal e As
      System.EventArgs) Handles Button1.Click
    If File.Exists("c:\testFile.txt") Then MsgBox("File
        'testFile' Exist ")
    Else File.Create("c:\testFile.txt")
      MsgBox("File 'testFile' created ") End If
  End Sub End Class
```

The above program is to illustrate the two file operations such as File exist and create. Program given above first checks for he file existence if it exists already it will display the message “File already exits” otherwise using create function it creates File object. Next we will see how to make a copy of an existing file

Copy: *Copy (source filename, Destination filename)*

Here this copy method requires two parameters, first is the source file that you wanted to move. Second is the destination to where you wanted to move the file.

Delete: This method is used to delete an existing file, below is the syntax for delete operation

Delete (Path and file name)

This operation requires single parameter, the name and path of the file that you wanted to delete.

```
Imports System.IO
    Public Class Form1
      Private Sub Button1_Click(ByVal sender As System.Object, _ ByVal e As
          System.EventArgs) Handles Button1.Click
        If Not File.Exists("c:\testFile.txt") Then MsgBox("FIle not
          exist ")
        Else
          File.Copy("c:\testFile.txt", "c:\testDir\testFile.txt") MsgBox("File
          Copied ") File.Delete("c:\testFile.txt")
          MsgBox("file deleted ") End If
      End Sub End Class
```

The above program is to exhibit the copy and delete operations. It checks for the given file availability, if it exits it will be copied to the different location and it gets deleted or it prints the error message that the specified file does not exist.

Directory class

Similar to File class in directory class also, you can do directory operations like create, move or delete directories. We can use or call the methods directly with the directory class. Now we will discuss various operations of directories.

Create: We can use CreateDirectory method from the Directory class the syntax is as follows

Directory.CreateDirectory (DirPath)

This method requires one parameter to execute the function, Path of the directory to specify where the new directory needs to be created and located.

Directory Existence: Generally we check for the availability of the particular directory before we create any directory.

Directory.Exists (Path and directory name) as Boolean

This particular directory check method requires only one parameter that indicates the path of the directory and returns the status of the directory with the Boolean expression. If it returns True means the directory is existing, and the false state indicates the absence of the directory.

Move: This method will be useful whenever you wanted to move the contents of directory from one location to other.

Move (source DirName, Destination DirName)

Here the parameters require for this method is two, the source directory from where you wanted move and the destination indicates the new location to move.

Delete: This method is useful where you want to delete a directory

Delete (Path and directory name)

One parameter is required for this method which is the path of the directory that you wanted to delete.

The below program is basically to exhibit the various directory methods that we discussed earlier. Here first we are checking for the availability of the testDir1 directory if it already exists we are going to display the message

"Directory Exist". Otherwise using CreateDirectory method the directory will be created. Next using same method testDir2 is created inside the testDir1 directory. Now, using move function testDir2 is been relocated then the testDir1 has been deleted.

```
Imports System.IO

Public Class Form1
    Private Sub Button1_Click(ByVal sender As System.Object,_ ByVal e As
        System.EventArgs) Handles Button1.Click
      If Directory.Exists("c:\testDir1") Then MsgBox("Directory
        'testDir' Exist ")
      Else Directory.CreateDirectory("c:\testDir1")
        MsgBox("testDir1 created ! ")
          Directory.CreateDirectory("c:\testDir1\testDir2")
        MsgBox("testDir2 created ! ") Directory.Move("c:\testDir1\testDir2",
        "c:\testDir") MsgBox("testDir2 moved ")
        Directory.Delete("c:\testDir1")
        MsgBox("testDir1 deleted ") End If
    End Sub End Class
```

We will see here now one more example that Open File dialog to select a file that gets copied to the new directory, using the File class's Copy method, here the pathname of the filename returned by the Open File dialog using handled and using String class's SubString method.

```
Imports System.IO

    Public Class Form1
      Inherits System.Windows.Forms.Form
        Private Sub Button1_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles Button1.Click
        Try
          Directory.CreateDirectory(TextBox1.Text)
        Catch
```

```
            MsgBox("Could not create directory.")
            Exit Sub
        End Try
        MsgBox("Directory created.")
    End Sub


    Private Sub Button2_Click(ByVal sender As System.Object, _ ByVal e As
        System.EventArgs) Handles Button2.Click
        Try
            If   OpenFileDialog1.ShowDialog   <>   DialogResult.Cancel
Then
          File.Copy(OpenFileDialog1.FileName, TextBox1.Text & "\" & _
                OpenFileDialog1.FileName.Substring(_
                OpenFileDialog1.FileName.LastIndexOf("\")))
            End If Catch
            MsgBox("Could not copy file.") Exit Sub
        End Try
        MsgBox("File copied.") End Sub
End Class
```

Here the Form has inherited the System window to generate the Windows Form Designer code window. Using createDirectory method it tries to create the method taking the parameter value from the TextBox control (path and name of the Directory given by the user). The code is written inside the try block if it encounters any error will throw to catch block and the error message will be displayed. These business logics are written in the Button1 click event. You can observe one more Button control is available named Button2 where we are going to write the code to copy a file to the new directory. Here also we have try catch block to display the error message if the program encounters any error. When you run the above program the output appears as shown in figure 7.2.

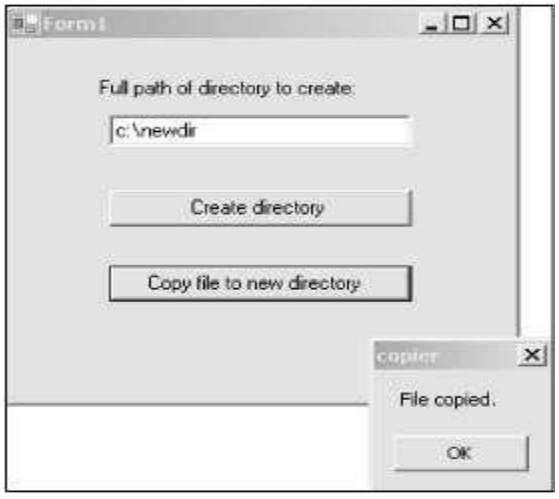

Fig. 7.2: creating a directory and copying a file to it

7.6 Summary

- File is a collection of data stored in a disk with a specific name and a directory path.
- Group of data channeled through the communication path is called stream.
- Input and the output streams are considered as the two major streams in FileStream.
- FileStream is base class belongs to the System.IO namespace used for various file operations.
- There are 5 different file opening modes are supported in VB .NET.
- Seek method helps to browse through the file.
- BinaryReader and BinaryWriter classes are used to read and write the content in the binary files.
- The File class supports in various file operations like open, move and copy a file to the different location.
- Using directory class, you can do directory operations like create, move or delete directories.

7.7 Questions and Exercises

1. What is FileStream? Discuss the parameters required to create a FileStream object.
2. Discuss the various file modes used to open a file with appropriate example.
3. Explain the StreamReader class with appropriate example.
4. List and explain the StreamWriter class methods.
5. Explain brief about the BinaryReader and BinaryWriter classes.
6 What are File and Directory classes discuss with example

7.8 Suggested Readings:

- http://www.yaldex.com/vb-net-tutorial- /library.books24x7.com/ book/ id_ 5526 / viewer.asp@ bookid= 5526& chunkid=0326962437.htm
- http://www.tutorialspoint.com/vb.net/vb.net_file_handling.htm
- http://vb.net-informations.com/files/vb.net_binaryReader.htm