Files and Data Streams

System.IO – The System.IO namespace provides a variety of classes for working with files and for managing directories, files, and paths.

Classes for managing directories, files and paths –

* The classes for managing directories, files, and paths are stored in the System.IO namespace.
* To use the classes in the System.IO namespace, you should include an Imports statement. Otherwise, you have to qualify the references to its classes with System.IO.
* All of the methods of the Directory, File and Path classes are shared methods.

System.IO classes used to work with drives and directories –

Class	Description
Directory	Used to create, edit, delete or get information on directories.
File	Used to create, edit, delete or get information on files.
Path	Used to get path information from a variety of platforms.

Common methods of Directory class -

Method	Description
Exists(path)	Returns a Boolean value indicating whether a directory exists.
CreateDirectory(path)	Creates the directories in a specified path.
Delete(path)	Deletes the directory at the specified path. The directory must be empty.
Delete(path, recursive)	Deletes the directory at the specified path. If True is specified for the recursive argument, any subdirectories and files in the directory are deleted. If False is specified, the directory must be empty.

Common methods of File class -

Method	Description
Exists(path)	Returns a Boolean value indicating whether a file exists.
Delete(path)	Deletes a file.
Copy(source, dst)	Copies a file from a source path to a destination path.
Move(source, dst)	Moves a file from a source path to a destination path.

Ex.1
```
Dim dir As String = "C:\VB 2010 Files\"
If Not Directory.Exists(dir) Then
    Directory.CreateDirectory(dir)
End If
```

Ex.2
```
Dim path As string = dir & "Products.txt"
If File.Exists(path) Then
    File.Delete(path)
End If
```

How files and streams work

* An input file is a file that is read by a program; an output file is a file that is written by a program. Input and output operations are often referred to as I/O operations or file I/O.
* A stream is the flow of data from one location to another. To write data, you use an output stream. To read data, you use an input stream. A single stream can also be used for both input and output.
* To read and write text files, you use text streams. To read and write binary files, you use binary streams.

Two types of files

Type	Description
Text	A file that contains text (string) characters. The fields in each record are typically delimited by special characters like tab or pipe characters, and records are typically delimited by new line character.
Binary	A file that can contain a variety of data types.

Two types of streams

Stream	Description
Text	Used to transfer text data.
Binary	Used to transfer binary data.

System.IO classes used to work with files and streams

Class	Description
FileStream	Provides access to input and output files.
StreamReader	Used to read a stream of characters.
StreamWriter	Used to write a stream of characters.
BinaryReader	Used to read a stream of binary data.
BinaryWriter	Used to write a stream of binary data.

How to use the FileStream class

Syntax for creating a FileStream object

```
New FileStream(path, mode [, access [, share]])
```

Members in the FileMode enumeration

Member	Description
Append	Opens the file if it exists and seeks to the end of the file. If the file doesn't exist, it's created. This member can only be used with Write file access.
Create	Creates a new file. If the file already exists, it's overwritten.
CreateNew	Creates a new file. If the file already exists, an exception is thrown.
Open	Opens an exesting file. If the file doesn't exist, an exception is thrown.
OpenOrCreate	Opens a file if it exists, or creates a new file if it doesn't exist.
Truncate	Opens an existing file and truncates it so its size is zero bytes.

Members in the FileAccess enumeration

Member	Description
Read	Data can be read from the file but not written to it.
ReadWrite	Data can be read from and written to the file. This is the dofault.
Write	Data can be written to the file but not read from it.

Members in the FileShare enumeration

Member	Description
None	The file cannot be opened by other applications.
Read	Allows other applications to open the file for reading only. This is the default.
ReadWrite	Allows other applications to open the file for both reading and writing.
Write	Allows other applications to open the file for writing only.

Common method of the FileStream class

Method	Description
Close()	closes the file stream and releases any resources associated with it.

Ex.1- Code to create a FileStream object for writing.

```
Dim path As String = "C:\VB 2010\Files\Products.txt"
Dim fs As New FileStream (path, FileMode.Create,
                          FileAccess.Write)
```

Ex.2- Code to create a new FileStream object for reading

```
Dim path As String = "C:\VB 2010\Files\Products.txt"
Dim fs As New FileStream (path, FileMode.Open,
                          FileAccess.Read)
```

How to use the exception classes for file I/O –

* To catch any I/O exception, you can use the IOException class.
* To catch specific I/O exceptions, you can use the exception classes that inherit the IOException class such as DirectoryNotFoundException, FileNotFoundException, EndOfStreamException.

The exception classes for file I/O

Class	Description
IOException	The base class for exceptions that are thrown during the processing of a stream, file or directory.
DirectoryNotFoundException	Occurs when part of a directory or file path can't be found.
FileNotFoundException	Occurs when a file can't be found.
EndOfStreamException	Occurs when an application attempts to read beyond the end of a stream.

Ex.
```
Dim dirPath As String = "C:\VB 2010\Files\"
Dim filePath As String = dirPath & "Products.txt"
Dim fs As FileStream
Try
    fs = New FileStream(filePath, FileMode.Open)
    ' # code that uses the file stream
    ' # to read and write data from the file
```

```
Catch ex As FileNotFoundException
    MessageBox.Show(filePath & " not found.", "File
        Not found")
Catch ex As DirectoryNotFoundException
    MessageBox.Show(dirPath & " not found.", "Directory Not Found")
Catch ex As IOException
    MessageBox.Show(ex.Message, "IOException")
Finally
    If fs IsNot Nothing Then
        fs.Close()
    End If
End Try
```

How to work with binary files

To read and write data in a binary file, you use the BinaryReader and BinaryWriter classes.

How to write a binary file –

First of all we create a BinaryWriter object using the following syntax:

```
New BinaryWriter(stream)
```

To do that we must supply a FileStream object as the argument for the constructor of the BinaryWriter class. This links the stream to the

BinaryWriter object, so it can be used to write to the file.
After creating BinaryWriter object, we can use its Write method to write all types of data. This method begins by figuring out what type of data has been passed to it. Then it writes that type of data to the file.

Common methods of the BinaryWriter class

Method	Description
Write(data)	Writes the specified data to the output stream.
Close()	closes the BinaryWriter object and the associated FileStream object.

Ex.1
```
Dim binaryOut As New BinaryWriter(
    New FileStream(path, FileMode.Create, FileAccess.Write))
For Each product As Product In products
    binaryOut.Write(product.Code)
    binaryOut.Write(product.Description)
    binaryOut.Write(product.Price)
Next
binaryOut.Close()
```

In the above code, a binary writer is created for a file stream that specifies a file that has write-only access. Since the mode argument has been set to

Create, this will overwrite the file if it exists, and it will create the file if it doesn't exist. Then, a for each loop is used to write the elements in a List() collection named products to the file. Since each element in the List() collection is an object of the Product class, each property of the Product object is written to the file seperately using the Write method. After all of the elements in the List() collection have been written to the file, the Close method is used to close both the BinaryWriter and the FileStream objects.

Ex.2

```
Imports System.IO
Module Module1
    Sub Main()
        Dim arr() As Int32 = {2, 6, 9, 11, 45, 87, 777, 23,
                              266, 44, 83, 94}
        Using writer As BinaryWriter = New
            BinaryWriter(File.Open("file.bin", fileMode.Create))
            For Each value As Int32 In arr
                writer.Write(value)
            Next
        End Using
    End Sub
End Module
```

How to read a binary file -

Like the BinaryWriter class, the argument that we pass to the BinaryReader is the name of the FileStream object that connects the stream to a file.

In a binary file, there's no termination character to indicate where one record ends and another begins. Because of that, we can't read an entire record at once. Instead, you have to read one character or one field at a time. To do that, you use the Read methods of the BinaryReader class. You must use the appropriate method for the data type of the field that you want to read.

Before you read the next ~~next~~ character or field, you want to be sure that you aren't at the end of the file. To do that you use the PeekChar method. Then, if there's at least one more character to be read, this method returns that character without advancing the cursor to the next position in the file. If there isn't another character, the PeekRead method returns a value of -1. Then you can use the Close method to close the binary reader and the associated file stream.

Syntax for creating a BinaryReader object

```
New BinaryReader(stream)
```

Common methods of the BinaryReader class

Method	Description
PeekChar()	Returns the next available character in the input stream without advancing to the next position. If no more characters are available, this method returns -1.
Read()	Returns the next available character from the input stream and advances to the next position in the file.
ReadBoolean()	Returns a Boolean value from the input stream and advances the current position of the stream by one byte.
ReadByte()	Returns a byte from the input stream and advances the current position of the stream accordingly.
ReadChar()	Returns a character from the input stream and advances the current position of the stream accordingly.
ReadDecimal()	Returns a decimal value from the input stream and advances the current position of the stream by 16 bytes.
ReadInt32()	Returns a 4-byte signed integer from the input stream and advances the current position of the stream by 4 bytes.
ReadString()	Returns a string from the input stream and advances the current position of

the stream by the number of characters in the string.

Close() Closes the BinaryReader object and the associated FileStream object.

Ex.1
```
Dim binaryIn As New BinaryReader(New
    FileStream(path, FileMode.OpenOrCreate, FileAccess.Read))
Dim products As New List(Of Product)
Do While binaryIn.PeekChar <> -1
    Dim product As New Product
    product.Code = binaryIn.ReadString
    product.Description = binaryIn.ReadString
    product.Price = binaryIn.ReadDecimal
    products.Add(product)
Loop
binaryIn.Close()
```

In the above code, a FileStream object is created for a file that will have read-only access. Since the mode argument for the file stream specifies OpenOrCreate, this opens an existing file if one exists or creates a new file that's empty and opens it. Then, a new BinaryReader object is created for that file stream. Finally, the Do loop that follows is executed until the PeekChar method returns a value of -1, which means the end of the file has been reached.

Within the Do loop, the three fields in each record are read and assigned to the properties of the Product object. Because the first two fields in each record contain string data, the ReadString method is used to retrieve its contents. Then, the Product object is added to the List() collection. When the Do loop ends, the Close method of the BinaryReader object is used to close both the BinaryReader and the FileStream objects.

Ex.2

```
Module Module1
    Sub Main()
        Using reader As New BinaryReader(File.Open(
            "file.bin", FileMode.Open))
        'Dim pos As Integer = 0
        'Dim length As Integer = reader.BaseStream.Length
        While pos < length reader.PeekChar <> -1
            Dim value As Integer = reader.ReadInt32
            Console.WriteLine(value)
            'pos += 4
        End While
        End Using
    End Sub
End Module
```

How to work with text files

To read and write characters in a text files, you use the StreamReader and StreamWriter classes. When working with text files, you often need to use string, date and numeric data.

How to write text file —

Create a StreamWriter object which takes a FileStream object as its argument as follows:

```
New StreamWriter (stream)
```

After creating StreamWriter object we write any data to a text file by using the Write, and WriteLine methods. When we use WriteLine method, a line terminator is automatically added. Typically, a line terminator is used to end each record. The fields in a record are typically separated by special characters such as pipe character, by adding those characters throughout code.

Both the Write and WriteLine methods of StreamWriter class are overloaded to accept any type of data. As a result if we pass a non-string data type to

* Dim sw As New StreamWriter (New FileStream(path, FileMode.Create, FileAccess.Write))

either of these methods, the method converts the data type to a string that represents the data type and then it writes that string to the stream. To do that, these methods automatically call the ToString method of the data type.

Common methods of the StreamWriter class-

Method	Description
Write(data)	Writes the data to the output stream
WriteLine(data)	Writes the data to the output stream and appends a line terminator (usually a carriage return and a line feed).
Close()	Closes the StreamWriter object and the associated FileStream object.

Ex.-
```
'Get the directories currently in the C drive
Dim cDirs As DirectoryInfo() = New
    DirectoryInfo("C:\").GetDirectories()
* Using sw As StreamWriter = New StreamWriter
    ("CDriveDirs.txt")
    For Each Dir As DirectoryInfo In cDirs
        sw.WriteLine(Dir.Name)
    Next
End Using
```

How to read a text file —

Create a StreamReader object, you can use a FileStream object as the Argument.

Basic syntax for creating StreamReader object—

```
New StreamReader(stream)
```

Common methods of StreamReader class —

Method	Description
Peek()	Returns the next available character in the input stream without advancing to the next position. If no more characters are available, this method returns -1.
Read()	Reads the next character from the input stream.
ReadLine()	Reads the next line of characters from the input stream and returns it as a string
ReadToEnd()	Reads the data from the current position in the input stream to the end of the stream and returns it as a string. This is typically used to read the contents of an entire file.
Close()	Closes both the StreamReader and the associated FileStream object.

Ex. -

```
Dim line As String = ""
Using sr As StreamReader = New StreamReader(
        "CDriveDirs.txt"
    Do
        line = sr.ReadLine()
        Console.WriteLine(line)
    Loop Until line Is Nothing
End Using
```