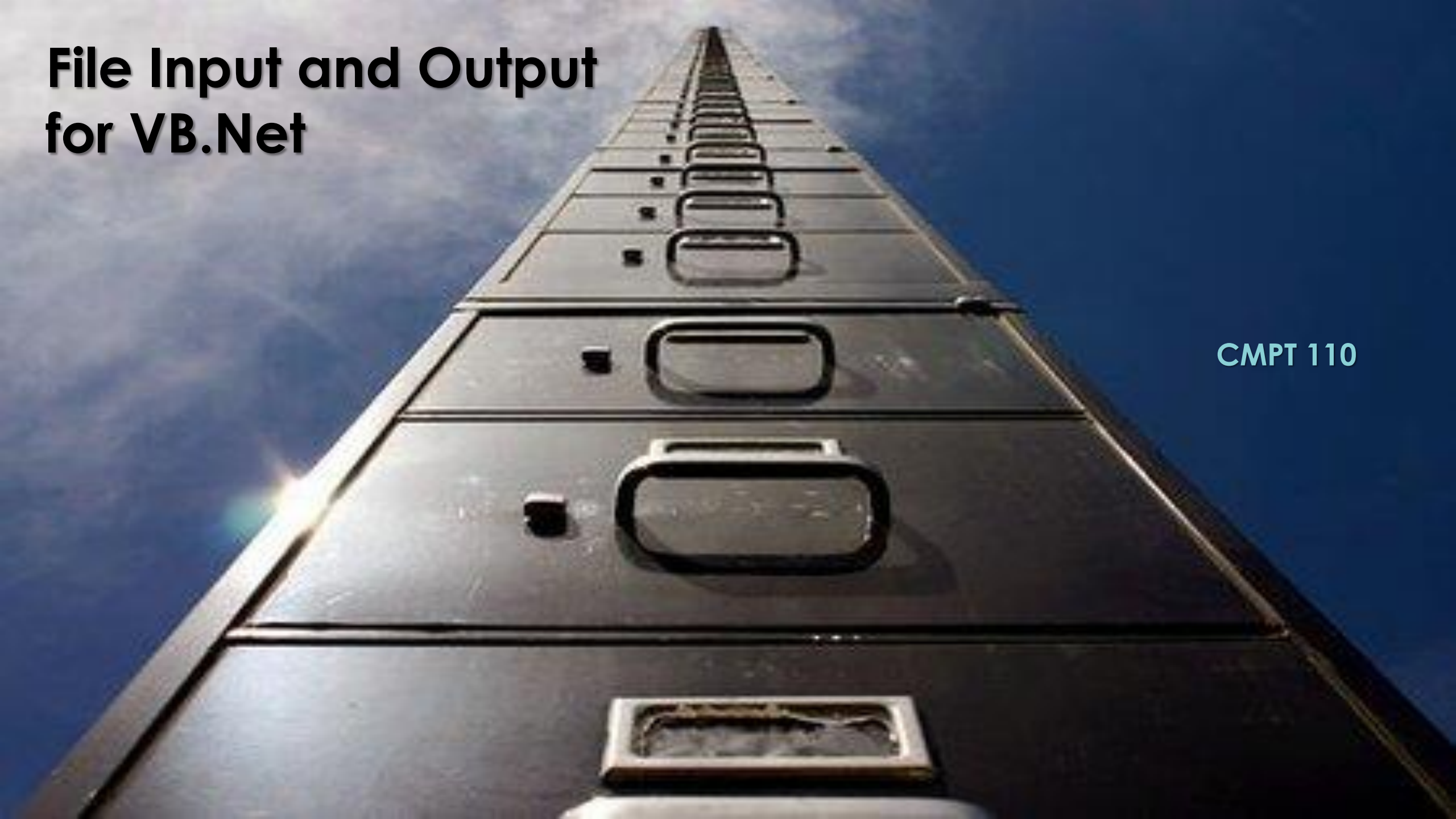


File Input and Output for VB.Net

CMPT 110



Objectives From Unit 10 – Study Guide

2

- ▶ To familiarize ourselves with all the steps required to **access files** successfully.
- ▶ To understand the **stream model of data storage** and how **sequential files** are accessed.
- ▶ To learn how to program to handle simple **exceptions**.

Additional Objectives

- ▶ Define fundamental terminology.
- ▶ Define sequential versus random accessing of information.
- ▶ Define file handling terminology within the context of OO.

Preamble

DBMS Nomenclature

- Field
 - ▶ Basic element of data
- Record
 - ▶ An organization of related fields treated as a unit
- File
 - ▶ A collection of similar records
- Database
 - ▶ A collection of related data

Definition: File

- ▶ A **file** is 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**.
- ▶ The stream is basically the sequence of bytes passing through the communication path. There are two main streams:
 - ▶ the **input stream** and
 - ▶ the **output stream**.
- ▶ The **input stream** is used for reading data from file (read operation) and the **output stream** is used for writing into the file (write operation).

The File Concept from an OS View

7

- ▶ A Contiguous logical address space:
 - ▶ The OS makes abstractions of the physical properties of its secondary storage device(s) to define a logical storage unit called a *file*.
 - ▶ The OS maps files to physical devices.

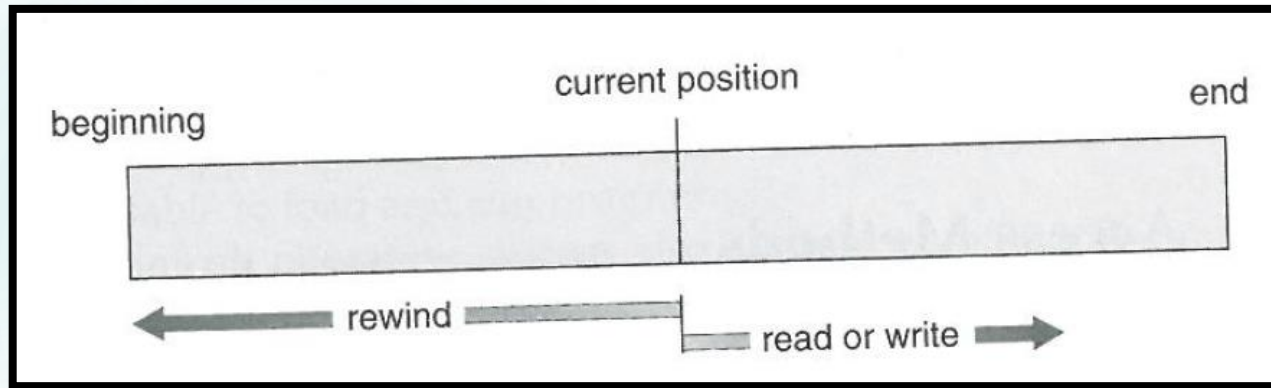
File Types

- ▶ File type refers to the ability of the operating system to distinguish different types of files such as text files source files and binary files etc.
- ▶ Fundamental Types of files
 - ▶ Data
 - ▶ numeric, character, binary
 - ▶ Program
 - ▶ source, object
 - ▶ Documents

File Access

9

- ▶ Two fundamental classes:
 - **Sequential Access** (i.e., tape)



File Access

10

- ▶ Two fundamental classes:
 - Sequential Access (*i.e.*, tape)
 - **Direct Access** (*i.e.*, disk)
 - ▶ *a.k.a.*, random access



File Attributes

11

Attribute	Meaning
Protection	Who can access the file and in what way
Password	Password needed to access the file
Creator	ID of the person who created the file
Owner	Current owner
Read-only flag	0 for read/write; 1 for read only
Hidden flag	0 for normal; 1 for do not display in listings
System flag	0 for normal files; 1 for system file
Archive flag	0 for has been backed up; 1 for needs to be backed up
ASCII/binary flag	0 for ASCII file; 1 for binary file
Random access flag	0 for sequential access only; 1 for random access
Temporary flag	0 for normal; 1 for delete file on process exit
Lock flags	0 for unlocked; nonzero for locked
Record length	Number of bytes in a record
Key position	Offset of the key within each record
Key length	Number of bytes in the key field
Creation time	Date and time the file was created
Time of last access	Date and time the file was last accessed
Time of last change	Date and time the file was last changed
Current size	Number of bytes in the file
Maximum size	Number of bytes the file may grow to

Figure 4-4. Some possible file attributes.

File Operations

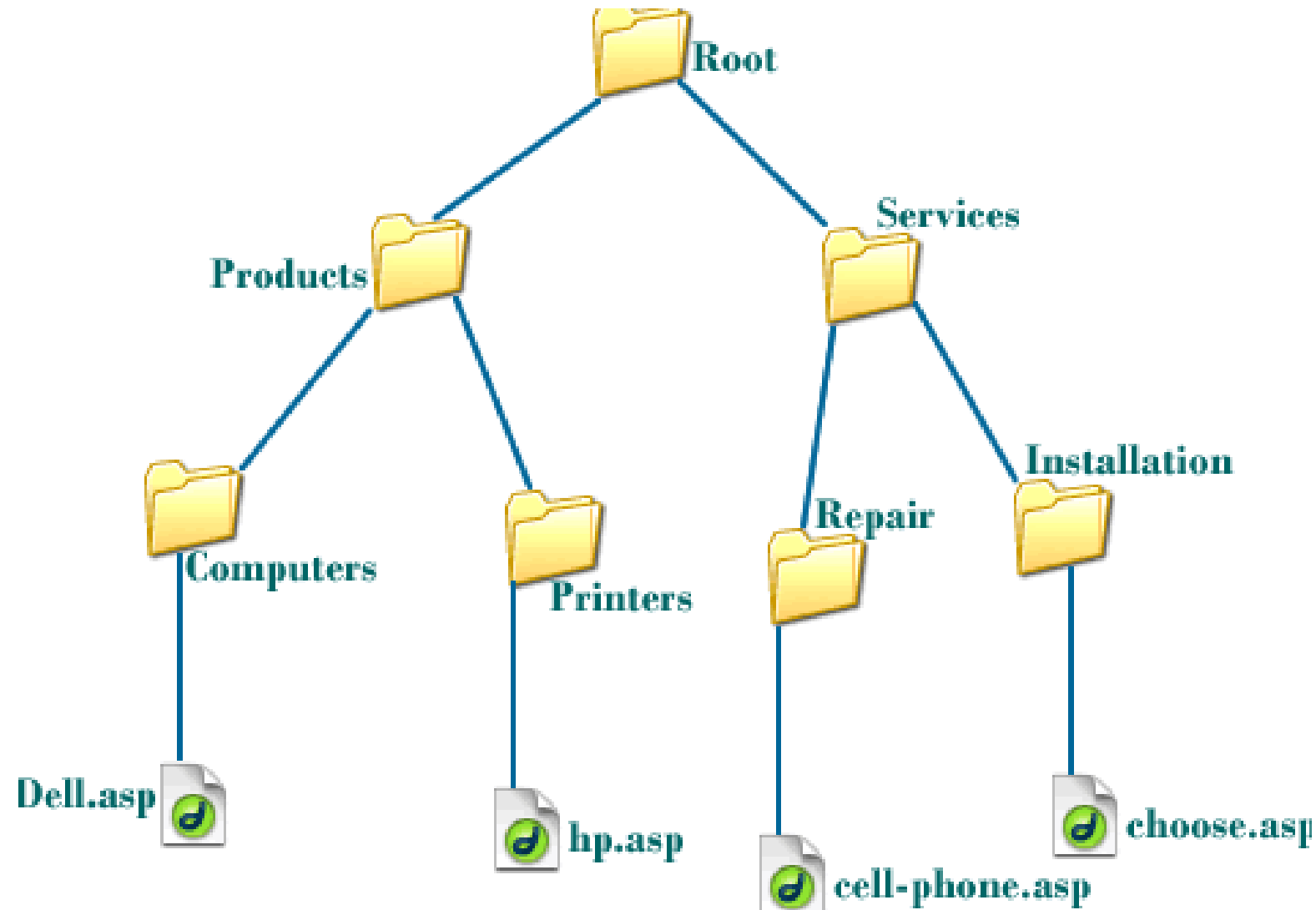
- ▶ A file is an abstract data type. It can be defined by operations:
 - ▶ Create a file
 - ▶ Write a file
 - ▶ Read a file
 - ▶ Reposition within file (file seek)
 - ▶ Delete a file
 - ▶ Truncate a file
 - ▶ Open(F_i)
 - ▶ search the directory structure on disk for entry F_i , and move the content of entry to memory.
 - ▶ Close(F_i)
 - ▶ move the content of entry F_i in memory to directory structure on disk.

Extensions

13

<u>FILE TYPE</u>	<u>EXTENSION</u>	<u>PURPOSE</u>
Executable	Exe, com, bin	Machine language program
Object	Obj, o	Compiled machine lang., not linked
Source code	c, CC, p, java, asm...	Source code in various languages
Batch	Bat, sh	Commands to command interpreter
text	Txt, doc	Textual data, documents
Print, view	ps, dvi, gif	ASCII or binary file
archive	Arc, zip, tar	Group of files, sometimes compressed
Library	Lib, a	Libraries of routines

Directory Structure: Tree Data Structure



VB.Net OO Details

Definition: Namespace

In [computing](#), a **namespace** is a set of symbols that are used to organize objects of various kinds, so that these objects may be referred to by name. Prominent examples include:

- [file systems](#) are namespaces that assign names to files;^[1]
- some [programming languages](#) organize their [variables](#) and [subroutines](#) in namespaces;^{[2][3][4]}
- [computer networks](#) and [distributed systems](#) assign names to resources, such as computers, printers, [websites](#), (remote) files, etc.

Definition: Namespace

17

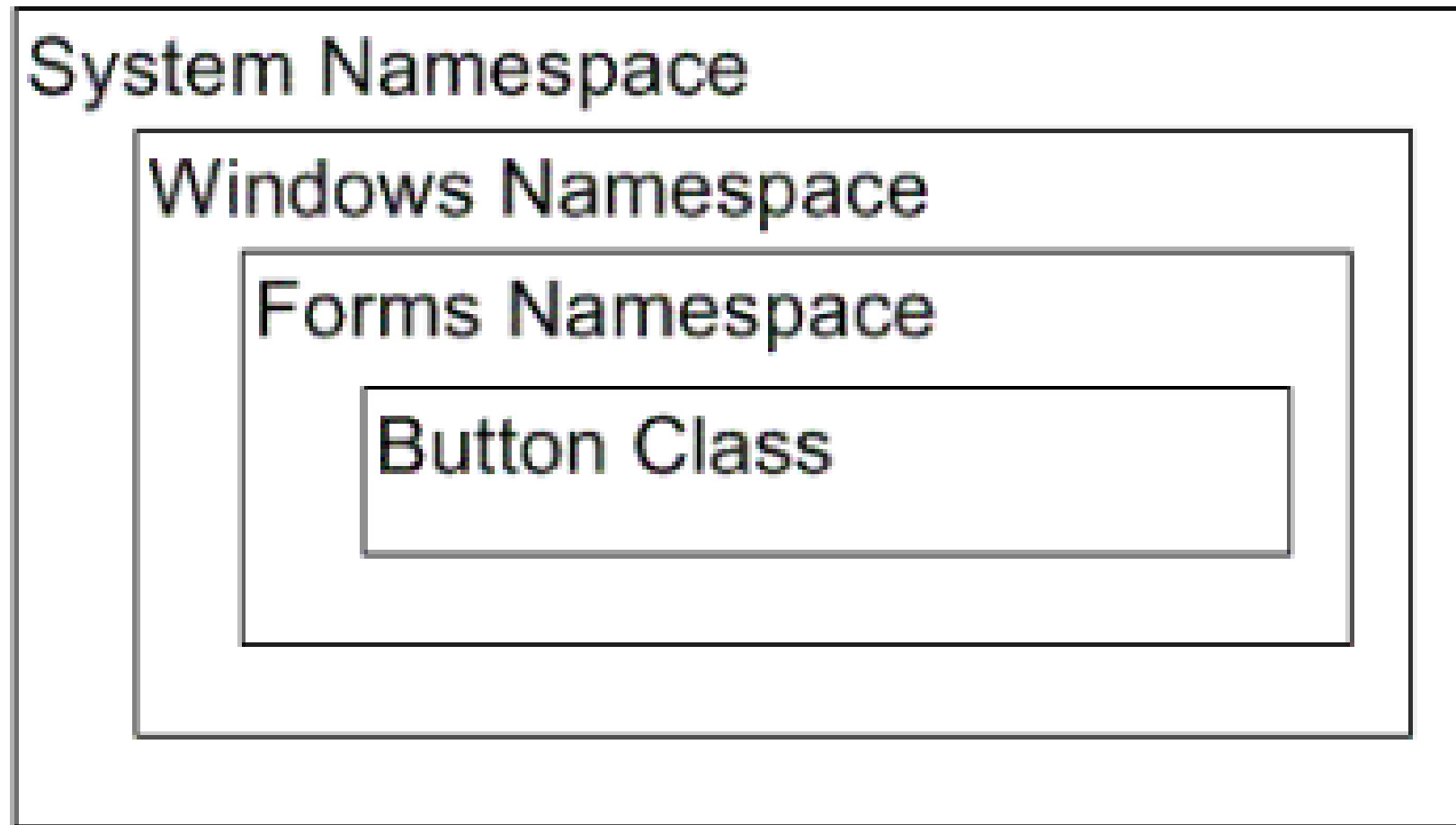
A namespace is an object-oriented concept which allows for organization of variables, functions, and other elements of a program. A namespace encapsulates code such that to access that code, you must first reference the namespace to which you are referring. The global namespace is one such default namespace – all global variables are considered to be in the "global namespace." Programmers can create other namespaces, thus organizing code and reducing pollution of the global namespace. The following example in C++ shows a good example of how namespaces work, and why they are useful:

```
namespace foo
{
    int a = 1; // Make an integer here
};
namespace bar
{
    int a = 2; // Make another integer here
};
cout << foo::a << "\n"; // Prints 1
cout << bar::a << "\n"; // Prints 2
cout << a << "\n"; // Illegal - "a" is undefined.
```

Understanding and Using Assemblies and Namespaces in .NET

- ▶ The Microsoft .NET Framework provides several ways to think of your code as more than just a bunch of disconnected lines. As a Visual Basic programmer, you're already familiar with the concept of a *class*, a section of code that defines an object and its behavior. But two of the higher-level groupings may be unfamiliar to you:
 - ▶ An **assembly** provides a fundamental unit of ***physical code grouping***.
 - ▶ A **namespace** provides a fundamental unit of ***logical code grouping***.
- ▶ As you'll see in this document, you can use Visual Basic .NET to create both assemblies and namespaces. You'll need to understand both of these concepts to be a productive Visual Basic .NET developer.
- ▶ Read further: <https://msdn.microsoft.com/en-us/library/ms973231.aspx>

A Namespace and Hierarchy in VB.Net



VB.Net I/O Classes

20

- ▶ The System.IO namespace has various classes that are used for performing various operations with files, like creating and deleting files, reading from or writing to a file, closing a file, etc.
- ▶ The following table on the next slide shows some commonly used non-abstract classes in the System.IO namespace.

VB.Net I/O Classes

21

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 of 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.

VB.Net FileStream Class

VB.Net FileStream Class

- ▶ The **FileStream** class in the System.IO namespace helps in reading from, writing to and closing files. This class derives from the abstract class Stream.
- ▶ You need to create a **FileStream** object to create a new file or open an existing file. The syntax for creating a **FileStream** object is as follows:

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

- ▶ For example, for creating a FileStream object **F** for reading a file named **sample.txt**:

```
Dim F As FileStream = New FileStream("sample.txt", FileMode.OpenOrCreate, FileAccess.ReadWrite)
```

The FileStream Class

- ▶ File streams can be stored in plain text and binary format.
- ▶ There are two types of file stream:
 - ▶ **Sequential-access file** (reading or writing data from the beginning of the file to the end and vice versa).
 - ▶ **Random-access file** (provides access a particular record immediately. The records in random-access file can be accessed directly and quickly without searching through a large number of records as in the case of sequential-access file. Random-access file is used in instant access systems such as banking system, sale system, air-line reservation system...etc.)

The FileStream Class

Example: Reading from a sequential-access file

Imports System.IO 'Don't forget to import IO package

Module Module1

Sub Main()

Dim SF As New SequentialFile

'reading from file

SF.readingFromFile("D:\sequentialfile.txt")

'Note: In this part, we introduce error handling with try..catch..end try block

End Sub

Class SequentialFile

Public Sub readingFromFile(ByVal filename As String)

Dim fn As FileStream

Dim fr As StreamReader

Try

'Open file for reading

fn = New FileStream(filename, FileMode.Open, FileAccess.Read)

'Create reader tool

fr = New StreamReader(fn)

'Read the content of file

Dim filecontent As String = fr.ReadToEnd

Console.WriteLine(filecontent)

Console.ReadLine()

Catch e As IOException

MsgBox(e.Message)

End Try

'Close file

fr.Close()

fn.Close()

End Sub

End Class

The FileStream Class

Example: Writing to a sequential-access file

Imports System.IO 'Don't forget to import IO package
Module Module1

```
Sub Main()  
    Dim SF As New SequentialFile  
    SF.writingToFile("D:\sequentialfile.txt")  
    'Note: In this part, we introduce error handling with try..catch..end try block  
End Sub
```

```
Class SequentialFile  
    Public Sub writingToFile(ByVal filename As String)  
        'Open file for writing  
        Dim fout As New FileStream(filename, FileMode.OpenOrCreate, FileAccess.Write)  
        'Create StreamWriter tool  
        Dim fw As New StreamWriter(fout)  
        'write text to file  
        fw.WriteLine("Hello World")  
        'close file  
        fout.Close()  
        fw.Close()  
    End Sub  
End Class  
End Module
```

The FileStream Class

27

See the link below for an example of reading and writing using random-access (which we are not covering in this course).



You can also refer to the following video:

<https://www.youtube.com/watch?v=LNLogeRqce8>

The FileStream Class

28

Parameter	Description
FileMode	<p>The FileMode enumerator defines various methods for opening files. The members of the FileMode enumerator are:</p> <ul style="list-style-type: none">•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.

The FileStream Class

29

Parameter	Description
FileAccess	FileAccess enumerators have members: Read , ReadWrite and Write .
FileShare	FileShare enumerators have the following members: <ul style="list-style-type: none">• 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: Example

The following program demonstrates use of the **FileStream** class:

```
Imports System.IO
Module fileProg
    Sub Main()
        Dim f1 As FileStream = New FileStream("sample.txt", _
            FileMode.OpenOrCreate, FileAccess.ReadWrite)
        Dim i As Integer
        For i = 0 To 20
            f1.WriteByte(CByte(i))
        Next i
        f1.Position = 0
        For i = 0 To 20
            Console.Write("{0} ", f1.ReadByte())
        Next i
        f1.Close()
        Console.ReadKey()
    End Sub
End Module
```

When the above code is compiled and executed, it produces the following result:

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 -1
```

Exercise: What does this function do?

31

```
Function IntRead(ByRef stream As String) As Integer
    Dim ch As String, txt As String
    txt = GetNextChar(stream)
    If txt <> "-" And (txt < "0" Or txt > "9") Then
        Return 0 ' Can't make an integer so return 0
    Else
        ' txt is now initialized to either "-" or first digit.
        ' Now obtain the remaining digits, terminating when
        ' a non-digit is encountered:
        Do While Len(stream) > 0
            ch = GetNextChar(stream)
            If ch >= "0" And ch <= "9" Then
                txt = txt & ch
            Else
                ' Non-digit terminates the input
                ' Convert txt to Integer and return.
                Return CInt(txt)
            End If
        Loop
        ' End of stream encountered when Len(stream) = 0.
        ' Convert txt to Integer and return
        Return CInt(txt)
    End If
End Function
```

```
Function GetNextChar(ByRef stream As String)
    Dim ch As String
    ch = stream.Substring(0, 1) ' Get next character of stream
    stream = stream.Substring(1) ' Remove 1st character of stream
    Return ch
End Function
```


Exercise: What does this function do?

32

Processing Character Sequences

Regardless of data type, data is represented by a sequence of characters. These characters can be grouped logically into sequences that can represent `Integer`, `Double`, `Boolean`, or `String` values. It would be pretty tedious if the programmer had to write subprograms that grouped the characters into suitable sequences depending on the type of value that was being read. Not surprisingly, therefore, the Visual Basic programming environment provides such routines and performs these kinds of tasks, known as "lexical analysis" and "parsing," so you won't have to write these subprograms yourself. However to give you a better understanding of what is required, consider the problem of extracting enough characters from a long string of characters representing a sequence of integers in order to obtain the integer values. To solve the problem the program must be able to do the following:

- Distinguish digits from other possible characters
- Determine when one number ends and another begins
- Convert the sequence of digits into a numeric value.



The following Function models the behaviour of a stream using a `String` called `stream`. The function `GetNextChar` returns characters one at a time from stream. Once the character has been retrieved, it is removed from stream.

From the Unit 10 Study Guide

File Specification and Stream Access

33

From the requirements analysis, the programmer must decide whether a file is to be opened for the purpose of obtaining input data from the file or for the purpose of writing data to a new or existing file. In either case, a `Dim` statement is used to declare a variable that will represent an object that will model the behaviour of the file.

If the objective is to read from an existing file, then the type of object you need to create is a `StreamReader` object and therefore the appropriate declaration statement is:

```
Dim rdr As IO.StreamReader
```

Otherwise, if you wish to create a new file or append data to the end of an existing file, then the type of object required is a `StreamWriter` object and the appropriate statement is:

```
Dim wrtr As IO.StreamWriter
```

In each of these examples, `rdr` and `wrtr` are simply variable names. Although the declaration statements describe what kind of values they will have, as yet they are undefined. To give them values, since they will represent files, it seems reasonable that you, the programmer, will need to provide the location and name of the file that you will be reading from or writing to. The location and name is provided by a "file-spec" expressed as a character string.

File Specification and Stream Access

34

File-specs

If you have accessed files in a Windows environment, you will recall that files are stored in folders and that these folders may be inside other folders. Furthermore the top-level folders are stored on virtual disks labelled "A:", "C:", "D:", etc. If you have accessed files using the Command Window, then you will have used file-specs to specify the full name of the file.

For example, if a file called "data.txt" resides in a folder called "stats" and that folder resides on virtual disk "C:" then the file-spec for accessing that file is given by:

```
C:\stats\data.txt
```

As a second example, suppose that the folder "stats" is inside another folder called "results" and "results" is on virtual drive "D:". Then the file-spec is given by:

```
D:\results\stats\data.txt
```

File Specification and Stream Access

35

Accessing Files

In Visual Basic, these file-specs can be expressed as Strings by enclosing them in double-quotes. So, to complete the process of opening a file for reading, having declared a variable of type `StreamReader`, you assign it a value that identifies the file-spec and assigns it "Open" mode, indicating that the file is to be opened for reading. As an example using the declared variable, `rdr`, this process can be accomplished with the assignment statement:

```
rdr = IO.File.OpenText("C:\stats\data.txt")
```

where `OpenText` is a method of `StreamReader` objects whose purpose is to open existing files for the purpose of reading from them.

When storing data in a file, there are two possible "modes" under which a file can be opened:

1. "Create" mode is used if output is to be made to a new file.
2. "Append" mode is used if data is to be added to the end of an existing file.

So, to assign a value to the `StreamWriter` object, `wtr`, you can indicate that you are opening a new file to store your output with the assignment statement:

```
wtr = IO.File.CreateText("C:\stats\newdata.txt")
```

`CreateText` is a method of `StreamWriter` objects for use in creating new files for the purpose of storing data in them.

Alternatively, if you wish to add additional output to the end of the existing file `C:\stats\data.txt` then you should open the file using the "AppendText" method as follows:

```
wtr = IO.File.AppendText("C:\stats\data.txt")
```

Dealing With Abnormal Situations

36

Dealing With Abnormal Situations

37

An important part of programming with files is dealing with situations that can result in errors unless your programming checks for them. Such situations are called **exceptions**, and any program statements that you write to address these exceptions are called "**exception handling statements**."

Good programs are robust. That means that the programmer has tried to anticipate exceptions and deal with them if they occur rather than leave them unaddressed.

Visual Basic provides exception handling in three ways:

- ▶ By providing **default exception handlers** that are automatically executed if an exception occurs and you as the programmer have not explicitly addressed it.
- ▶ By providing a set of **Boolean functions that test for specific situations**.
- ▶ By providing statements that permit the programmer to **try to execute a statement that may not succeed**. If an error occurs, Visual Basic permits the programmer to attempt to recover from the error (**TRY-CATCH**) – refer to the video entitled "How to Browse Files, Folders and Directories" for a demonstration (located at the following website - <https://www.youtube.com/watch?v=BZr3vUAUSbU>).

Further Topics

Read the following from Study Guide 10:

- ▶ Transferring Data to Opened Files
- ▶ Transferring Data from Opened Files
- ▶ Terminating File Access
- ▶ Text Data File Processing
- ▶ Updating Files

Advanced Topics

39

Reading from and Writing into Text files (see slide #25)

It involves reading from and writing into text files.

The **StreamReader** and **StreamWriter** classes help to accomplish it.

Reading from and Writing into Binary files

It involves reading from and writing into binary files.

The **BinaryReader** and **BinaryWriter** classes help to accomplish this.

Manipulating the Windows file system

It gives a VB.Net programmer the ability to browse and locate Windows files and directories.

From Microsoft

40

The `My.Computer.FileSystem` object provides tools for working with files and folders. Its properties, methods, and events allow you to create, copy, move, investigate, and delete files and folders. `My.Computer.FileSystem` provides better performance than the legacy functions (`FileOpen`, `FileClose`, `Input`, `InputString`, `LineInput`, etc.) that are provided by Visual Basic for backward compatibility.

In This Section

[Reading from Files](#)

Lists topics dealing with using the `My.Computer.FileSystem` object to read from files

[Writing to Files](#)

Lists topics dealing with using the `My.Computer.FileSystem` object to write to files

[Creating, Deleting, and Moving Files and Directories](#)

Lists topics dealing with using the `My.Computer.FileSystem` object to creating, copying, deleting and moving files and folders.

[Parsing Text Files with the TextFieldParser Object](#)

Discusses how to use the `TextFieldReader` to parse text files such as logs.

[File Encodings](#)

Describes file encodings and their use.

[Walkthrough: Manipulating Files and Directories in Visual Basic](#)

Demonstrates how to create a utility that reports information about files and folders.

[Troubleshooting: Reading from and Writing to Text Files](#)

Lists common problems encountered when reading and writing to text files, and suggests remedies for each.

Exercise: What does this program do?

41

```
Private Sub create_Click()  
Dim intMsg As String  
Dim StudentName As String  
Open "c:\My Documents\sample.txt" For Output As #1  
intMsg = MsgBox("File sample.txt opened")  
StudentName = InputBox("Enter the student Name")  
Print #1, StudentName  
intMsg = MsgBox("Writing a" & StudentName & "to sample.txt")  
Close #1  
intMsg = MsgBox("File sample.txt closed")  
End Sub
```

Exercise: What does this program do?

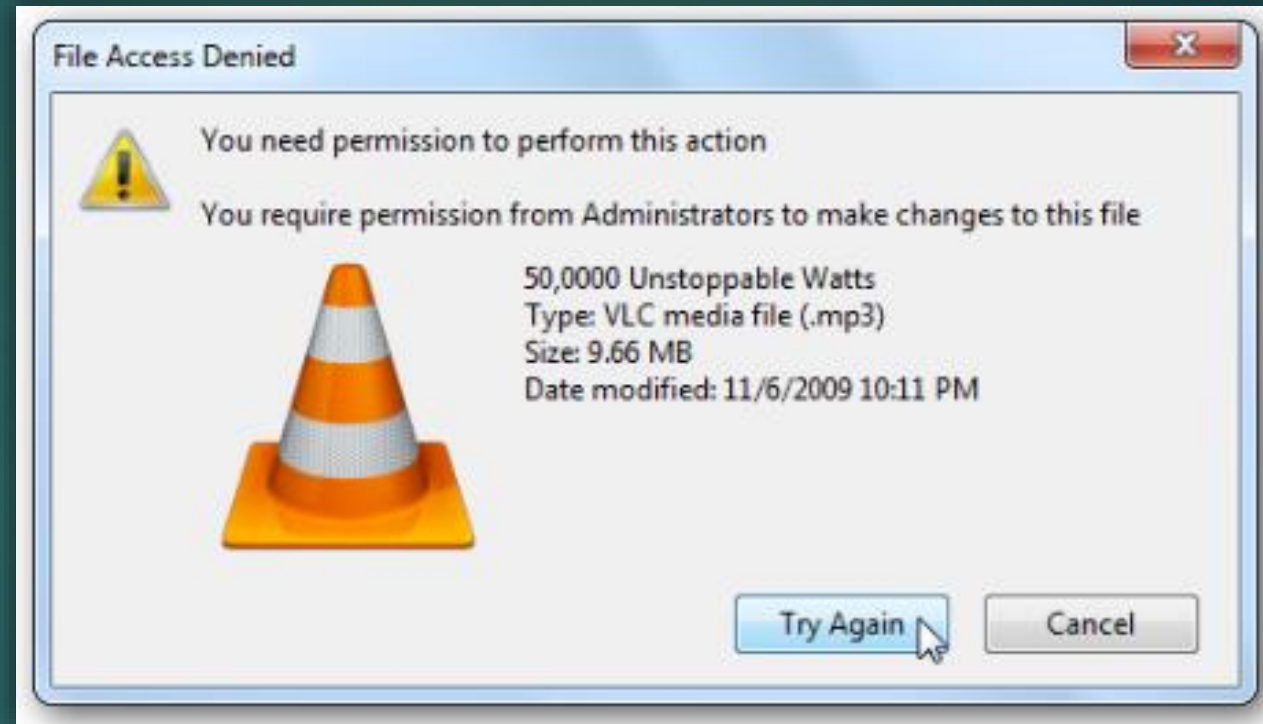
42

```
Private Sub create_Click()  
Dim intMsg As String  
Dim StudentName As String  
Open "c:\My Documents\sample.txt" For Output As #1  
intMsg = MsgBox("File sample.txt opened")  
StudentName = InputBox("Enter the student Name")  
Print #1, StudentName  
intMsg = MsgBox("Writing a" & StudentName & "to sample.txt")  
Close #1  
intMsg = MsgBox("File sample.txt closed")  
End Sub
```

The above program will create a file sample.txt in the My Documents folder and ready to receive input from users. Any data input by the user will be saved in this text file.

File Access Video in VB.Net

43



<https://www.youtube.com/watch?v=edGdjuJH85M>

Presentation
Terminated

44

