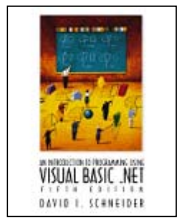
Chapter 8: Sequential Files

- Sequential Files (8.1)
  - Creating a Sequential File
  - Adding Items to a Sequential File
  - Error Trapping
- Using Sequential Files (8.2)
  - Sorting Sequential Files
  - Merging Sequential Files
  - Control Break Processing



David I. Schneider, *An Introduction to Programming using Visual Basic.NET, 5th Edition*, Prentice Hall, 2002.

Sequential Files (8.1)

- A sequential file consists of data stored in a text file on disk.
- May be created using Notepad
- May also be created directly from VB.NET

Sequential Files (8.1) (cont.)

- Creating a Sequential File

1. Choose a filename – may contain up to 215 characters
2. Select the path for the folder to contain this file
3. Execute a statement like the following:

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

Sequential Files (8.1) (cont.)

- Creating a Sequential File…

1. Place a line of data into the file with a statement of the form:

   sw.WriteLine(datum)

2. Close the file:

   sw.Close()

Sequential Files (8.1) (cont.)

- Example 1

```
Private Sub btnCreateFile_Click(...) _
    Handles btnCreateFile.Click
  Dim sw As IO.StreamWriter =
       IO.File.CreateText("PIONEERS.TXT")
  With sw
   .WriteLine("Atanasoff")
   .WriteLine("Babbage")
   .WriteLine("Codd")
   .WriteLine("Dijkstra")
   .Close()
  End With
End Sub
```

Sequential Files (8.1) (cont.)

- Caution
  - If an existing file is opened for output, the computer will erase the existing file and create a new one.

Sequential Files (8.1) (cont.)

- Adding Items to a Sequential File

1. Execute the statement

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

where sw is a variable name and filespec identifies the file.

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

```
sw.Close()
```

Sequential Files (8.1) (cont.)

- IO.File.AppendText
  - Will add data to the end of an existing file
  - If a file does not exist, it will create one

Sequential Files (8.1) (cont.)

- Sequential File Modes
  - CreateText
  - OpenText
  - AppendText
  - A file should not be opened in two different modes at the same time.

Sequential Files (8.1) (cont.)

- Avoiding Errors
  - Attempting to open a non-existent file for input causes the following error:

```
'System.IO.FileNotFound Exception' occurred…
```

  - There is a method to determine if a file exists before attempting to open it:

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

  will return a True if the file exists

Sequential Files (8.1) (cont.)

- Testing for the Existence of a File

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

Sequential Files (8.1) (cont.)

- Deleting Information from a Sequential 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 renamed with the name of the original file.

Sequential Files (8.1) (cont.)

- Delete and Move Methods
  - Delete method:

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

  - Move method (to change the filespec of a file):

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

  - ***Note:*** The IO.File.Delete and IO.File.Move methods cannot be used with open files.

Sequential Files (8.1) (cont.)

- Imports System.IO
  - Simplifies programs that have extensive file handling.
  - Place the statement

```
Imports System.IO
```

    near the top of the Code window, just after the Option Strict On statement. Then, there is no need to insert the prefix "IO." before the words StreamReader, StreamWriter, and File.

Sequential Files (8.1) (cont.)

- Structured Exception Handling
  - Two types of problems in code:
    - *Bugs* – something wrong with the code the programmer has written
    - *Exceptions* – errors beyond the control of the programmer
  - Programmer can use the debugger to find bugs; but must anticipate exceptions in order to be able to keep the program from terminating abruptly.

Sequential Files (8.1) (cont.)

- How VB.NET Handles Exceptions
  - An unexpected problem causes VB.NET first to throw an exception then to handle it.
  - If the programmer does not explicitly include exception-handling code in the program, then VB.NET handles an exception with a default handler.
  - The default exception handler terminates execution, displays the exception's message in a dialog box and highlights the line of code where the exception occurred.

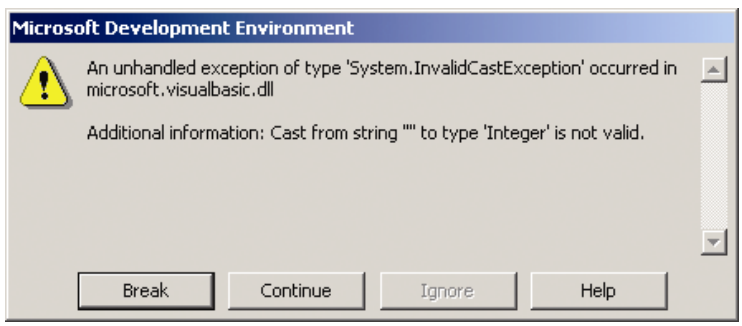
Sequential Files (8.1) (cont.)

- Exception Example
  - If the user enters a word or leaves the input box blank in the following program, an exception will be thrown:

```
Dim taxCredit As Double
Private Sub btnComputeCredit_Click(...)
    Handles btnComputeCredit.Click
  Dim numDependants As Integer
  numDependants = CInt(InputBox( _
              "How many dependants do you
  have?"))
   taxCredit = 1000 * numDependants
End Sub
```

Sequential Files (8.1) (cont.)

- Exception Handled by VB.NET



Sequential Files (8.1) (cont.)

- Try Catch Block

```
Dim taxCredit As Double
Private Sub btnComputeCredit_Click(...)
    Handles btnComputeCredit.Click
  Dim numDependents As Integer, message As String
  Try
    numDependents = CInt(InputBox("How many" & _
                    " dependents do you have?"))
  Catch
    message = "You did not answer the question " _
            & " with an integer value. We will " _
            & " assume your answer is zero."
    MsgBox(message)
    numDependents = 0
  Finally
    taxCredit = 1000 * numDependents
  End Try
End Sub
```

Sequential Files (8.1) (cont.)

- Catch Blocks
  - VB.NET allows Try-Catch-Finally blocks to have one or more specialized Catch clauses that only trap a specific type of exception.
  -
  - The general form of a specialized Catch clause is

```
Catch exp As ExceptionName
```

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

Sequential Files (8.1) (cont.)

- Try Catch Block Syntax

```
Try
  normal code
Catch exc1 As FirstException
  exception-handling code for FirstException
Catch exc2 As SecondException
  exception-handling code for
  SecondException
.
.
Catch
  exception-handling code for any remaining
  exceptions
Finally
  clean-up code
End Try
```

Sequential Files (8.1) (cont.)

- Exception Handling and File Errors
  - Exception handling can also catch file access errors.
  - File doesn't exist causes an IO.FileNotFoundException
  - If the Internet connection is broken an IO.IOException error is thrown.

Using Sequential Files (8.2)

- Sorting sequential files:

1. Read data from file into an array of UDT
2. Sort the data based on chosen field in UDT
3. Write sorted data to file

Using Sequential Files (8.2) (cont.)

- CSV File Format
  - Comma Separated Values
  - Records are stored on one line with a comma between each field

Using Sequential Files (8.2) (cont.)

- LSV File Format
  - Line Separated Values
  - Each value appears on its own line

Using Sequential Files (8.2) (cont.)

- Split Function
  - Facilitates working with CSV formatted files.
  - Split can convert a line containing commas into a String array.
  - The 0th element contains the text preceding the first comma, the 1st element contains the text between the first and second commas, ..., and the last element contains the text following the last element.

Using Sequential Files (8.2) (cont.)

- Split Example
  - For instance, suppose the String array employees() has been declared without an upper bound, and the String variable *line* has the value “Bob, 23.50, 45”.

```
employees = line.Split(",c)
```

  - sets the size of employees() to 3
  - sets employees(0) = “Bob”
  - employees (1) = “23.50”
  - employees(2) = “45”.

Using Sequential Files (8.2) (cont.)

- Split Comments

```
Employees = line.Split(",c)
```

  - In this example, character comma is called the delimiter for the Split function, and the letter c specifies that the comma has data type Character instead of String. (If Option Strict is Off, the letter c can be omitted.)
  - Any character can be used as a delimiter. If no character is specified, the Split function will use the space character as delimiter.

Using Sequential Files (8.2) (cont.)

- Example 2

```
Private Sub btnConvert_Click(...)
    Handles btnConvert.Click
  Dim stateData(), line As String, i As Integer
  line = "California, 1850, Sacramento, Eureka"
  stateData = line.Split(",")c)
  For i = 0 To stateData.GetUpperBound(0)
    stateData(i) = stateData(i).Trim
   'Get rid of extraneous spaces
    lstOutput.Items.Add(stateData(i))
  Next
End Sub
```

Using Sequential Files (8.2) (cont.)

- Example 2 Output

California
1850
Sacramento
Eureka

Using Sequential Files (8.2) (cont.)

- Example 3

```
Private Sub btnConvert_Click(...) Handles btnConvert.Click
  Dim line, fields(), fromFile, toFile As String
  Dim i As Integer
  Dim sr As IO.StreamReader
  Dim sw As IO.StreamWriter
  fromFile = InputBox("Name of original file:", _
             "Convert from CSV to LSV")
  toFile = InputBox("Name of converted file:", _
             "Convert from CSV to LSV")
  sr = IO.File.OpenText(fromFile)
  sw = IO.File.CreateText(toFile)
  Do While (sr.Peek() <> -1)
    line = sr.ReadLine()
    fields = line.Split(","c)
```

Using Sequential Files (8.2) (cont.)

- Example 3 continued

```
    For i = 0 To fields.GetUpperBound(0)
        sw.WriteLine(fields(i).Trim)
      Next
    Loop
    sr.Close()
    sw.Close()
    sr = IO.File.OpenText(toFile)
    Do While sr.Peek() <> -1
      lstFile.Items.Add(sr.ReadLine)
    Loop
    sr.Close()
  End Sub
```

Using Sequential Files (8.2) (cont.)

- Example 3 output

California
1850
Sacramento
Eureka
New York
1788
Albany
Excelsior

Using Sequential Files (8.2) (cont.)

- Join Function
  - The reverse of the Split function is the Join function
  - Join concatenates the elements of a string array into a string containing the elements separated by a specified delimiter.

```
Dim greatLakes() As String = _
  {"Huron","Ontario","Michigan","Erie","Superior"}
Dim lakes As String
lakes = Join(greatLakes, ",")
txtOutput.Text = lakes
```

produces the output

Huron,Ontario,Michigan,Erie,Superior

Using Sequential Files (8.2) (cont.)

- Merging Sequential Files Algorithm

1. Open the two ordered files for input, and open a third file for output.
2. Try to get an item of data from each file.
3. Repeat the following steps until an item of data is not available in one of the files:
   a) If one item precedes the other, write it into the third file and try to get another item of data from its file.
   b) If the two items are identical, write one into the third file and try to get another item of data from each of the two ordered files.

Using Sequential Files (8.2) (cont.)

- Merge Algorithm continued

4. At this point, an item of data has most likely been retrieved from one of the files and not yet written to the third file. In this case, write that item and all remaining items in that file to the third file.
5. Close the three files.

Using Sequential Files (8.2) (cont.)

- Control Break Processing
  - To create subtotals
  - When some value contained in a variable changes, such a variable is called a control variable
  - Each change of its value is called a break.

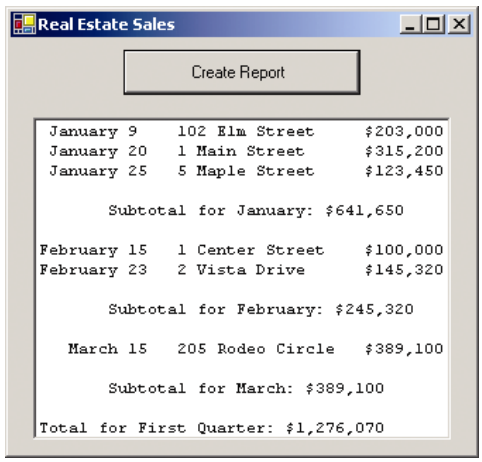
Using Sequential Files (8.2) (cont.)

- Data for Example 5

Month	Day	Address	Price
January	9	102 Elm St	$203,000
January	20	1 Main St	$315,200
January	25	5 Maple St	$123,450
February	15	1 Center St	$100,000
February	23	2 Vista Dr	$145,320
March	15	5 Rodeo Cir	$389,100

Using Sequential Files (8.2) (cont.)

- Output from Example 5



Using Sequential Files (8.2) (cont.)

- Comments
  - Files to be processed can be opened and closed within a single procedure.
  - Files can also be opened just once the instant the program is run and stay open until the program is terminated.
  - To open a file once, open it in the Form1_Load procedure and put the Close method and End statement in the click event procedure for a button labeled “Quit.”