Chapter 8 – Text Files

8.1 Managing Text Files

8.2 StreamReaders, StreamWriters, and Structured Exception Handling

8.3 XML

8.1 Managing Text Files

- Preliminaries
- WriteAllLines Method
- Sorting a Text File
- Set Operations
- Seaching a CSV Text File
- The OpenFileDialog Control

CSV File Format

- Comma Separated Values
- Records are stored on one line with a comma between each field
- Example: USStates.txt

```
Delaware,DE,1954,759000
Pennsylvania,PA,44817,12296000
New Jersey,NJ,7417,8135000
Georgia,GA,57906,7637000
```

(*name of state,abbreviation,area,population*)

LINQ Query for USStates.txt

```
Dim states() As String =
        IO.File.ReadAllLines("USStates.txt")
Dim query = From line In states
            Let data = line.Split(","c)
            Let name = data(0)
            Let abbr = data(1)
            Let area = CInt(data(2))
            Let pop = CInt(data(3))
            Select name, abbr, area, pop
```

DataGridView Control

- Used to display a table of data determined by a LINQ query.
- Found in the *Data* group and the *All Windows Forms* group of the Toolbox.

DataGridView for Query from USStates.txt

```
dgvStates.DataSource = query.ToList
dgvStates.CurrentCell = Nothing
```

	name	abbr	area	pop
	Delaware	DE	1954	759000
	Pennsylvania	PA	44817	12296000
	New Jersey	NJ	7417	8135000
	Georgia	GA	57906	7637000
	Connecticut	CT	4845	3271000

DataGridView Headers

By default the rows have blank headers and the column headers contain the names of the items in the Select clause.

column headers

row headers

	name	abbr	area	pop
	Delaware	DE	1954	759000
	Pennsylvania	PA	44817	12296000
	New Jersey	NJ	7417	8135000
	Georgia	GA	57906	7637000
	Connecticut	CT	4845	3271000

DataGridView Headers (cont.)

- Row headers can be deleted by setting the **RowHeadersVisible property** of the DataGridView control to False.
- A column header can be customized with a statement such as

```
dgvStates.Columns("area").HeaderText =
                                    "Land Area"
```

Altered Headers

State	State Abbreviation	Land Area	Population
Delaware	DE	1954	759000
Pennsylvania	PA	44817	12296000
New Jersey	NJ	7417	8135000
Georgia	GA	57906	7637000
Connecticut	CT	4845	3271000

Data in Table

- The data appearing in the DataGridView control can be modified by using Where and Order By clauses in the LINQ query. Or by changing the selection of items in the Select clause.
- **Note:** The Select clause must contain two or more items in order to use a DataGridView control.

Modified Data

```
Where name.StartsWith("New")
Order By area Descending
```

State	State Abbreviation	Land Area	Population
New Mexico	NM	121598	1823000
New York	NY	47214	18237000
New Hampshire	NH	8968	1165000
New Jersey	NJ	7417	8135000

Sorting a Text File

1. Read data from file into a string array.
2. Use a LINQ query to sort the data.
3. Write sorted data to a new file with the WriteAllLines method.

```
IO.File.WriteAllLines("fileName.txt",
                      strArrayOrQueryName)
```

File to Sort: AgeAtInaug.txt

```
George Washington,57
John Adams,61
Thomas Jefferson,57
James Madison,57
    .
    .
Barack Obama,47
```

Sort AgeAtInaug.txt by Age

```
Dim agesAtInaug() As String =
        IO.File.ReadAllLines("AgeAtInaug.txt")
Dim query = From line In agesAtInaug
            Let age = CInt(line.Split(","c)(1))
            Order By age
            Select line
IO.File.WriteAllLines("Sorted.txt", query)
```

File Sorted.txt

```
Theodore Roosevelt,42
John Kennedy,43
Ulysses Grant,46
Bill Clinton,46
    .
    .
Ronald Reagan,69
```

Ways to Combine Two Files

- Merge (with or without duplications)
- Create a file consisting of the items appearing in both files
- Delete items appearing in one file from the other file

The tasks above are carried out with the Set operations on arrays.

Set Operations on Arrays

- Concat – merges with duplications
- Union – merges without duplications
- Intersect – finds common items
- Except – deletes items in one array from the other array

Concat Operation

array1.Concat(array2).ToArray consists of the merge of the two arrays

```
Dim array1() = {"Alpha", "Bravo", "Charlie"}
Dim array2() = {"Bravo", "Delta"}
Dim array3() = array1.Concat(array2).ToArray
```

Size of array3: 5

Elements of array3: **"Alpha", "Bravo", "Charlie", "Bravo", "Delta"**

Union Operation

array1.Union(array2).ToArray consists of the merge of the two arrays without duplications

```
Dim array1() = {"Alpha", "Bravo", "Charlie"}
Dim array2() = {"Bravo", "Delta"}
Dim array3() = array1.Union(array2).ToArray
```

Size of array3: 4

Elements of array3: **"Alpha", "Bravo", "Charlie", "Delta"**

Intersect Operation

array1.Intersect(array2).ToArray consists of the items in both arrays

```
Dim array1() = {"Alpha", "Bravo", "Charlie"}
Dim array2() = {"Bravo", "Delta"}
Dim array3() = array1.Intersect(array2).ToArray
```

Size of array3: 1

Elements of array3: **"Bravo"**

Except Operation

array1.Except(array2).ToArray consists of the items in array1 that are not in array2

```
Dim array1() = {"Alpha", "Bravo", "Charlie"}
Dim array2() = {"Bravo", "Delta"}
Dim array3() = array1.Except(array2).ToArray
```

Size of array3: 2

Elements of array3: **"Alpha", "Charlie"**

Steps to Combine Two Files

1. Read each file into an array.
2. Apply a Set operation to the two arrays to create a third array.
3. Apply WriteAllLines to the third array to create a new text file.

Note: LINQ queries can be used in Step 2 for greater flexibility.

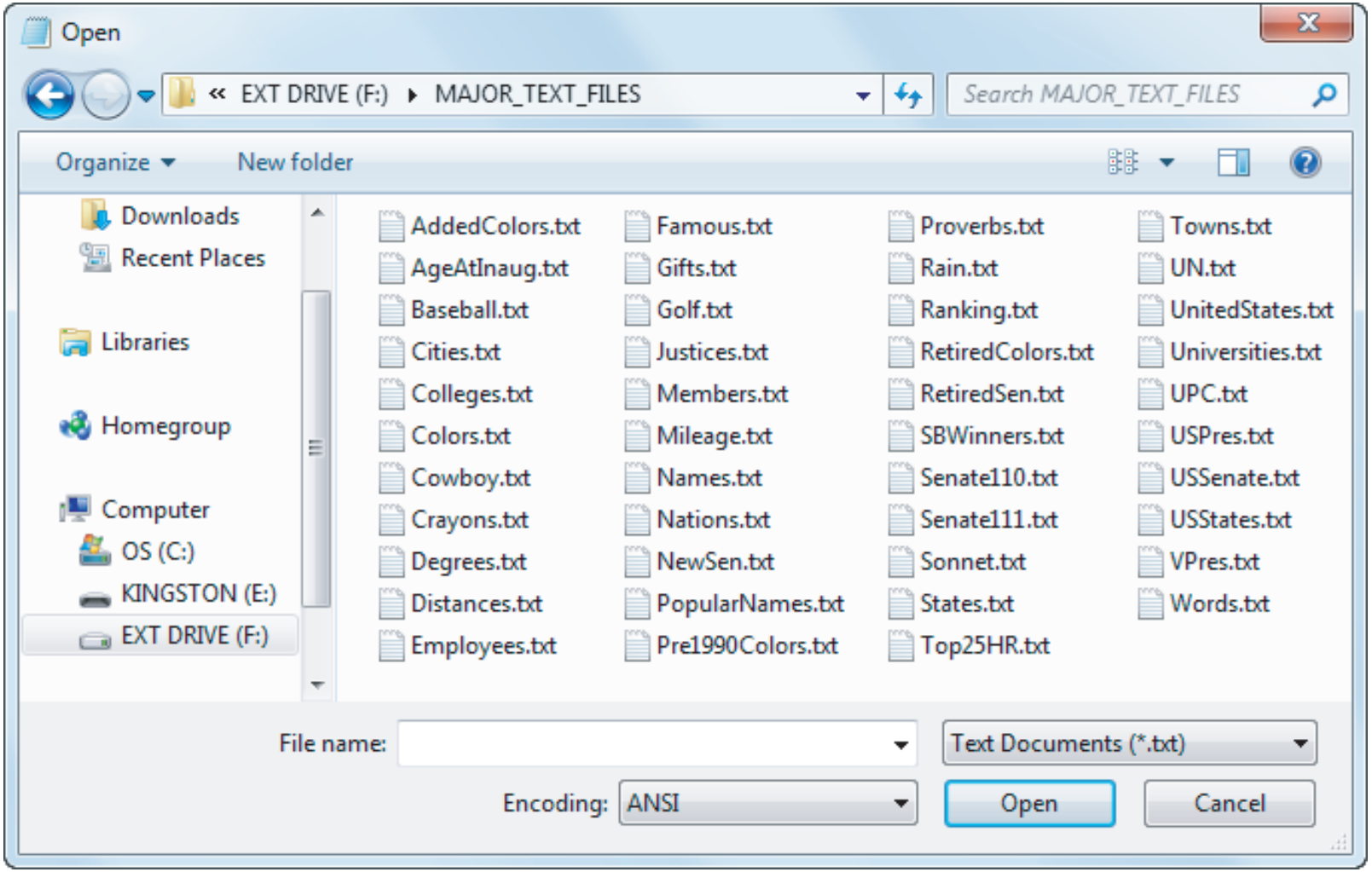
How to Search a Text File

1. Read the contents of the file into an array.
2. Use a LINQ query with a Where clause to search for the sought-after record.
3. If query.count = 0, the record was not found. Otherwise, the sequence returned by the query will contain the record.

The OpenFileDialog Control

- Implements the standard File Open dialog box
- Found in the *Dialogs* group of the Toolbox
- The icon and default name will appear in a component tray below the Document window.

OpenFileDialog Control



The Filter Property

Determines what appears in the box above the *Open* button, and what types of files will be displayed. The setting has the general form

text for box|*.*ext*

Example: Text Files (*.txt)|*.txt

Using the OpenFileDialog Control

- To display the control:

```
OpenFileDialog1.ShowDialog()
```

- After the *Open* button has been pressed, the file name selected and its complete filespec will be contained in the property:

```
OpenFileDialog1.FileName
```

Example 3: Task

- Select a text file and display its contents.
- **Note:** The Filter property of OpenFileDialog1 is set to

  Text Files (*.txt)|*.txt

Example 9: Code

```
Private Sub btnSelect_Click(...) Handles _
                                    btnSelect.Click
  Dim textFile As String
  OpenFileDialog1.ShowDialog()
  textFile = OpenFileDialog1.FileName
  lstOutput.DataSource =
                IO.File.ReadAllLines(textFile)
  lstOutput.SelectedItem = Nothing
End Sub
```

8.2 StreamReaders, StreamWriters, Structured Exception Handling

- Reading a Text File with a StreamReader
- Creating a Text File with a StreamWriter
- Adding Items to a Text File
- System.IO Namespace
- Structured Exception Handling

Reading Data from a Text File

- Data stored in a text file can be read one line at a time with a StreamReader object.
- The following statement declares a variable of type StreamReader and specifies the file to be read.

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

Note: A pointer is set to the first line of the file.

Reading Data from a Text File (continued)

- **`strVar = srVar.ReadLine`** reads the line pointed to, assigns the line to the string variable *strVar*, and moves the pointer to the next line of the file.
- The value of **`srVar.EndOfStream`** will be True after the entire file has been read.
- The statement **`srVar.Close()`** terminates communication with the file.

Reading Data from a Text File (continued)

If *sr* is a variable of type StreamReader, an entire text file can be read with a loop of the form

```
Do Until sr.EndOfStream
   strVar = srVar.ReadLine
  .
  .
Loop
```

Writing Data to a Text File

- Data can be placed in a text file one line at a time with a StreamWriter object.
- The following statement declares a variable of type StreamWriter and specifies the file to be created.

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

Writing Data to a Text File (continued)

- **`swVar.WriteLine(info)`** initally places the information into the first line of the file.
- Subsequent statements of that form place information into lines at the end of the file.
- The statement **`swVar.Close()`** terminates communication with the file.

Adding Items to a Text File

1. Execute the statement

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

   where *filespec* identifies the file.

2. Add lines of data to the end of the file with the WriteLine method.
3. After all the data have been written into the file, close the file with `swVar.Close()`.

Note: If the file does not exist, the AppendText method will create it.

Text File Modes

- OpenText – open for input
- CreateText – open for output
- AppendText – open for append
- A file should not be opened in two different modes at the same time.

Avoiding Errors

- Attempting to open a non-existent file for input brings up a message box titled:

  `FileNotFoundException`

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

  `IO.File.Exists(filespec)`

  will return True if the file exists.

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."
  MessageBox.Show(message, "File Not Found")
End If
```

Deleting Information from a Text 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.

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.

Imports System.IO

- Simplifies programs that have extensive file handling.
- Place the statement **`Imports System.IO`** at the top of the Code Editor, before the **`Class`** ***`frmName`*** statement. Then, there is no need to insert the prefix “IO.” before the words StreamReader, StreamWriter, and File.

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.

How Visual Basic Handles Exceptions

- An unexpected problem causes Visual Basic first to throw an exception then to handle it.
- If the programmer does not explicitly include exception-handling code in the program, Visual Basic handles exceptions 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.

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 numDependents As Integer
  numDependents =
        CInt(InputBox("How many dependents?"))
  taxCredit = 1000 * numDependents
End Sub
```

Exception Handled by Visual Basic

InvalidCastException was unhandled

Conversion from string "" to type 'Integer' is not valid.

Troubleshooting Tips

When casting from a number, the value must be a num
Make sure the source type is convertible to the destinat
Get general help for this exception.

Search for more Help Online...

Actions

View Detail...

Try-Catch-Finally Block

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

Catch Blocks

- Visual Basic allows Try-Catch-Finally blocks to have one or more specialized Catch clauses that 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.

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
```

Exception Handling and File Errors

- Exception handling can also catch file access errors.
- File doesn't exist causes an IO.FileNotFoundException
- If an attempt is made to delete an open file, IO.IOException is thrown.

8.3 XML

- Format of XML Files
- LINQ to XML

XML Files

- XML formatted files are a more robust alternative to CSV files.
- XML stands for e**X**tensible **M**arkup **L**anguage.

Sample CSV File

First two lines of USStates.txt

```
Delaware,DE,1954,759000
Pennsylvania,PA,44817,12296000
```

name abbreviation area population

XML Formatted Version

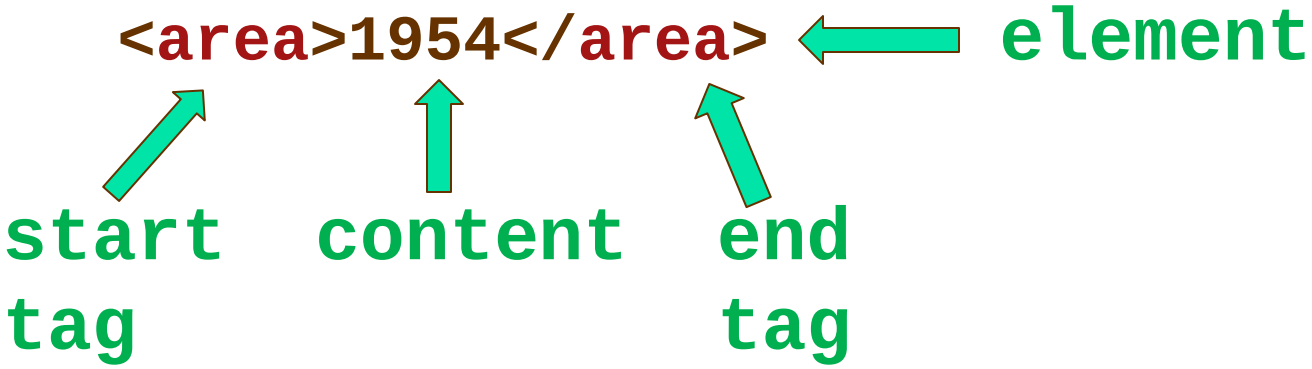
```
<?xml version='1.0'?>
<us_states>
  <state>
    <name>Delaware</name>
    <abbreviation>DE</abbreviation>
    <area>1954</area>
    <population>759000</population>
  </state>
```

(continued on next slide)

XML Version (continued)

```
    <state>
      <name>Pennsylvania</name>
      <abbreviation>PA</abbreviation>
      <area>44817</area>
      <population>1229600</population>
    </state>
</us_states>
```

XML Lingo



More XML Lingo

```
<state>
  <name>Delaware</name>
  <abbreviation>DE</abbreviation>
  <area>1954</area>
  <population>749000</population>
</state>
```

child of *state*

siblings (descendants of *state*)

state is a parent of *name*, *abbreviation*, *area*, and *population*

CSV Format versus XML

- CSV files are loaded into arrays
- XML files are loaded into XElement objects

```
Dim xmlElementName As XElement =
                      XElement.Load(filespec)
```

CSV Format versus XML

- With CSV files, Split is used to access a field.
- With XML files, an expression of the form <childName>.Value is used to access a field.

```
Dim stateData As XElement =
                    XElement.Load("USState.xml")
Dim query = From st In
                  stateData.Descendants("state")
              Let name = st.<name>.Value
```