CSE 1530

Introduction to Computer Use II: Programming

Winter 2006 (Section M)

Topic F: External Files and Databases – Using Classes and Objects

Friday, March 31 2006

Bill Kapralos

Overview (1):

- **Before We Begin**
  - Some administrative details
  - Some questions to consider
- **Writing to a File**
  - Writing to a new file → example program
  - Appending to an existing file
  - Modifying a file
- **Text Files Containing Fields and Records**
  - Overview
  - Reading/Writing such files

Overview (2):

- **Error Handling**
  - Detecting and handling run-time errors

Before We Begin

Administrative Details (1):

- **Exercise 7-6**
  - Due Monday, April 3 2006 before noon
  - I will be in the Glade Lab today after the lecture
  - Today's lecture will cover the material you require
- **Last Lecture is Monday, April 3**
  - Entire lecture will be review for exam
  - Bring your questions!

Some Questions to Consider (1):

- What is the OpentTextFile method for ?
- What does the OpenTextFile method return ?
- What is the TextStreamClass ?

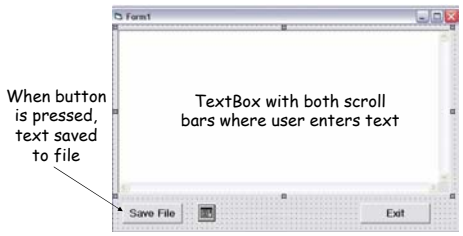
Writing To a File

Overview (1):

- **Writing to a File May Take Three Different Forms**
  1. The file we write to may be a new file
  2. The file we write to may be an existing file and we simply want to append data to the file (e.g., add data to the end of the file) → the data that was originally in the file remains the same
  3. The file we write to exists and we can write data to it anywhere → the original data may be modified in any way (e.g., some of it may be erased, some of it over-written etc.)

Writing to a New File (1):

- **Example Program**
  - Develop a program that allows the user to type text in a TextBox and then save the text to a file



Form in design mode

Writing to a New File (2):

- **Example Program (cont.)**
  - The InitDir property should be set to a directory appropriate to your computer system
    - Set this during design mode
  - The Filter and Flags properties will be set during run-time
  - The Filter property should be set to text files since we will be dealing with text files only
    - cdlSave.Filter = "Text Files (*.txt) | *.txt"

Writing to a New File (3):

- **Setting The Flags Property**
  - When the common file dialog appears, the user might specify the name of a new file, choose an existing file which they wish to over-write or type the name of an existing file
    - We can assume the user wishes to create a new file so if an existing file is specified we can assume user wishes to replace it
    - When specifying an existing file, we need to prompt the user and confirm with them that they wish to replace/overwrite the file

Writing to a New File (4):

- **Setting The Flags Property (cont.)**
  - Setting the Flags property
    - cdlSave.Flags = cdlOFNoverwritePrompt
  - We have now performed all the set-up (initialization) for this program and we are now ready to actually write the data to the file

Writing to a New File (5):

- **Displaying the "Save As" Dialog**
  - Since we want to save the file (and not simply open it), we need to display the "Save As" dialog box
    - ShowSave method → cdlSave.ShowSave
- **Writing the Data to a File**
  - We will need references to the FileSystemObject class and the TextStream class
    - Of course, we need to create objects for these references → recall the Set and New keywords!
  - The FileSystemObject class contains a method called CreateTextFile

Writing to a New File (6):

- **Writing the Data to a File (cont.)**

```
Function CreateTextFile(FileName As String,
    [Overwrite As Boolean = True],
    [Unicode As Boolean = False])
    As TextStream
    Member of Scripting.FileSystemObject
    Create a file as a TextStream
```

  - The CreateTextFile method → will create the file for us (if the file doesn't exist) or allow us to over-write data of an existing file if the Overwrite argument is True (it is by default)

Writing to a New File (7):

- **Writing the Data to a File (cont.)**
  - Basically, after calling the CreateTextFile method, it returns a TextStream object to us
    - Think of the TextStream object as an abstraction to the actual file itself → no need to worry about how the data will be written to the file, we only use the provided methods!
    - The TextStream object contains a method called Write → requires a String argument (the data to be written to the file)

Writing to a New File (8):

- **Writing the Data to a File (cont.)**
  - Lets look at the Visual Basic code…

Appending Data to a File (1):

- **Recall → The OpenTextFile Function**
  - Before we begin, lets take a closer look at the description for the OpenTextFile method

```
Function OpenTextFile(FileName As String, _
    [IOMode As IOMode = ForReading], _
    [Create As Boolean = False], _
    [Format As Tristate = TristateFalse]) _
    As TextStream
```

Appending Data to a File (2):

- **Very Similar to Writing Data to a New File**
  - The argument called IOMode has a default value of ForReading
    - This argument specifies the mode that the file has been opened for → ForReading specifies the file is opened for reading only (e.g., can only read data form the file and not write to it)
  - Alternative values are ForWriting and ForAppending
    - ForWriting → writing data to the file
    - ForAppending → appending data to a file

Modifying a File (1):

- **May Modify Any Portion of the File**
  - Can change and erase any portion of the file
  - Can also add (append) data to the file
  - Can add data to any part of the file
  - Can also read the file as well
  - Once again, modifying a file in such a manner is specified via the IOMode argument of the OpenTextFile
  - Set the argument value to ForWriting

Files Containing Fields & Records (1):

- **File Contents Can Be Data In The Form of Lists of Values**
  - Instead of containing (English) text in sentences and paragraphs, the file may contain data in the form of a list of values
  - As an example, consider a file used in a personal address/phone book → it may look as follows

    Kapralos; Bill; (416) 555-5555; 4700 Keele St.; Toronto

    - Or in general

    last name; first name; phone number; street; city

Files Containing Fields & Records (2):

- **What is a Record ?**
  - A complete set of information
  - Composed of fields
    - Each field in the record contains one piece of information
    - Fields within a record are separated by a special separator character
  - Typically, we will have some collection of records within a single file
    - Simplifying assumption → each line in the file will contain one record
    - Each record will contain all fields

Files Containing Fields & Records (3):

- **Example Records**
  - Phone list → a record is an individual person's info. and may have the following general form (where the separator character is the ";")

    Last name; first name; phone number

    - Or more specifically

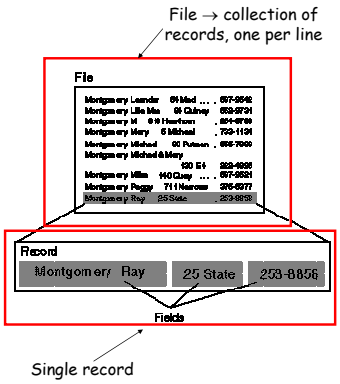
    Kapralos; Bill; (416) 555-5555

Files Containing Fields & Records (4):

- **Example Records (cont.)**
  - Address book entries →

    Last name first name phone number

    - Separator character is "white-space" between fields



Files Containing Fields & Records (5):

- **How Do We Work With Such Files ?**
  - Visual Basic provides the appropriate functions to access records and the fields of records
  - To read records form a file
    - Read one line of a file at a time using the ReadLine method of the TextStream class → since one record per line, essentially we are reading one record at a time
    - Access the fields of the record using the available String-related functions

Files Containing Fields & Records (6):

- **How Do We Work With Such Files ? (cont.)**
  - To write records to a file (append)
    - Create the string representing the "new" record
    - Write the string (hence the record) to the previously opened file → of course, file has been opened for appending

Handling Errors

Handling Errors (1):

- **In The Event of a Run-Time Error**
  - While executing a program, in the event of a run-time error, Visual Basic generates an object of the ErrObject class
  - This always happens however, it is up to us (the programmers) whether or not we want to actually handle the error
    - Whether or not we wish to write the code to take advantage of the information about the error that the object provides

Handling Errors (2):

- **In The Event of a Run-Time Error (cont.)**
  - If we ignore the error
    - Program will "crash" with a run-time error window being displayed
  - If we detect and handle the error
    - Write code to issue a message alerting the user to correct the problem

Handling Errors (3):

- **Detecting and Handling the Error**
  - We can basically handle such errors using nothing more than a selection statement

```
If there is no error
   proceed normally
otherwise
   deal with the error
```

  - We use a slightly different syntax than If statement
    - On Error GoTo lineX

Handling Errors (4):

- **Detecting and Handling the Error (cont.)**
  - Where, "lineX" is a name we choose that identifies where the error handling code begins
  - "On Error GoTo" are Visual Basic keywords

```
On Error GoTo ErrorHandler
  ...
  normal code (when no error)
  ...
Exit Sub
ErrorHandler:
  code to deal with error
```

Handling Errors (5):

- **Detecting and Handling the Error (cont.)**
  - ErrorHandler is the name denoting where the error handler code begins
  - Error handling code must be last part of the sub-program
    - Cannot have any "normal" statements after the error handling code
    - "Exit Sub" must be the last statement before the error handling code → causes sub-program to exit after "normal" code has been executed

Handling Errors (6):

- **The ErrObject Class**
  - Contains two useful properties
    - Number → a unique number to describe the error that was generated
    - Description → a description of the error

```
ErrorHandler:
  'user pressed Cancel - do nothing
  Dim messg As String
  messg = Str(Err.Number) & ": " & Err.Description
  MsgBox (messg)
```