

# Windows File I/O

## Files

- Collections of related data stored on external storage media and assigned names so that they can be accessed later
  - Entire collection is a file
  - A file is made up of records
    - One record for each entity stored in the file
    - Each record broken down into fields (data elements)

	Last Name	First Name	Phone
	-----		
Records →	Smith	John	777-1111
→	Jones	Mary	777-2222

↑  
Fields

## **File I/O Under Win32 API and MFC**

- Use Standard C Library File functions

*FILE \*fp; // a file pointer*

*fp = fopen("filename\_string", "mode\_string");*

*//Open file and retrieve a pointer to it*

*fread (buffer, size, number, fp);*

*// Read number of size elements into address pointed to by buffer*

*fwrite (buffer, size, number, fp);*

*// Write number of size elements from buffer to file*

*fseek(fp, offset, origin);*

*// Go offset bytes in open file, measured from origin*

*cur\_posn = ftell(fp);*

*// Retrieve current byte position in open file*

*fclose(fp); // Close file pointed to by fp*

- To be able to use these, we must #include <stdio.h>

- For details see:

– <http://www.cs.binghamton.edu/~reckert/360/class12.htm>

## **.NET File Class to Read/Write Files**

- File class in System.IO namespace
- Static methods ReadAllText( ) and WriteAllText( )
  - File.ReadAllText(string FilePath);
  - File.WriteAllText(string FilePath, string contents);
  - Can also use ReadAllBytes(...) and WriteAllBytes(...) for binary files
- Examples: FileSimpleWrite and FileSimpleRead
  - Use a textbox for user to enter file name, a multiline text box for the contents of the file to be written or read and a button to initiate the reading/writing

## Files and Streams in .NET

- .NET Framework handles data files using Streams
  - When a file is opened for reading or writing it becomes a stream
  - Stream: an object that transfer a series of bytes from one location to another
  - Read and write operations can be performed on a stream
  - Streams can be more than just open disk files
    - Network Streams: Data moving over a network connection is a stream
    - Memory Streams: Memory to memory transfers
- Most .NET File and Stream I/O support is implemented in System.IO namespace
- Any file-handling project should include the statement:  
using System.IO;

## FileStream File I/O Class

- FileStream
  - Most basic File I/O class: like C file pointer
  - Use to: open, read from, write to, and close files
  - To open or create a file, create an object of type FileStream
  - Some FileStream constructors:
    - FileStream(strFileName, FileMode);
    - Some FileMode properties: Create, Open, Append
    - FileStream(strFileName, FileMode, FileAccess);
    - Some FileAccess properties: Read, Write, ReadWrite
  - Some FileStream Methods:  
int ReadByte();  
int Read(byte[] abyBuffer, int iBufferOffset, int iCount);  
void WriteByte (byte byteValue);  
void Write(byte[] abyBuffer, int iBufferOffset, int iCount);  
long Seek(long offset, SeekOrigin origin);

## Problems with FileStream

- C# casting is not as flexible as C casting
  - FileStream Read() and Write() methods work only with byte arrays
    - For other data types the bytes in an array would have to be read and assembled into other basic data types
    - Very tedious
  - Difficult to use with text files because of different encodings (ASCII vs. Unicode)
- Better to work with StreamReader and StreamWriter classes for reading/writing text files
  - These create the file stream for you
- Or BinaryReader and BinaryWriter classes for reading/writing binary files
  - Files that are not text files

## Writing Data to a File Sequentially Using StreamWriter

- Declare and instantiate a new StreamWriter object
  - In constructor specify name of the data file

```
StreamWriter phoneStreamWriter;  
phoneStreamWriter = new StreamWriter("Phone.txt");
```
  - Opens the file for writing
- Use StreamWriter's WriteLine() method to copy a line of text (a string) from a buffer in memory to the file

```
phoneStreamWriter.WriteLine("777-1111");
```

  - Or use Write(char) to write single elements (characters)
- After all data is written call StreamWriter's Close() method

```
phoneStreamWriter.Close();
```

  - Transfers the data from the buffer to the file and releases system resources used by the stream
  - Usually done just before closing the window form

## FileStream-Write Example Program

- “Name” and “Phone” text boxes allow user to enter a name and a phone number
- A StreamWriter object will save names and phone number to a file
  - File name is hard-coded when StreamWriter object is instantiated
    - This occurs in the form’s constructor
    - Causes the file to be opened
- “Save” button: Click handler saves the current name and phone number at the end of a file
- “Exit” button: Click handler calls the StreamWriter’s Close() method & closes the form

## Reading Data from a File Using StreamReader

- Declare and instantiate a StreamReader class object
  - In constructor specify the file name

```
StreamReader phoneStreamReader;  
phoneStreamReader = new StreamReader("Phone.txt");
```

    - Opens the file for reading
- Use ReadLine() method to read next line (string)

```
string str = phoneStreamReader.ReadLine();
```

  - Use a loop to retrieve multiple lines
  - Use Peek() method to check for end of file
    - Looks at next element without reading it
    - Value returned after peeking beyond last item is -1
  - Use Read() to read the next character from the stream
  - Or numCharsRead = sr.Read(charArray, startPos, nChars);
- When done, close stream with StreamReader’s Close();

```
phoneStreamWriter.Close();
```

## **FileStream-Read Example** **Program**

- Form has “Name” and “Phone” label controls to display each name and phone number stored in a file
- When form is first loaded in Form1’s “Load” event handler:
  - try/catch block attempts to instantiate a StreamReader object (file may not exist)
    - File name to open is hard-coded in constructor
    - If successful, a call is made to a helper function DisplayRecord()
    - DisplayRecord() uses StreamReader’s Peek() method to see if there are more records to read
      - If so, its ReadLine() method reads the next name and number records from the file (same order as written) and stores them in the label controls
- “Next” Button: Click handler calls helper function DisplayRecord() to read and display next name & phone number from the file
- “Exit” button: Calls the StreamWriter’s Close() method and closes the form

## **Appending data to a File**

- As we’ve used StreamWriter, if the file exists at construction time, its contents will be destroyed
- Another constructor for StreamWriter:
  - StreamWriter(string strFileName, bool bAppend)
  - If bAppend is true, the file is not destroyed
    - Data can be appended to it

## Common File Dialog Boxes

- OpenFileDialog
  - Allows user to browse directories or enter a file name for a file to open
- SaveFileDialog
  - In same way, allows user to select or enter a file name to save
  - It just adds two new Boolean properties to OpenFileDialog:
    - CreatePrompt: true means if file specified by user doesn't exist, display a message box asking if user really wants to create the file
    - OverwritePrompt: true means a message box will prompt for confirmation if selected file already exists – to avoid undesired overwriting
  - If these properties are not needed, you can use OpenFileDialog for both opening and saving
- Both set its FileName property to the fully qualified file name the user selects from a list box or types into a text box
  - This can then be used to read from that file or to save data to it

## Some Important OpenFileDialog Class Properties

- |                    |                                                                                                                                                                                                   |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| • Name             | Name of OpenFileDialog object (VS Designer default: OpenFileDialog1)                                                                                                                              |
| • Title            | Title bar of dialog box                                                                                                                                                                           |
| • FileName         | Name of file selected/entered by user, including path                                                                                                                                             |
| • CheckFileExists  | Display error message if file does not exist; set to <u>false</u> for saving a file since you want to create a new file if it doesn't exist; leave <u>true</u> (default) to read an existing file |
| • CheckPathExists  | Same, but for the file path                                                                                                                                                                       |
| • Filter           | Filter file extensions to display in "Files of Type" combo box, e.g.: "All Files (*.*)*.*)"                                                                                                       |
| • InitialDirectory | Directory to display when dialog box opens; set to "Applications.StartupPath" to begin in same directory as application's executable                                                              |

## **File-Write-OpenFileDialog Example Program**

- Adds “File” | “Open” menu item to FileStream-Write example program
  - “Open” menu item starts an OpenFileDialog box for user to select or type in a file to write names and phone numbers to
    - Checks to see if file is already open first
      - If so, it closes it before starting the OpenFileDialog box and instantiating a StreamWriter object (opening the selected file)
  - “Save” button handler checks to see if file is open, and if so, saves current name and phone number
    - If not, puts up a message box to warn user, then calls the “Open” menu click handler so user can select the file to open
    - Also clears the text boxes and sets the focus to the “Name” text box
  - “Exit” menu item click handler closes the file if it’s open before closing the form

## **File-Read-OpenFileDialog Example Program**

- Adds “File” | “Open” menu item to FileStream-Read example program
  - Click “Open” menu item to start an OpenFileDialog box for user to select or type in a file to read names and phone numbers from
    - Checks to see if file is already open first
      - If so, it closes it before starting the OpenFileDialog box and instantiating a StreamReader object (opening the selected file)
  - “Next” button handler Peeks to make sure we’re not at the end of file, then reads the next name an number, and displays them in the label controls
    - Note that initially “Next” button’s enable property is set to false
      - Makes no sense for user to ask for the next item if file is not open
  - “Exit” menu item click handler closes the file if it’s open before closing the form



## BinaryWriter for integer data

- Instantiate a FileStream object and from it an associated BinaryWriter
- Then use its Write(data) method to write binary data to the stream

```
int[] x = new int[] { 3, 5, 7 };  
FileStream fs = new FileStream(writeFileName, FileMode.Create);  
BinaryWriter bw = new BinaryWriter(fs);  
for (int i=0; i<3; i++)  
    bw.Write(x[i]);  
fs.Close();
```

## BinaryReader for integer data

- Instantiate a FileStream object and from it an associated BinaryReader
- Then use its ReadInt32( ) method to read binary data from the stream

```
int[] x = new int[100];  
FileStream fs = new FileStream(readFileName, FileMode.Open);  
BinaryReader br = new BinaryReader(fs);  
For (int i=0; i<100; i++)  
    x[i] = br.ReadInt32();  
fs.Close();
```

## Other File Handling Static Methods

- All are member of the System.IO.File class
- Since they're static, don't need to instantiate a File object
- Some of them:
  - Determining whether a file exists
    - `bool File.Exists(string strFileName)`
  - Copying a file
    - `File.Copy(string strSrcFN, string strDestFN)`
  - Moving a file
    - `File.Move(string strSrcFN, string strDestFN)`
  - Deleting a file
    - `File.Delete(string strFileName)`

## Retrieving a File's Properties

- System.IO.File class has many other static methods
  - `GetCreationTime (string strFN)`
  - `GetLastAccessTime (string strFN)`
  - `GetLastWriteTime (string strFN)`
    - All return a DateTime object
  - `GetAttributes (string strFN)`
    - Returns a FileAttributes enumeration
    - Stores bit-packed Boolean Attribute Flags:
      - Archive, Directory, Hidden, Normal, ReadOnly, System, Temporary
        - » Do Boolean AND with appropriate mask to determine if a given attribute is true (bit is set)

## Manipulating Directories

- System.IO.Directory Class
- Use its static methods just like the File methods
- Some of its static methods:
  - Directory.CreateDirectory(string strDirName)
  - bool Directory.Exists(string strDirName)
  - Directory.Move(string strSrc, string strDest)
  - Directory.Delete(string strDirName)

## Serialization

- Saving and Retrieving complex objects instantiated from a class
- Serialization refers to converting an object's state to a stream of bits that can be saved
- Deserialization refers to reading the data back and recreating the object
- Declare a class as Serializable and use a formatter to serialize the object
  - BinaryFormatter formats data in binary form
  - SoapFormatter formats data in an XML format (Web)

## Using Serialization: Saving an Object

- Include Using statements

```
Using System.IO;
Using System.Runtime.Serialization;
Using System.Runtime.Serialization.Formatters.Binary;
```
- Declare the object's class as **Serializable**; for example:

```
[Serializable] public class Book { ... };
```
- In the form's code:
  - Instantiate the object; for example:

```
Book bookObject = new Book();
```
  - Declare a **FileStream** object that specifies the name of the file

```
FileStream bookStream = new FileStream("books.txt", FileMode.Create);
```
  - Declare a **BinaryFormatter** object; for example:

```
BinaryFormatter bookFormatter = new BinaryFormatter();
```
  - Use **BinaryFormatter** object's **Serialize()** method to save the object

```
bookFormatter.Serialize(bookStream, bookObject);
```
  - Close the **FileStream** object

```
bookStream.Close();
```

## Using Deserialization: Recreating an Object

- Read the object back with the **Deserialize()** method of the formatter
- Steps:
  - Declare a **BinaryFormatter** object; for example:

```
BinaryFormatter bookFormatter = new BinaryFormatter();
```
  - Create a **FileStream** object; for example:

```
FileStream bookStream = new FileStream("books.txt", FileMode.Open);
```
  - Use formatter's **Deserialize()** method, converting the input to the desired object type; for example, a **Book** object:

```
bookObject = (Book) bookFormatter.Deserialize(bookStream);
```
  - Use the object's fields/properties as desired
  - Close the **FileStream** object

```
bookStream.Close();
```

## File-Serializable-Book Example Program

- A “Book” class encapsulates information about a book
  - Public Properties to access private data fields:
    - Title
    - Quantity
    - Price
    - Total
  - Private Method calculates Total:
    - ComputeTotal()
      - Invoked in constructor
- Main form:
  - Text boxes to enter information about a book
  - A “Compute Total” button to calculate the total
  - File Menu:
    - Save Record
      - Uses serialization to save a book object’s data to disk
    - Retrieve Record
      - Uses deserialization to retrieve a book object’s data from disk



The screenshot shows a Windows application window titled "Form1". It features a "File" menu with two options: "Save Record" and "Retrieve Record". Below the menu, there are four text boxes for data entry: "Title" (with the text "Gone with the Wind"), "Quantity" (with the text "7"), "Price" (with the text "12.00"), and "Total" (with the text "84.00"). A "Compute Total" button is positioned at the bottom right of the form.