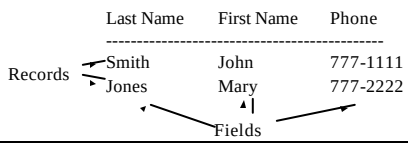
Visual Studio .Net File I/O

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)**



Files and Streams

- Visual Studio handles data files using Streams
  - When a file is opened for reading or writing it becomes a stream
  - Designed to 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
    - Data moving over a network is a stream
    - Memory streams can also be created
- 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
  - Use to open, read from, write to, and close files
  - To open or create a file, create an object of type FileStream
  - Some FileStream constuctors:

```
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 byValue);
Void Write(byte[] abyBuffer, int iBufferOffset, int iCount);
```

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
- Better to work with StreamReader and StreamWriter classes for reading/writing text files
- 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 text data (a string) to a buffer in memory

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

- 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");
```

- Use ReadLine() method to read next data item (string)

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

  - Use a loop to retrieve multiple records
  - 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
- When done, close stream with StreamReader's Close() method

```
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 FileStreamReader object
    - File name to open is hard-coded in constructor
    - If successful, a call is made to a helper function DisplayRecord()
    - DisplayRecord() uses FileStreamReader'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 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 dialog box will prompt for confirmation if selected file already exists – to avoid undesired overwriting
  - If these properties are not needed, use OpenFileDialog for both opening and saving
- Both return a 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 false for saving a file since you want to create a new file if it doesn't exist; leave true (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.: "Text Files (*.txt)|*.txt|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

Other System.IO File Handling Static Methods

- 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 methods
  - GetCreationTime (string strFN)
  - GetLastAccessTime (string strFN)
  - GetLastWriteTime (string strFN)
  - GetAttributes (string strFN)
    - Returns a FileAttributes enumeration
    - Stores bit-packed boolean Attribute Flags:
      - Archive, Directory, Hidden, Normal, ReadOnly, System, Temporary
        - Do boolean AND 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/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 stores data in binary form
  - SoapFormatter stores 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 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 includes the name of the file
    ```
    FileStream bookStream = new FileStream("books.txt", FileMode.Create);
    ```
  - Declare a formatter object; for example:
    ```
    BinaryFormatter bookFormatter new BinaryFormatter ();
    Use Formatter object's Serialize() method to save the object
    bookFormatter .Serialize(bookStream , bookObject);
    ```
  - Close the FileStream
    ```
    bookStream.Close();
    ```

Using Deserialization: Recreating an Object

- Read the object back with the Deserialize() method of the formatter
- Steps:
  - Declare a formatter 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:
    ```
    bookObject = (Book) bookFormatter.Deserialize (bookStream);
    ```
  - Use the object's fields/properties as desired
  - Close the stream
    ```
    bookStream.Close();
    ```

File-Serializable Book Example Program

- A "Book" class encapsulates information about a book
  - Properties:
    - Title
    - Quantity
    - Price
    - Total
  - Method:
    - ComputeTotal()
- 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 serialization to retrieve a book object's data from disk