File IO

CSIS 3540

Client Server Systems

Class 05

Topics

- Open a file
- Reading/Writing to a file
- Comma-separated-values file processing
- File dialogs

File Accessing

- A file object is an object that is associated with a specific file and provides a way for the program to work with that file
- The .NET Framework provide two classes to create file objects through the **System.IO** namespace
  - **StreamWriter**: for writing data to a text file
  - **StreamReader**: for reading data from a text file
- You need to write the following directives at the top of your program

```
Using System.IO;
```

Writing Data to a File

- Start with creating a StreamWriter object

  StreamWriter outputFile;

- Use one of the File methods to open the file to which you will be writing data. Sample File methods are:
  - File.CreateText
  - File.AppendText
- Or use StreamWriter constructor
  - outputFile = new StreamWriter(filename)

- Use the **Write** or **WriteLine** method to write items of data to the file
- Close the connection.

Sample Code

```
StreamWriter outputFile;
outputFile = File.CreateText(“courses.txt”);
outputFile = new StreamWriter(“courses.txt); // same as above just for demo
outputFile.WriteLine(“Introduction to Computer Science”);
outputFile.WriteLine(“English Composition”);
outputFile.Write(“Calculus I”);
outputFile.Close();
```

- The **WriteLine** method writes an item of data to a file and then writes a newline characters which specifies the end of a line
- The **Write** method writes an item to a file without a newline character

CreateText vs. AppendText

- The previous code uses the File.CreateText method for the following reasons:
  - It creates a text file with the name specified by the argument. If the file already exists, its contents are erased
  - It creates a StreamWriter object in memory, associated with the file
  - It returns a reference to the StreamWriter object
- When there is a need not to erase the contents of an existing file, use the AppendText method

```
StreamWriter outputFile;
outputFile = File.AppendText(“Names.txt”);
outputFile.WriteLine(“Lynn”);
outputFile.WriteLine(“Steve”);
outputFile.Close();
```

StreamReader and StreamWriter

- Easier to use the constructors for each
- Just give the file name

```
StreamWriter fileStreamWriter = new StreamWriter(fileName);
fileStreamWriter.WriteLine(“Hello”);

StreamReader fileStreamReader = new StreamReader(fileName);
string input = fileStreamReader.Readline();
```

Specifying the Location of an Output File

- If you want to open a file in a different location, you can specify a path as well as filename in the argument
- Be sure to prefix the string with the @ character
- Or use File Dialogs (discussed later)

```
StreamWriter outputFile;

outputFile = new StreamWriter(@”C:\Users\chris\Documents\Names.txt”);
```

Reading Data from a File

- Start with creating a StreamReader object

  StreamReader inputFile;

- Use the **File.OpenText** method to open the file to which you will be writing data

  inputFile = new StreamReader(“students.txt”);

- Use the **Read** or **ReadLine** method to write items of data to the file
  - StreamReader.ReadLine: Reads a line of characters from the current stream and returns the data as a string.
  - StreamReader.Read: Reads the next character or next set of characters from the input stream.

- Close the connection – don’t forget this!
  - inputFile.Close();

Reading a File with a Loop

- StreamReader objects have a Boolean property named EndOfStream that signals whether or not the end of file has been reached
- You can write a loop to detect the end of the file.

```
while (inputFile.EndOfStream == false) { }
```

- Or

```
while (!inputFile.EndOfStream) { }
```

Reading comma-separated values

- Read each line from the file
- Use the Split method
  - returns a string array of all comma-separated fields
  - Don’t forget to Trim()
- See OpenFileDialog example

```
while (inputFile.EndOfStream == false)
{
    StringBuilder sb = new StringBuilder();
    string input = inputFile.ReadLine();
    string[] fields = input.Split(',');
    for(int i = 0; i < fields.Length; i++)
    {
        sb.Append(fields[i].Trim()); // remove leading trailing white space
        if (i < (fields.Length - 1))
            sb.Append(" : ");
    }
    listBoxOutput.Items.Add(sb);

}
```

Reading CSV File using LINQ

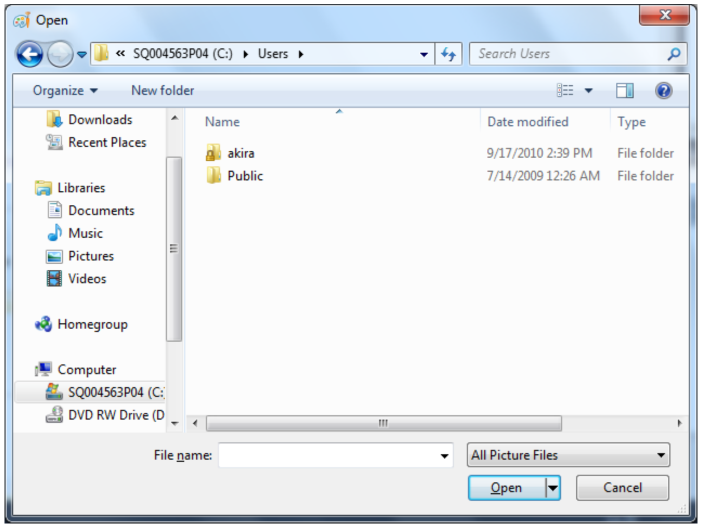
- Same as previous slide but …
  - Use Select to perform a function on each field returned from Split().
  - Select returns IEnumerable, so change it to an Array.
- Or use LINQ syntax

```
string[] fields = input.Split(',').Select(f => f.Trim()).ToArray();

string[] fields2 =
                   (from field in input.Split(',')
                   select field.Trim()).ToArray();
```

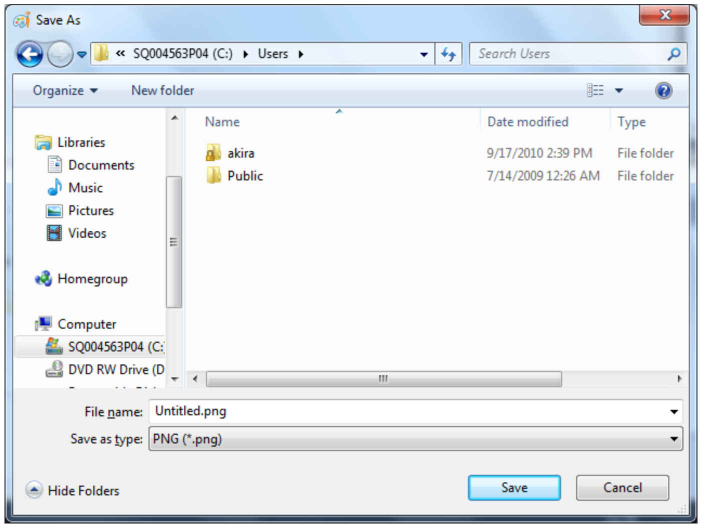
The OpenFileDialog and SaveFileDialog Controls

- The **OpenFileDialog** and **SaveDialog** controls allow your application to display standard Windows dialog boxes for opening and saving files
- Unlike Label, Button, and TextBox, they are invisible controls
- The OpenFileDialog control displays a standard Windows *Open* dialog box.
- The SaveDialog control displays a standard Windows *Save As* dialog box



Name	Date modified	Type
akira	9/17/2010 2:39 PM	File folder
Public	7/14/2009 12:26 AM	File folder

File name:
All Picture Files



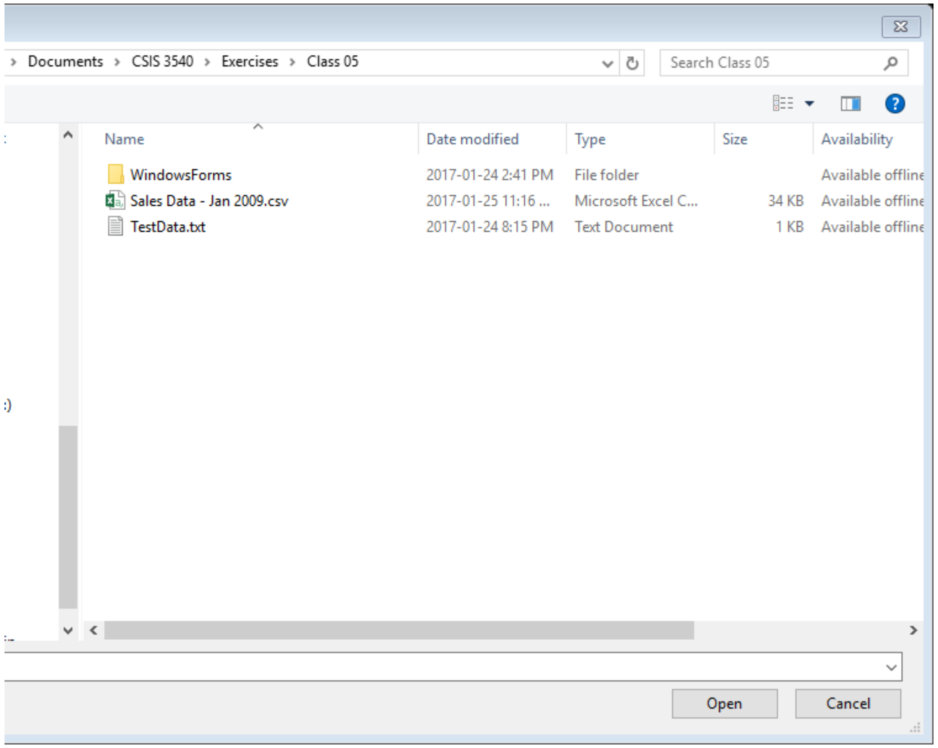
Name	Date modified	Type
akira	9/17/2010 2:39 PM	File folder
Public	7/14/2009 12:26 AM	File folder

File name: Untitled.png
Save as type: PNG (*.png)

OpenFileDialog

- No need to drag/drop control
  - Will only make things messy
- Create OpenFileDialog object

```
OpenFileDialog openFileDialogCSV = new OpenFileDialog();
```


› Documents › CSIS 3540 › Exercises › Class 05

Search Class 05

Name	Date modified	Type	Size	Availability
WindowsForms	2017-01-24 2:41 PM	File folder		Available offline
Sales Data - Jan 2009.csv	2017-01-25 11:16 ...	Microsoft Excel C...	34 KB	Available offline
TestData.txt	2017-01-24 8:15 PM	Text Document	1 KB	Available offline

Detecting the User's Selection

- The **showDialog** method returns a value that indicates which button the user clicks to dismiss the dialog box
  - If the user clicked the Open button, the value **DialogResult.OK** is returned
  - If the user clicked the Cancel button, the value **DialogResult.Cancel** is returned
- Note that OpenFileDialog inherits from CommonDialog, which is where ShowDialog resides.
- See OpenFileDialogExample

```
OpenFileDialog openFileDialogCSV = new OpenFileDialog
{
    // we start up in the debug directory, go two levels up to get to the main project area
    // note need to use Path.GetFullPath() as InitialDirectory does not like relative directories
    InitialDirectory = Path.GetFullPath(Application.StartupPath + "\\..\\.."),
};

StreamReader inputFile;

// open the filedialog, get a name, and open the file

if (openFileDialogCSV.ShowDialog() == DialogResult.OK)
{
    // could use new StreamReader() here as well
    inputFile = File.OpenText(openFileDialogCSV.FileName);
}
else return; // failure!!
```

The Filename and InitialDirectory Property

- When the user selects a file with the Open dialog box, the file's path and filename are stored in the control's **Filename** property
- You can specify a directory to be initially displayed with the **InitialDirectory** property.

```
OpenFileDialog openFileDialogCSV = new OpenFileDialog
{
    // we start up in the debug directory, go two levels up to get to the main project area
    // note need to use Path.GetFullPath() as InitialDirectory does not like relative directories
    InitialDirectory = Path.GetFullPath(Application.StartupPath + "\\..\\.."),
};

StreamReader inputFile;

// open the filedialog, get a name, and open the file

if (openFileDialogCSV.ShowDialog() == DialogResult.OK)
{
    // could use new StreamReader() here as well
    inputFile = File.OpenText(openFileDialogCSV.FileName);
}
else return; // failure!!
```

Displaying a Save As Dialog Box

- Create SaveFileDialog
  - Set InitialDirectory if desired
  - Note use of Path.GetFullPath()
  - Set Filter
- Use the ShowDialog method, and check to see if DialogResult.OK is returned.
- Open the file for writing
  - Note use of StreamWrite, could have used File.CreateText()
- See OpenFileDialogExample

```
SaveFileDialog openFileToSave = new SaveFileDialog
{
    InitialDirectory = Path.GetFullPath(Application.StartupPath + "\\..\\.."),
    Filter = "Text Files|*.txt"  // only allow .txt files for output
};
StreamWriter outputFile;

if (openFileToSave.ShowDialog() == DialogResult.OK)
    outputFile = new StreamWriter(openFileToSave.FileName);
else return;
```