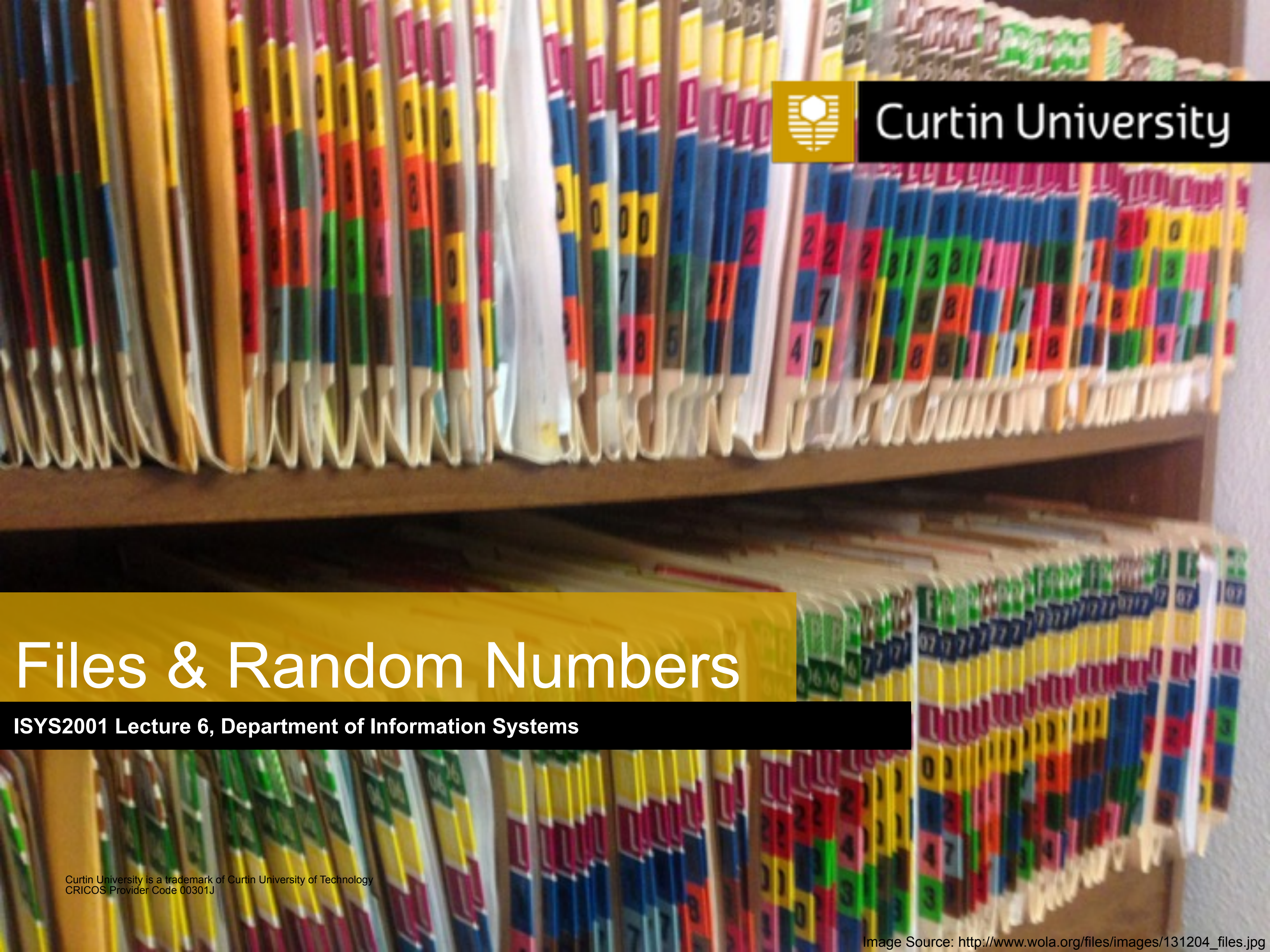


Curtin University

Files & Random Numbers

ISYS2001 Lecture 6, Department of Information Systems

Image Source: http://www.wola.org/files/images/131204_files.jpg

Last Week

- Add item to ListBoxes
- Clear items from ListBoxes
- Understand the while Loop
- Use increment and decrement operators
- Understand the for Loop
- Understand the do-while Loop

Today you will be able to…

- Use files for data storage
- Use the OpenFileDialog Control
- Use the SaveFileDialog Control
- Generate Random Numbers
- Create a Load event handler

Using File for Data Storage

- When a program needs to save data for later use, it writes the data in a file
- There are always three steps:

  Open the file: create a connection between the file and the program

  Process the file: either write to or read from the file

  Close the file: disconnect the file and the program
- In general, there are two types of files:

  Text file: contains data that has been encoded as test using scheme such as Unicode

  Binary file: contains data that has not been converted to text. You cannot view the contents of binary files with a text editor.
- This chapter only works with text files

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
```

- 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.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();
```

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
- StreamWriter outputFile;

  outputFile = File.CreateText(@”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 = FileOpenText("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

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.

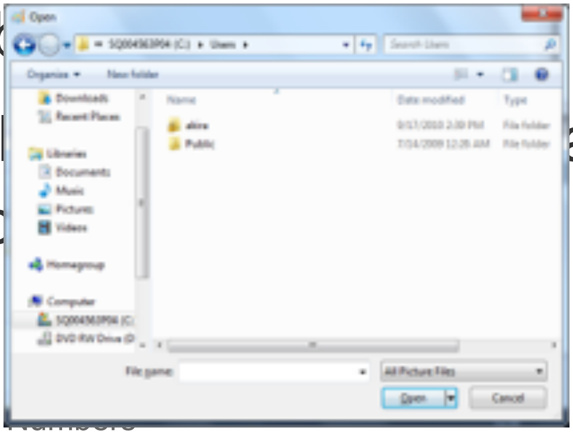
```
while (inputFile.EndOfStream == false) { }
```

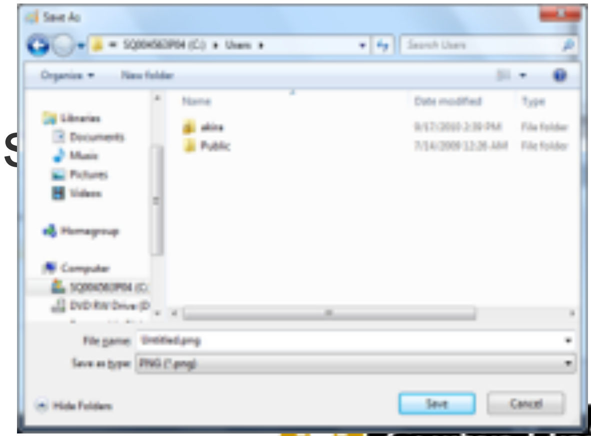
- Or

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

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 b
- The SaveDial a s Save As dialo



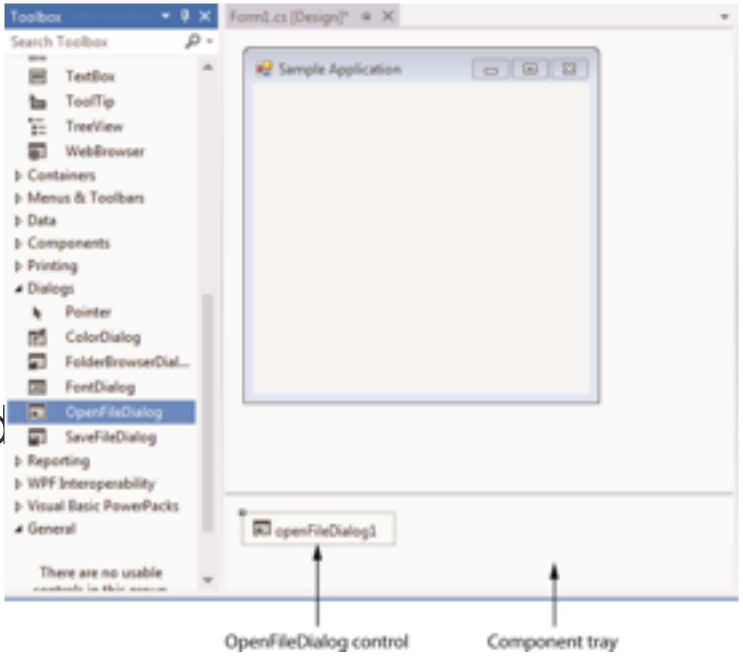


File name: Untitled.png
Save as type: PNG (*.png)

Displaying an Open Box

- When adding an OpenFileDialog control to the form, it does not appear on the form, but in an area at the bottom of the Designer called the component tray
- In code, you can display an Open
- Dialog box by calling the ShowDialog
- method

```
private void button1_Click(object send
{
    openFileDialog1.ShowDialog();
}
```



Curtin University

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

  The following is an example that calls the ShowDialog method to determine the user’s choice:

```
if (openFile.ShowDialog() == DialogResult.OK) { }
else if (openFile.ShowDialog() == DialogResult.Cancel) { }
else { }
```

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
- The following is an example of how to open the selected file:

```
if (openFile.ShowDialog() == DialogResult.OK)
{
  inputFile = File.OpenText(openFile.Filename);
}
else { }
```

- You can specify a directory to be initially displayed with the InitialDirectory property. For example,

```
openFile.InitialDirectory = “C:\Data”;
```

Displaying a Save As Dialog Box

- Use the following to call the SaveFileDialog control's ShowDialog method

```
saveFile.ShowDialog();
```

- Use the following to detect the user's choice

```
if (saveFile.ShowDialog() == DialogResult.OK) { }
```

- Use the following to open the selected file

```
if (saveFile.ShowDialog() == DialogResult.OK)
{
  outputFile = File.CreateText(openFile.Filename);
}
```

5.8 Random Numbers

- The .NET Framework provides the Random class to generate random numbers.
- To create an object, use:

```
Random rand = new Random();
```

- Two commonly used methods to generate random numbers are:

  Next: randomly create an integer

  NextDouble: randomly create a floating-point number from 0.0 to 1.0

- Examples,

```
rand.Next();
rand.NextDouble();
```

Syntax of Random.Next Method

- Random.Next generates a random number whose value ranges from zero to 2,147,483,647
- It also allow you to generate a random number whose value ranges from zero to some other positive number. The syntax is:

```
    Random.Next(max+1);
```

- For example, to create a random number from 0 to 99, use:

```
    rand.Next(10);
```

5.9 The Load Event

- When running an application, the application's form is loaded into memory and an event known as Load takes place
- To create a Load event handler, simply double click the form in the Designer
- An empty Load event handler looks like:

```
    private void Form1_Load(object sender, EventArgs e) { }
```

- Any code you write inside the Load event will execute when the form is launched. For example,

```
        private void Form1_Load(object sender, EventArgs e)
        {
          MessageBox.Show("Prepare to see the form!");
        }
```

Can You…

- Use files for data storage
- Use the OpenFileDialog Control
- Use the SaveFileDialog Control
- Generate Random Numbers
- Create a Load event handler

Curtin University

Next Week

- Introduction to Methods
- void Methods
- Passing Arguments to Methods
- Passing Arguments by Reference
- Value-Returning Methods