Programming in C#

Working with text files

Agenda:

- Reading and writing text files
- StreamReader/StreamWriter
- FileDialogs

Farid Naisan, Senior Lecturer, Malmö University, farid.naisan@mau.se

Save data to retrieve a later time

- When we are running a program, all data is kept in the computer's memory (RAM). When the program closes, all data is lost.
- To save data more permanently, it has to be saved to secondary memory such as the hard disk, DVD or flash memory.
- Data saved becomes data source (data storage). Data can be saved in a variety of forms like text files, xml-files, databases, binary files and streams.
- Both files and streams refers to the transfer of data either to or from a storage medium.
- Every programming language provides support to help programmers accomplish saving and reading of data in various forms.

Three major steps in saving and retrieving data

- Save data: values of instance variables → file
  - **Open** an existing file or create a new one.
    - If a file exists, you can decide whether the contents can be cleared or data can be appended. Most often the file is cleared.
  - **Save** data
    - Get the value of an objects instance variables and transfer them to the file.
  - **Close** the file (as soon as you are done with saving data)
- Read data: values in file →instance variables
  - **Open** an existing file (file must exist)
  - **Read** data
    - Get the values from the file and save them to instance variables of an object
  - **Close** the file (as soon as you are done with it).

System.IO

- When working with files, you work with directory paths, disk storage, and file and directory names.
- The namespace **System.IO** (IO stands for Input Output) of the .NET Framework contains many types that help you manipulate physical files and directories as well as different type of streams.
- Some of the common classes are:
  - File
  - BinaryWriter, BinaryReader
  - StreamWriter, StreamReader
  - StringWriter, StringReader

Files and directories

- **System.IO** namespace has many classes that can be used to work with files and directories. These classes can be used to create, delete, copy or move files and directories. Here are some commonly used file and directory classes:
- **File** - provides static methods for creating, copying, deleting, moving, and opening files, and helps create a FileStream object.
- **FileInfo** - provides instance methods for creating, copying, deleting, moving, and opening files, and helps create a FileStream object.
- **Directory** - provides static methods for creating, moving, and enumerating through directories and subdirectories.
- **DirectoryInfo** - provides instance methods for creating, moving, and enumerating through directories and subdirectories.
- **Path** - provides methods and properties for processing directory strings in a cross-platform manner.

The File class

- The class File Provides static methods for the creation, copying, deletion, moving, and opening of a single file, and aids in the creation of FileStream objects.
- Use the File class for typical operations such as copying, moving, renaming, creating, opening, deleting, and appending to a single file at a time.
- You can also use the File class to get and set file attributes or DateTime information related to the creation, access, and writing of a file.
- Do not use File for performing operations on multiple files. It is not recommended due to performance issues. Use the classes Directory and DirectoryInfo in such cases.

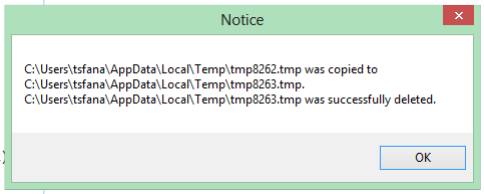
```
string path = @"c:\temp\Test.txt";
if (!File.Exists(path))
{
    // Create a file to write to.
    using (StreamWriter sw = File.CreateText(path))
    {
        sw.WriteLine("Data saved by xx");
        sw.WriteLine("Version 2.0");
    }
}
```

```
using (StreamReader sr = File.OpenText(path))
{
    string s;
    while ((s = sr.ReadLine()) != null)
    {
        //process the
    }
}
```

FileInfo Example

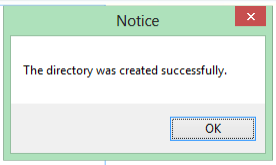
```
public  string TestFileInfo()
{
    string message = null;
    //Create a tempory file in the current user's temp
    string path = Path.GetTempFileName();
    FileInfo fi1 = new FileInfo(path);

    //Create a file to write to.
    using (StreamWriter sw = fi1.CreateText())
    {
        sw.WriteLine("Hello and welcome");
    }

    try
    {
        string path2 = Path.GetTempFileName();
        FileInfo fi2 = new FileInfo(path2);

        //Ensure that the target does not exist.
        fi2.Delete();

        //Copy the file.
        fi1.CopyTo(path2);
         message = String.Format("{0} was copied to {1}.", path, path2);
        //Delete the newly created file.
        fi2.Delete();
        message += Environment.NewLine;
        message += String.Format("{0} was successfully deleted.", path2);
    }
    catch (Exception e)
    {
        message =  string.Format("The process failed: {0}", e.ToString());
    }
    return message;
}
```

```
private void TestFileInfo()
{
    FileIOTest obj = new FileIOTest();
    MessageBox.Show(obj.TestFileInfo(), "Notice");
}
```



DirectoryInfo example

```
public string TestDirectoryInfo()
{
    string message = null;

    // Specify the directories you want to manipulate.
    DirectoryInfo di = new DirectoryInfo(@"c:\MyDir");
    try
    {
        // Determine whether the directory exists.
        if (di.Exists)
        {
            // Indicate that the directory already exists.
            return"That path exists already.";
        }

        // Try to create the directory.
        di.Create();
        message = "The directory was created successfully.";

    }
    catch (Exception e)
    {
        return String.Format("The process failed: {0}", e.ToString());
    }
    finally { }
    return message;
}
```



Reading and writing of a text file

- Text files are useful in many cases. You can for example use these to import/export data between different applications.
- Text files take more disk space, can be slower but have the advantage of being readable (and changeable) using an ordinary text editor.
- There are several types in the .NET class library that can be used for working with text files, but the StreamWriter, StreamReader classes are frequently used to write to a text file and read from a text file respectively.
- The StreamWriter and StreamReader classes are useful for writing and reading of character based data (e.g., strings).
- The System.IO namespace needs to be included in your class file.

StreamWriter/Reader Members

- The StreamWriter class has useful members
  - Write()
  - WriteLine()
  - NewLine
  - Close()
  - Flush()
- Some members of the StreamReader class
  - Read()
  - ReadLine()
  - ReadBlock()
  - Peek()
  - ReadToEnd()

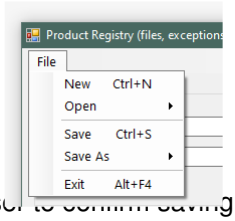
StreamWriter/Reader usage

- To write to a text file, you can use a StreamWriter object.
- The methods Write and WriteLine work in the same way as System.Console.
- To read from a text file, use a StreamReader object.
- Both Read and ReadLine work in the way as System.Console.

```
public void TestWrite(string fileName)
{
    StreamWriter writer = new StreamWriter(fileName);
    writer.WriteLine("Some text");
    writer.Close();
}

public void TestRead(string fileName)
{
    StreamReader reader = new StreamReader(fileName);
    string strLine = reader.ReadLine();
}
```

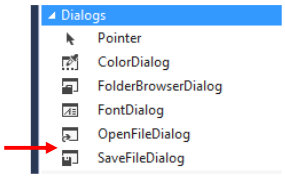

File handling - GUI

- File menus look much a like in most application.
  - It is usually a first menu item.
- The submenus also have a standard look and order.
- **New** – the application is to return to its start status. All data are initialized, and input/output components on the GUI are cleared.
  - If the current data is not saved, a user-friendly application ask the user to confirm saving or not saving the data before proceeding.
- **Open** – show open dialog box to open a file that has data saved by the application. Here also if unsaved data is to be saved first.
- **Save** – the program checks if the user has opened a file (check if the filename is not null). In that case, it opens the file and saves data. If not, it calls the method Save As (from code).
- **Save As** – show the save dialog box so the user can select a folder and a filename and extension. Using the file name, save data to the file.



File Dialogs

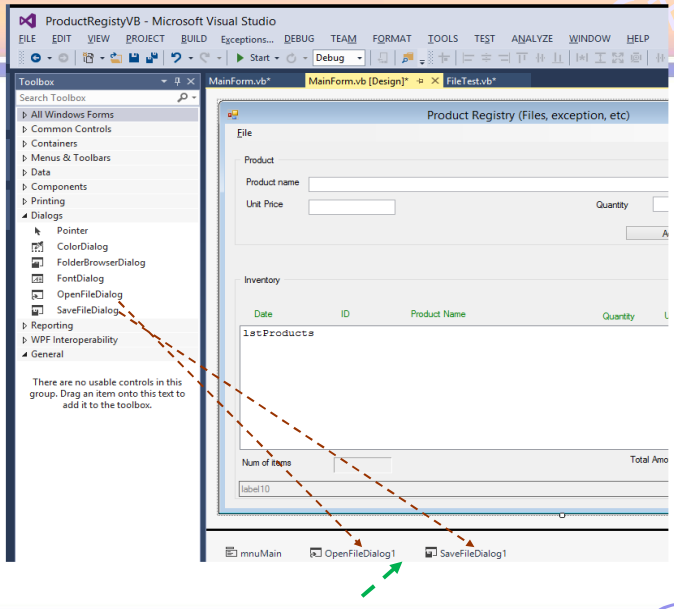
- To allow the user select a file, a directory and a drive for saving or reading, you make use of the controls OpenFileDialog and SaveFileDialog (VS's toolbox).



- These two classes have very useful methods and properties to make your application very user-friendly when opening a data file.
- Remember that the main result of the dialogs is to get a name and location of a file, in order words the file path.
- Reading or writing data are to be done by you as a next step.

FileDialogs

- In the Toolbox in Visual Studio, you find two components OpenFileDialog and SaveFileDialog that do not have a visual part.
- These two are very useful to let users select a device, a directory and a file to open for reading or create/open for writing data.
- The components have all functionalities needed to interact with the user for file management, as any other advanced application.



SaveFileDialog

- **SaveToFile** is a method which saves data to the opened file (not shown here due to space)

```
private void mnuFileSaveAs_Click(object sender, System.EventArgs e)
{
    //Show save-dialogbox
    if (saveFileDialog1.ShowDialog() == DialogResult.OK)
    {
        fileName = saveFileDialog1.FileName;  //important
        SaveToFile();
    }
}

private void mnuFileSave_Click(object sender, System.EventArgs e)
{
    if (fileName == String.Empty)  //no file name ==> Save As
        mnuFileSaveAs_Click(sender, e);
    else
        SaveToFile();

}
```


OpenFileDialog

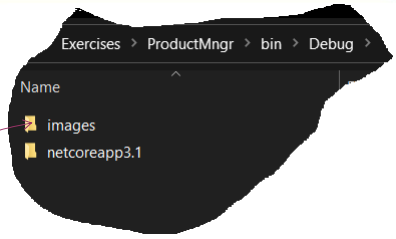
- ReadFile is a method that read data to the opened file (not shown here due to space).

```
private void mnuFileOpen_Click(object sender, System.EventArgs e)
{
    AskUserIfSaveDataToFile(sender, e);  //Save current data?
    //Show open-dialogbox
    if (openFileDialog1.ShowDialog() == DialogResult.OK)
    {
        fileName = openFileDialog1.FileName;
        string msg = ReadFile();

        if (msg != String.Empty)
            MessageBox.Show(msg);
        else
            UpdateListBox();
    }
}
```

Path relative to the application folder

- If you want to open a file for reading/writing as a part of your application, it is always good to select a path relative to the application folder.

```
private void btnLoadPic_Click(object sender, EventArgs e)
{
   string fileName = Application.StartupPath + @"..\images\MyHappyF.png";
    pictureBox1.Image = Image.FromFile(fileName);
}
```



- The file MyHappyF.png is located the …bin\images folder. The EXE file is located in the Debug folder. The path in the code sets the current directory one step backward by the symbol "..\".
- Application.StartupPath is a string that contains the path to your EXE-file, which at design-time while developing the application should the bin\Debug folder.
- The symbol '@' allows you to make slashes as part of the text (not Escape sequence char).

Summary

- Import **System.IO** (using System.IO;) in class files where your working with text files.
- Use **StreamWriter** and **StreamReader** to write data to and from a text file.
- Use the **FileInfo** and **DirectoryInfo** classes when working with files and directories.
- **OpenFileDialog** and **SaveFileDialog** are very good tools for letting the user select a file and a path.
- You should always use try-catch statements (as in the examples) to avoid program crash under runtime when something goes wrong.
- Use always StreamReader/StreamWriter as the reader and writer objects of the same file, i.e. data saved with StreamWriter should be read by StreamReader, and not by any other reader type
- Read a file exactly in same order the information was written to the file.
- It is always good to put some marks such as application's name, version number or other such codes so you can control if a file is saved by the application when reading it. Using version no, you can take care of compatibility of your versions.