

Programming with Event Driven in C#

Working with Files and Database Programming



Chalew Tesfaye

Department of Computer Science

Debre Berhan University

2017

Objectives

- After the end this lesson the student will be able to
 - ✓ Understand file system
 - ✓ Understand database system
 - ✓ Write a program that uses file system as a backend
 - ✓ Write a program uses database system as a backend

Lesson Outline

- File system and IO stream
- Read/write a file
- Relational Database system
- ADO.Net programming
- Connect a project to database
- Using database objects in application program

File System

Accessing flat file system

Saving and Retrieving Your Data with programs

- When developing an application you may need to save data in a file on disk and then read that data whenever it's need
- The classes in the namespace **System.IO** provides various methods for
 - ✓ working with Files and
 - ✓ managing directories, files and paths.
- In C#, to perform a file I/O operation,
 - ✓ you can use the same familiar basics of
 - > creating,
 - > opening,
 - > closing,
 - > reading, and
 - > writing using .NET Framework equivalent classes and methods.

Class for managing Directories, files and paths

- **Directory:**
 - ✓ Methods of directory class in CSharp exposes methods to create, delete, move etc. operations to directories and subdirectories.
- **File:**
 - ✓ methods of the file class to copy, delete, or move a file or to determine if a file exists.
- Both **Directory** and **File** methods are static methods,
 - ✓ we can call them directly from the class.
- **System.IO** class used to work with drivers and directories
 - ✓ **Directory:**
 - > used to create, edit, delete, or get information on directories/folders
 - ✓ **File:**
 - > used to create, edit, delete or get information on files
 - ✓ **Path:**
 - > used to get path information

Common methods of Directory class

- **Exists(path):**
 - ✓ returns a Boolean value indicating whether a directory exists
- **CreateDirectory(path):**
 - ✓ creates the directories in a specified path
- **Delete(path):**
 - ✓ deletes the directory at the specified path.
 - ✓ The directory must be empty
- **Delete(path, recursive):**
 - ✓ deletes the directory at the specified path.
 - ✓ If **true** specified for the recursive argument, any subdirectories and files in the directory are deleted.
 - ✓ If **false** is specified the directory must be empty.

Common methods of the File class

- **Exists(path):**
 - ✓ returns a Boolean value indicating whether a file exists.
- **Delete(path):**
 - ✓ delete files
- **Copy(source, dest):**
 - ✓ copies a file from source to destination
- **Move(source, dest):**
 - ✓ moves file from source to destination
- Statement that simplifies references to System.IO classes

Using System.IO;

Common methods of Path class

- **GetDirectoryName(path)**
 - ✓ returns the directory information for the specified path
- **GetFileName(path)**
 - ✓ Returns file name and extension of the specified path string
- **GetExtension(path)**
 - ✓ Returns the extensions of the specified path string
- **GetFullPath(path)**
 - ✓ Returns the absolute path
- **GetFileNameWithoutExtension**
 - ✓ Returns the file name of the specified path without extension

Directory methods > Example

```
private void btnSave_Click(object sender, EventArgs e)
{
    if (Directory.Exists("c:\\testDir1"))
    {
        //shows message if testdir1 exist
        MessageBox.Show ("Directory 'testDir' Exist ");
    }
    else
    {
        //create the directory testDir1
        Directory.CreateDirectory("c:\\testDir1");
        MessageBox.Show("testDir1 created ! ");
        //create the directory testDir2
        Directory.CreateDirectory("c:\\testDir1\\testDir2");
        MessageBox.Show("testDir2 created ! ");
        //move the directory testDir2 as testDir in c:\
        Directory.Move("c:\\testDir1\\testDir2", "c:\\testDir");
        MessageBox.Show("testDir2 moved ");
        //delete the directory testDir1
        Directory.Delete("c:\\testDir1");
        MessageBox.Show("testDir1 deleted ");
    }
}
```

File Methods > Example

```
private void btnSave_Click(object sender, EventArgs e)
{
    if (File.Exists("c:\\testFile.txt"))
    {
        //shows message if testFile exist
        MessageBox.Show ("File 'testFile' Exist ");
    }
    else
    {
        //create the file testFile.txt
        File.Create("c:\\testFile.txt");
        MessageBox.Show("File 'testFile' created ");
    }
}
```

Path Methods > Example

```
private void btnSave_Click(object sender, EventArgs e)
{
    string path = @"C:\SIS\course.txt";
    string dir = Path.GetDirectoryName(path);
    MessageBox.Show ("Directory :" + dir);
    string name = Path.GetFileName(path);
    MessageBox.Show ("File Name:" + name);
    string ext = Path.GetExtension(path);
    MessageBox.Show ("File Extension :" + ext);
}
```

Text and Binary files

- In a text file, all of the data is stored as text characters or string.
- Often, the fields in this of file are separated by delimiters like tabs or pipe characters and the records are separated end of line characters.
- In contrast, the data in a binary file can include text characters as well as data types.
 - ✓ Because of that the data isn't always displayed properly within a text editor.
- To handle I/O operations with text and binary files, the .NET Framework uses streams.
- You can think of streams as the flow of data from one location to another.
- For example,
 - ✓ an output stream can flow from the internal memory of an application to a disk file, and
 - ✓ an input stream can flow from a disk file to internal memory.

Text and Binary files

- When you work with a text file, you can use a text stream,
- when work with a binary file, use a binary stream.
- Since you can store all of the built-in numeric data types in a binary file,
 - ✓ this type of file is more efficient for applications that work with numeric data.
- In contrast, the numeric data in a text file is stored as characters,
 - ✓ so each field must be converted to a numeric data type before it can be used in arithmetic operations.
- When you save a text or binary file,
 - ✓ you can use any extension you want for the file name.

System.IO classes

- System.IO classes used to work with files and streams
 - ✓ **FileStream:**
 - > provides access to input and output files
 - ✓ **StreamReader:**
 - > to read a stream of characters
 - ✓ **StreamWriter:**
 - > to write a stream of characters
 - ✓ **BinaryReader:**
 - > to read a stream of binary data
 - ✓ **BinaryWriter:**
 - > to write a stream of binary data
- A stream is the flow of data from one location to another.
- To write data, use an output stream.
- To read data, use input stream.
- A single stream can also be used for both input and output operation.
- To read and write text files, use text stream.
- To read and write binary files, Use binary streams.

FileStream Class

- To create a stream that connects to a file,
 - > use the `FileStream` class.
- The `FileStream` Class represents a File in the Computer
- Use the `FileStream` class
 - > to read from, write to, open, and close files on a file system,
 - > to manipulate other file related operating system handles including pipes, standard input, and standard output.
- We operate on File using `FileMode` in `FileStream` Class
- The syntax for creating a `FileStram` object

`new FileStream(path, mode[,access[,share]]);`

FileStream ...

- To code the mode, access and share arguments, use the
 - ✓ `FileMode`,
 - ✓ `FileAccess`, and
 - ✓ `FileShare` enumerations.
- **Members in `FileModes` enumerations**
 - ✓ **`Append` :**
 - > Open and append to a file if the file does not exist , it create a new file
 - ✓ **`Create` :**
 - > Create a new file , if the file exist it will append to it
 - ✓ **`CreateNew` :**
 - > Create a new File , if the file exist , it throws exception
 - ✓ **`Open` :**
 - > Open an existing file, if the file doesn't exist, it throws exception
 - ✓ **`OpenOrCreate`:**
 - > opens a file if it exists, or create a new file if it doesn't exist
 - ✓ **`Truncate`:**
 - > opens an existing file and truncates it so its size is zero bytes

FileStream ...

➤ Members in the **FileAccess** enumeration

- ✓ **Read:**
 - > data can be read from the file but not write
- ✓ **ReadWrite:**
 - > data can be read from and written to the file. This is the default
- ✓ **Write:**
 - > data can be written to the file but not read from it

➤ Members in the **FileShare** enumeration

- ✓ **None:**
 - > the file cannot be opened by other application
- ✓ **Read:**
 - > allows other application to open the file for reading only.
 - > This is the default
- ✓ **ReadWrite:**
 - > allows other application to open the file for both reading and writing
- ✓ **Write:**
 - > allows other applications to open the file for writing only

FileStream ...

➤ Common methods of the **FileStream**

✓ **Close():**

- > Closes the file stream and releases any resources associated with it

➤ Code that create a **FileStream** object for writing

- ✓ **FileStream** path = @"C:\csharp\files\course.txt";
- ✓ **FileStream** fileWrite = new **FileStream**(path, **FileMode**.Create, **FileAccess**.Write);

➤ Code that create a **FileStream** object for reading

- ✓ **FileStream** path = @"C:\csharp\files\course.txt";
- ✓ **FileStream** fileRead = new **FileStream**(path, **FileMode**.Open, **FileAccess**.Read);

➤ **Note:**

- ✓ Operating system level permissions may limit which file access and file share options you can use.

SreamWriter

- **Common methods of the `SreamWriter` class**
 - ✓ `Write(data):`
 - > writes the data to the output stream
 - ✓ `WriteLine(data):`
 - > writes the data to the output stream and appends a line terminator
 - ✓ `Close():`
 - > closes the `StreamWriter` object and the associated `FileStream` object

StreamWriter > Example

```
StreamWriter fwriter = new StreamWriter(new FileStream(path,  
FileMode.Create, FileAccess.Write));
```

```
foreach (Course crs in Courses)
```

```
{
```

```
    fWriter.Write(crs.Code + " |");
```

```
    fWrite.Write(crs.Title + " |");
```

```
    fWriter.Write(crs.Credit);
```

```
}
```

```
fWrite.Close();
```

StreamReader

➤ Common methods of the **StreamReader** class

✓ **Peek()**

- > returns the next available character in the input stream without advancing to the next position.
- > If no more characters are available, this method returns -1

✓ **Read()**

- > reads the next character from the input stream

✓ **ReadLine()**

- > reads the next line character from the input stream and returns it as string

✓ **ReadToEnd()**

- > reads the data from the current position in the input stream to the end of the stream and returns it as a string.
- > This is typically used to read the contents of an entire file

✓ **Close()**

- > closes both the StreamReader object and the associated FileStream object

StreamReader > Example

```
StreamReader fReader = new FileStream(path, FileMode.OpenOrCreate, FileAccess.Read));
List<Course> courses = new List<Course>();
While(fReader.Peek() != -1)
{
    String row = fReader.ReadLine();
    String[] col=row.split('|');
    Course crs = new Course();
    crs.Code = col[0];
    crs.Title = col[1];
    crs.Credit = col[2];
    courses.Add(crs);
}
fReader.Close();
```

Work with binary file

- The basic syntax for creating a **BinaryWriter** object
`BinaryWriter bwrite = new BinaryWriter(stream);`
- **Common methods of the **BinaryWriter** class**
 - ✓ `Write(data)`
 - > writes the specified data to the output stream
 - ✓ `Close()`
 - > closes the BinaryWriter objects and associated FileStream objects

BinaryWriter > Example

```
BinaryWriter bWrite = new BinaryWriter(new FileStream(path,  
    FileMode.Create, FileAccess.write));  
foreach (Course crs in Courses)  
{  
    bWrite.Write(crs.Code);  
    bWrite.Write(crs.Title);  
    bWrite.Write(crs.Credit);  
}  
bWrite.Close();
```

BinaryReader class

- Common methods of the **BinaryReader** class
 - ✓ **PeekChar()**
 - > returns the next available in the input stream without advancing to the next position.
 - > If no more characters are available, this method returns -1
 - ✓ **Read()**
 - > returns the next available character from the input stream and advances to the next position in the file
 - ✓ **ReadBoolean()**
 - > returns a Boolean value from the input stream and advance the current position of the stream by one byte.
 - ✓ **ReadByte()**
 - > returns a byte from the input stream and advance the current position of the stream accordingly.

BinaryReader ...

- ✓ **ReadChar()**
 - > returns a character from the input stream and advance the current position of the stream accordingly
- ✓ **ReadDecimal()**
 - > returns a decimal value from the input stream and advances the current position of the stream by 16 byte
- ✓ **ReadInt32()**
 - > returns a 4-byte signed integer from the input stream and advance the current position the stream by 4 byte
- ✓ **ReadString()**
 - > returns a string from the input stream and advance the current position of the stream by the number of character
- ✓ **Close()**
 - > Closes the BinaryRead object and the associated FileStream object

BinaryReader > Example

```
BinaryReader bread = new BinaryReader( new FileStream(path,
    FileMode.OpenOrCreate, FileAccess.Read));
List<Course> courses = new List<Course>();
while(bRead.PeekChar() !=1)
{
    Course crs = new Course();
    crs.Code = bRead.ReadString();
    crs.Title = bRead.ReadString();
    crs.Credit = bRead.ReadInt32();
    courses.Add(crs);
}
bRead.Close();
```

Use the exception class for file I/O

- I/O exceptions are a serious problems like
 - ✓ hardware problems that an application can't do anything about.
- For example,
 - ✓ if an application needs to open a file that's on a network drive that is not available, an exception will be thrown.
- In this case it is common to handle the exception by displaying message.
- When handling I/O exception, it's common to use finally block.
- In this block, use stream Close method to close all streams that are open.
 - ✓ This frees resources that are used to access the file.

File IO Exception class

➤ IOException

- ✓ the base class for exceptions that are thrown during the processing of stream, file, or directory.

➤ DirectoryNotFoundException

- ✓ occurs when part of a directory or file path can't be found

➤ FileNotFoundException

- ✓ occurs when a file can't be found

➤ EndOfStreamException

- ✓ occurs when an application attempts to read beyond the end of a stream

➤ ...

File IO Exception > Example

```
private void btnOpen_Click(object sender, EventArgs e)
{
    string dirPath = @"C:\csharp\file\";
    string filePath = dirPath + "course.txt";
    FileStream readStream = null;
    string msg = null;
    try
    {
        readStream = new FileStream(filePath,
                                   FileMode.Open);

        BinaryReader readBinary = new
                                   BinaryReader(readStream);

        msg = readBinary.ReadString();
        MessageBox.Show(msg);
        readStream.Close();
    }
```

```
    catch (FileNotFoundException ex)
    {
        MessageBox.Show (filePath + "not found");
    }
    catch (DirectoryNotFoundException ex)
    {
        MessageBox.Show (dirPath + "not found");
    }
    catch (IOException ex)
    {
        MessageBox.Show (ex.ToString() + "IOException");
    }
    catch (Exception ex)
    {
        MessageBox.Show (ex.ToString());
    }
    Finally
    {
        if (readStream != null)
            readStream.Close();
    }
}
```

Textreader and TextWriter

- **Textreader** and **TextWriter** are
 - ✓ another way to read and write file respectively,
 - ✓ are not stream classes.
- **TextReader**
 - ✓ represents a reader that can read a sequential series of characters.
- **TextWriter**
 - ✓ represents a writer that can write a sequential series of characters.
- The **StreamReader** and **StreamWriter** classes are derived from **TextReader** and **TextWriter** classes respectively, which read characters from streams and strings.

TextReader > Example

```
private void btnRead_Click(object sender, EventArgs e)
{
    try
    {
        string line = null;
        TextReader readFile = new StreamReader("C:\\csharp.txt");
        while (true)
        {
            line = readFile.ReadLine();
            if (line != null)
            {
                MessageBox.Show (line);
            }
        }
        readFile.Close();
        readFile = null;
    }
    catch (IOException ex)
    {
        MessageBox.Show(ex.ToString());
    }
}
```

TextWriter > Example

```
private void btnSave_Click(object sender, EventArgs e)
{
    try
    {
        TextWriter writeFile = new StreamWriter("c:\\csharp.txt");
        writeFile.WriteLine("csharp");
        writeFile.Flush();
        writeFile.Close();
        writeFile = null;
    }
    catch (IOException ex)
    {
        MessageBox.Show(ex.ToString());
    }
}
```

Set File Attributes

- The **System.IO** namespace contains
 - ✓ types that allow reading and writing to files and data streams and
 - ✓ types that provide basic file and directory support.
- Use the **FileInfo** class for typical operations such as
 - ✓ copying, moving, renaming, creating, opening, deleting, and appending to files.
- **FileSystemInfo**.
 - ✓ Base class for **FileInfo** and **DirectoryInfo** class
 - ✓ **Attributes** property is used to gets or sets the attributes for the current file or directory.

File Attributes > Example

```
private void button1_Click(object sender, EventArgs e)
{
    try
    {
        FileInfo file = new FileInfo("c:\\course.txt");
        file.Attributes = FileAttributes.ReadOnly;
        file.Attributes = FileAttributes.Hidden;
    }
    catch (FileNotFoundException ex)
    {
        MessageBox.Show(ex.StackTrace);
    }
}
```

Copy, Move, Delete a File

➤ The `File` class

- ✓ used to get and set file attributes or DateTime information related to the creation, access, and writing of a file.
- ✓ Also used for
 - copying, moving, delete, renaming etc. to file.

➤ Example

- ✓ `File.Copy("c:\\temp.txt", "c:\\copytemp.txt", true);`
- ✓ `File.Delete("c:\\copytemp.txt");`
- ✓ `File.Move("c:\\temp.txt", "c:\\NewLocation\\movetemp.txt");`

Copy, Move, Delete File > Example

```
private void btnCopy_Click(object sender, EventArgs e)
{
    try
    {
        File.Copy("c:\\temp.txt", "c:\\copytemp.txt", true);
        File.Delete("c:\\copytemp.txt");
        File.Move("c:\\temp.txt", "c:\\NewLocation\\movetemp.txt");
    }
    catch (FileNotFoundException ex)
    {
        MessageBox.Show(ex.StackTrace );
    }
}
```

Exercise: working with file - 1

- Create a project **“NotePad”**
 - ✓ Add a **menuStrip** to your form
 - > Add File menu having a menu items
 - Save
 - Open
 - Exit
 - ✓ Add a **richTextBox** on the Form and set ‘**Dock**’ property of **richTextBox** to ‘**Fill**’ parent container
 - ✓ Write an event handler for the “Save” menu item click event.
 - > When the user click the ‘Save’ menu item, it brings the Save dialog box and save the content of the **richTextBox** to a file using **StreamWriter** I/O class.
 - ✓ Write an event handler for ‘Open’ menu item click event.
 - > When it clicked, it brings the open file dialog box, browse to a file and open the content of the file to the **notpad/richTextBox** using **StreamReader** I/O class.
 - ✓ Write event handler for ‘**Exit**’ button to close the form.

Exercise: working with file - 2

- Modify the previous example to save course information on a file
- Write an event handler for the “Save” button click event.
 - ✓ When the user click the ‘Save’ button,
 - > it brings the Save dialog box and
 - > save the Course information in to a file using `BinaryWriter` I/O class.
- Write an event handler for ‘Open’ button click event.
 - ✓ When it clicked,
 - > it brings the open file dialog box, browse to a Course information file and
 - > display the Course information on the list view using `BinaryReader` I/O class.

CGPA Calculator

Course Information

Course Code :

Title :

Credit Hour :

Total Mark :

Letter Grade :

Grade Point :

Code	Title	Credit	Total Mark	Letter Grade	Grade Point

Total Credit Hour :

Total Grade Point :

Cumulative Grade Point Average :

For more information

- Deitel, C#-How to Program. USA, 2010
- Svetlin Nakov *et al.* Fundamentals of Computer Programming With C#. Sofia, 2013
- www.csharp.net-informations.com
- Joel Murach, Anne Boehm. Murach C# 2012, Mike Murach & Associates Inc USA, 2013

THANK YOU!

