NET.C#.08.
Managinge the File system

Managing the files

For many applications a common requirement is the ability to interact with the file system:

- creation of new files
- copying files
- deleting files
- moving files from one directory to another

The classes that are used to browse around the file system and perform the operations are File, FileInfo, Directory, DirectoryInfo, FileSystemInfo, Path

FileSystemInfo – Base class that represents any file system object

FileInfo and File – These classes represent a file on the file system

DirectoryInfo and Directory – These classes represent a folder on the file system

Path – contains static method that you can use to manipulate pathnames

.Net Classes that represent Files and Folders

Directory and File contain only static methods. You use these classes by supplying the path to the appropriate file system object. If you only want to do one operation on a folder or file then using these classes is more efficient, because it saves the overhead of instantiating a .NET class

DirectoryInfo and FileInfo implement the same public methods as Directory and File, but the members of these classes are not static. You need to instantiate these classes before each instance is associated with a particular folder or file. This means that these classes using the same object, tion for the appropriate file ad that information again.

```
//Copying fileFileInfo myF
ile = new FileInfo(@"C:\
Program Files\ReadMe.tx
t");myFile.CopyTo(@"D:\Copies\ReadMe.txt");
```

```
...";
FileInfo file = new FileInfo(filePath);
string destPath = "...";
file.CopyTo(destPath);
```

FileInfo and DirectoryInfo properties

You can use following properties

Name	Description
CreationTime	Time file or folder was created
DirectoryName	Full pathname of the containing folder
Parent	The parent directory of a specified subdirectory
Exists	Whether file or folder exists
Extension	Extension of the file; returns blank for folders
FullName	Full pathname of the file or folder
LastAccessTime	Time file or folder was last accessed
Name	Name of the file or folder
Root	The root portion of the path
Length	The size of the file in bytes

FileInfo and DirectoryInfo methods

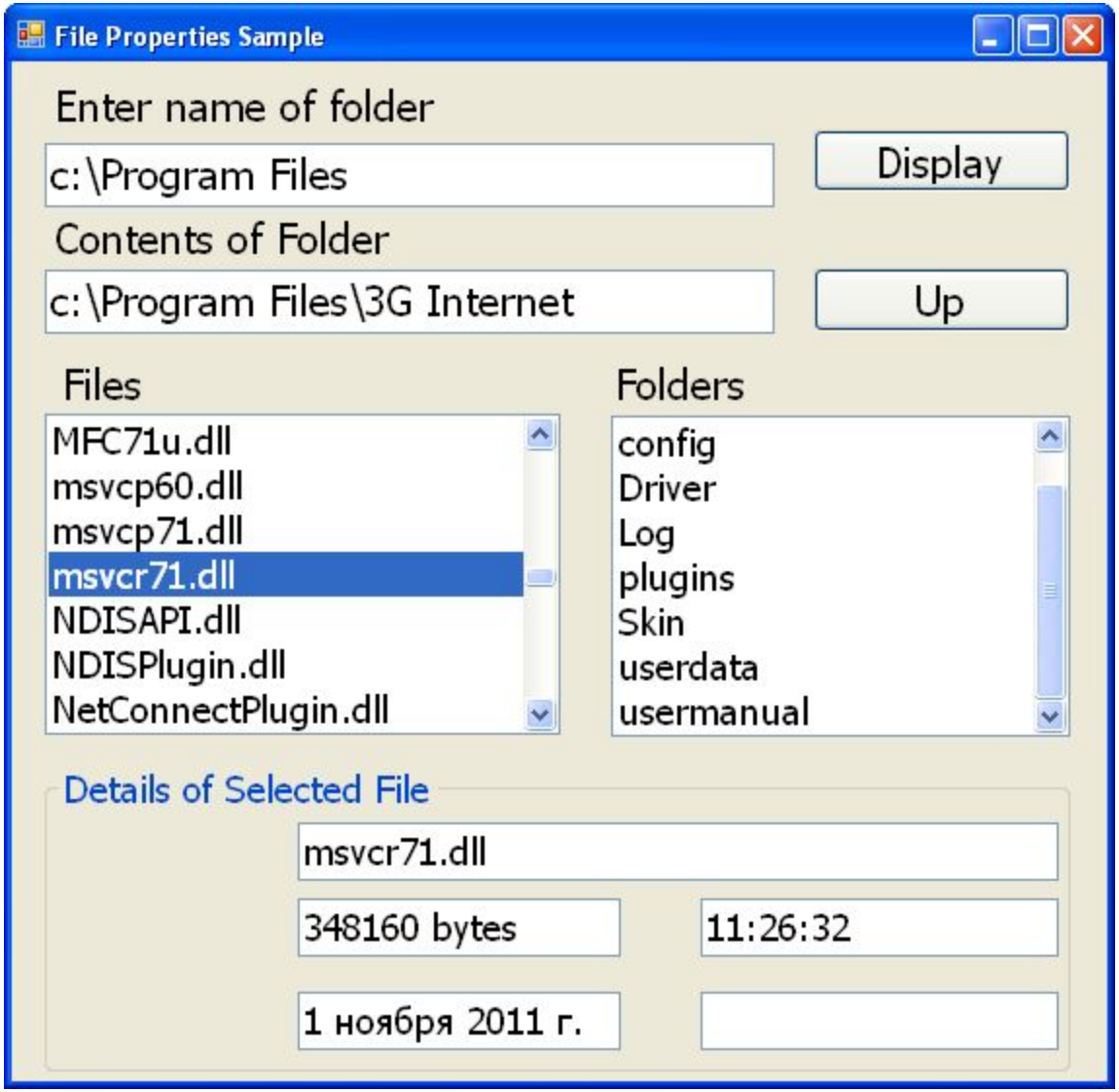
You can also perform actions on the file system object using the following methods:

Name	Description
Create()	**Creates a folder or empty file of the given name. For a FileInfo this also returns a stream object to let you write to the file.**
Delete()	**Deletes the file or folder. For folders there is an option for the Delete to be recursive.**
	r.
	there is no copy Method y trees you'll need to ew folders corresponding to
	Returns an array of DirectoryInfo objects representing all folders contained in this folder.
GetFiles()	**(DirectoryInfo only) Returns an array of FileInfo objects Representing all files contained in this folder.**

```
// displays the creation t
ime of a file, then chan
ges it and// displays i
t againFileInfo test = new FileInfo(@"C:\My
```

Example: A File Browser

that presents a simple user interface that allows you to browse around the file system, and view the creation time, last access time, last write time, and size of files



Example: A File Browser

Step 2. Indica

- textBoxInput
- textBoxFolder
- buttonDisplay
- buttonUp
- listBoxFiles
- listBoxFolders
- textBoxFileName
- textBoxCreationTime
- textBoxFileSize
- textBoxLastAccessTime
- textBoxLastWriteTime

```
Using System.IO;
```

Example: A File Browser

Step 4. Add ev

```
public partial class Form1 : Form
    {
        private string currentFolderPath;
```

Stores the path of the folder whose contants are displayed in the list box

User clicks the Display button

User clicks on a file name in the Files list box

User clicks on a folder name in the Folders list box

User clicks on the Up button

Example: A File Browser

Step 5. Add t

```
//clear the contents of al
l controls protected voi
d ClearAllFields()  {
  listBoxFolders.Items.Clear();    listBoxF
```

Example: A File Browser

Step 5. Add t

```
//Display information for
a given file in the text
 boxesprotected void Di
splayFileInfo(string fileFullName){FileInfo
```

Example: A File Browser

Step 5. Add t

```
//Display the contents of
a given folder in the tw
o list boxesprotected v
oid DisplayFolderList(string folderFullName
```

Example: A File Browser

Step 5. Add t

```
//Display button handlerpr
otected void OnDisplayBu
ttonClick(object sender
, EventArgs e){  string folderPath = textBo
```

Example: A File Browser

Step 5. Add t

```
//Event handler that is ca
lled when an item in the
 Files list//box is sel
ectedprotected void OnListBoxFilesSelected(
```

Example: A File Browser

Step 5. Add t

```
//Event handler for the se
lection of a folder in t
he Folder listprotected
 void OnListBoxFoldersSelected(object sende
```

Example: A File Browser

Step 5. Add t

```
//Event handler for Up but
tonprotected void OnUpBu
ttonClick(object sender
, EventArgs e){try {  string folderPath = n
```

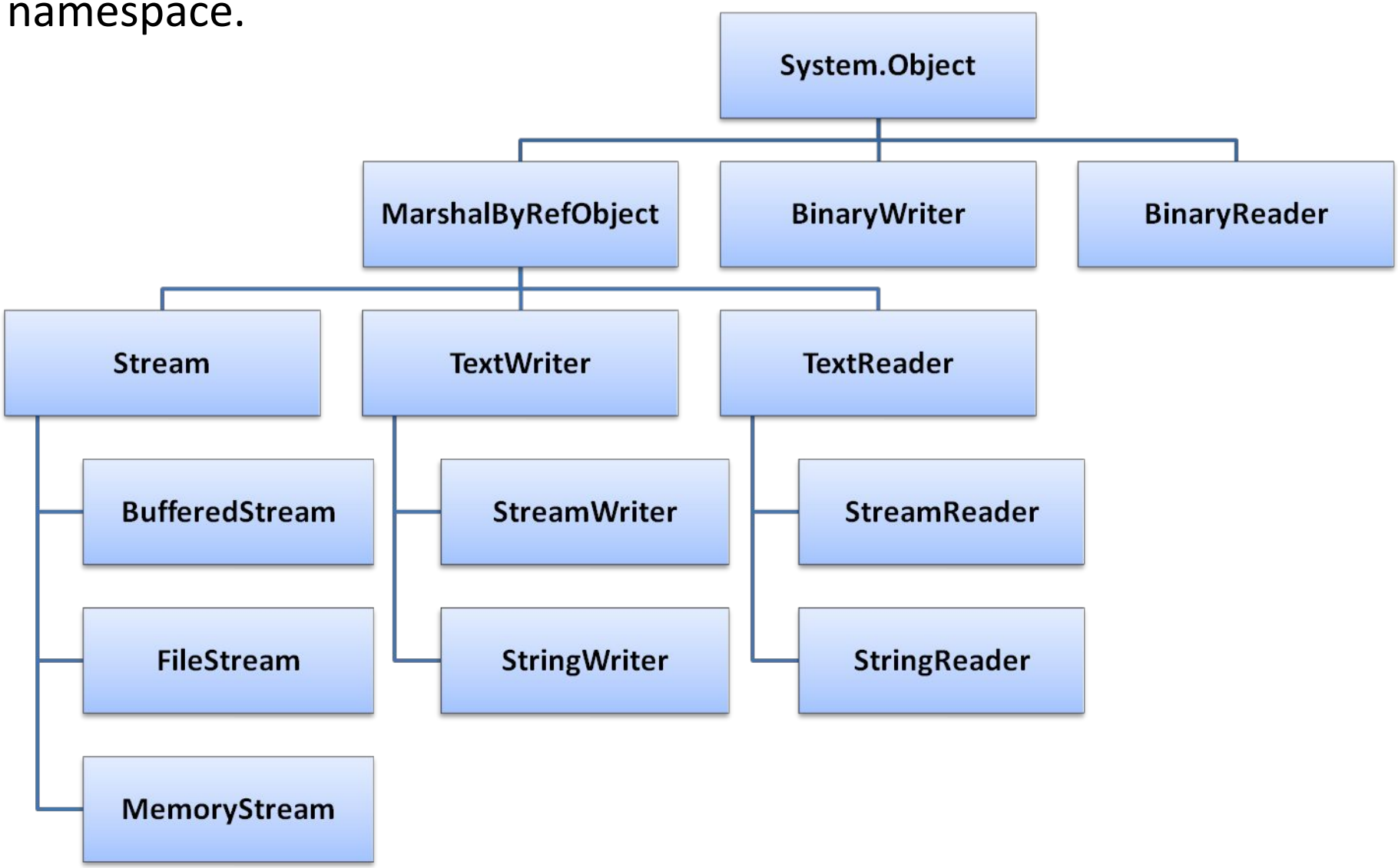
Reading and Writing to Files

Reading and writing to files is in principle very simple; however, it is not done through the DirectoryInfo or FileInfo objects. it is done through a number of classes that represent a generic concept called a stream.

A stream is an object used to transfer data.
The data can be transferred in one of two directions:

- If the data is being transferred from some outside source into your program, then we talk about reading from the stream.
- If the data is being transferred from your program to some outside source, then we talk about writing to the stream.

Actual hierarchy of stream-related classes in the System.IO namespace.



Streams

- FileStream — This class is intended for reading and writing binary data in a binary file. However, you can also use it to read from or write to any file.
- StreamReader and StreamWriter — These classes are designed specifically for reading from and writing to text files.stream.

Reading and Writing Text Files

StreamReader. The simplest constructor takes just a file name

```
StreamReader Sr = new St
reaReader(@”c:\Documents\ReadMe.txt”);//spe
cif
y encoding (ASCII, Unicode, UTF8)StreamReader
 sr
 = new StreamReader(@”c:\Documents\ReadMe.txt
```

Reading and Writing Text Files

StreamWriter. Write each line of the text box to a StreamWriter steram

```
void SaveFile(){  Stream
Writer sw = new StreamWriter(chosenFile, fa
lse
,  Encoding.Unicode);  foreach (string line i
n t
extBoxContents.Lines)  sw.WriteLine(line);  s
```

Чтение и запись двоичных данных

При построении объектам BinaryReader и BinaryWriter предоставляется поток, подключенный к источнику данных, из которого нужно читать или в который необходимо писать

```
string filePath = "...";
FileStream file = new FileStream(filePath);
...
BinaryReader reader = new BinaryReader(file);
...
BinaryWriter writer = new BinaryWriter(file);
```

- Close()
- ReadBytes()
- Read()
- BaseStream
- ReadByte()

- Close()
- Write()
- Flush()
- BaseStream
- Seek()

Когда использование объектов BinaryReader или BinaryWriter завершено, необходимо вызвать метод Close, чтобы флешировать поток и освободить любые ресурсы, связанные с потоком

Необходимо также закрыть объект FileStream, предоставляющий данные для объектов BinaryReader и BinaryWriter.

Чтение и запись двоичных данных

```
string destinationFilePath = @"C:\. . .\BinaryDataFile.bin";
byte[] dataCollection = { 1, 4, 6, 7, 12, 33, 26, 98, 82, 101
};
FileStream destFile = new FileStream(
                      destinationFile   th,
                      FileMode.Create
                      FileAccess.Write);
BinaryWriter writer = new BinaryWriter(destFile)
foreach (byte data in dataCollecti
{
    writer.Write(data);
}
writer.Close();
destFile.Close();
```

Коллекция байт

Создание объекта FileStream для взаимодействия с файловой системой

Создание объекта BinaryWriter, передавая объект FileStream как параметр

Запись каждого байта в поток

Закрыть оба потока для сброса данных в файл

```
00000000  01 04 06 07 0C 21 1A 62  52 65
```

Чтение и запись двоичных данных

```
string sourceFilePath = @"C:\. . .\BinaryDataFile.bin";
FileStream sourceFile = new FileStream(
                        sourceFilePath,
                        FileMode.Open
                        FileAccess.Read);
BinaryReader reader = new BinaryReader(sourceFile);
int position = 0;
int length = (int)reader.BaseStream.Length;
byte[] dataCollection = new byte[length];
int returnedByte;
while ((returnedByte = reader.Read()) != -1)
{
    dataCollection[position] = (byte)returnedByte;
    position += sizeof(byte);
}
reader.Close();
sourceFile.Close();
```

Создание объекта FileStream для взаимодействия с файловой системой

Создание объекта BinaryWriter, передавая объект FileStream как параметр

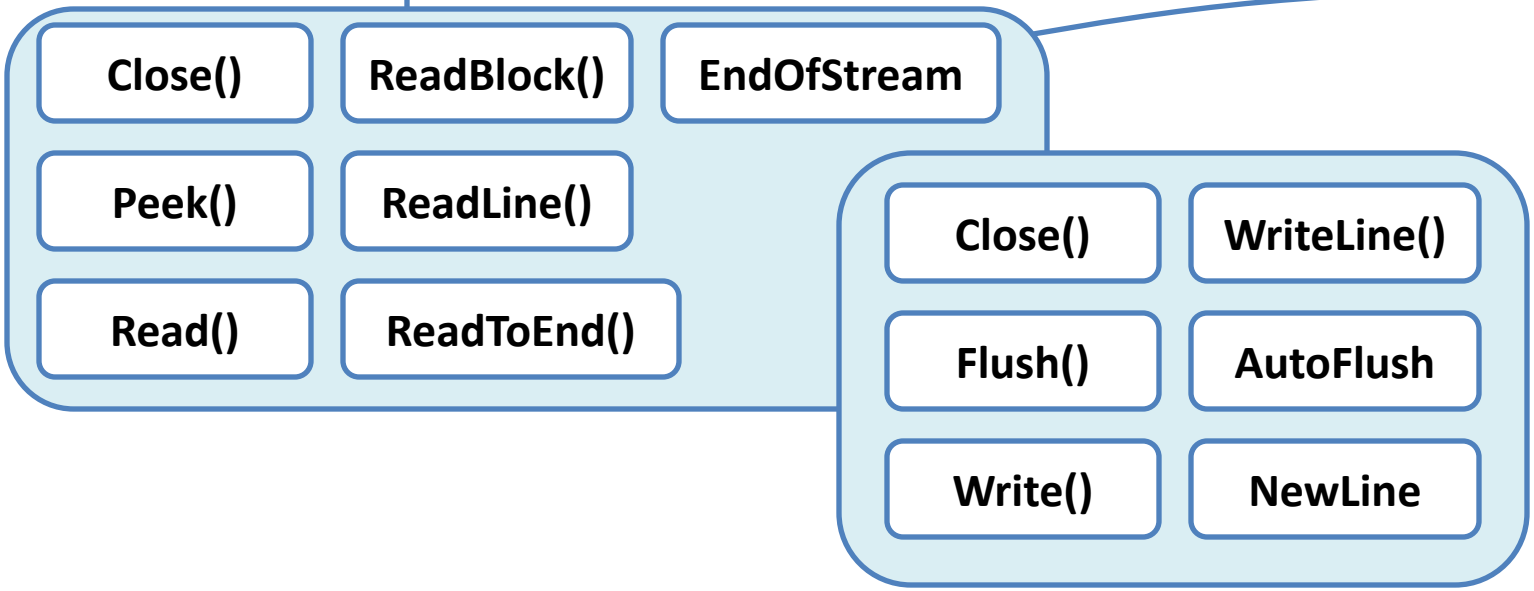
Закрытие потоков для освобождения всех дескрипторов файлов

Если операции чтения из файла или записи в файл генерируют исключение, следует убедиться, что потоки и дескрипторы файлов освобождены

Reading and writing Text Files

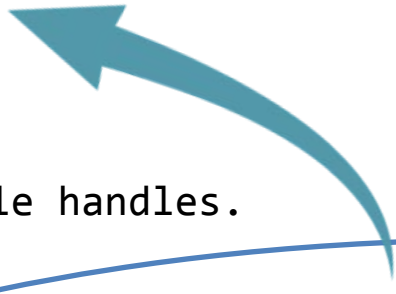
We can use StreamReader and StreamWriter classes

```
string filePath = "...";
FileStream file = new FileStream(filePath );
...
StreamReader reader = new StreamReader(file);
...
StreamWriter writer = new StreamWriter(file);
```



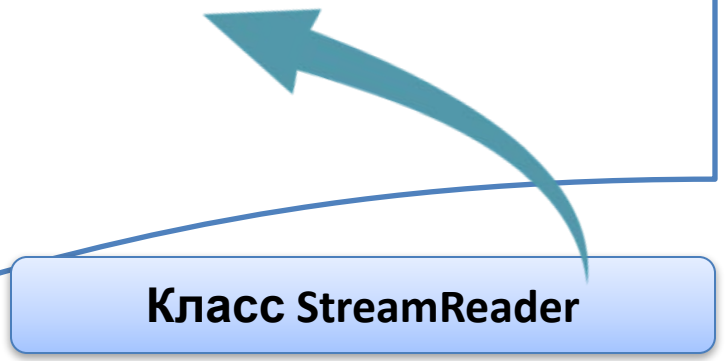
Reading and Writing Text Files

```
string sourceFilePath = @"C:\. . .\TextDataFile.txt";
// Create a FileStream object so that you can interact with the file system
FileStream sourceFile = new FileStream(
                          sourceFilePath, // Pass in the source file path.
                          FileMode.Open,   // Open an existing file.
                          FileAccess.Read);// Read an existing file.
StreamReader reader = new StreamReader(sourceFile);
StringBuilder fileContents = new StringBuilder();
// Check to see if the end of the file has been reached.
while (reader.Peek() != -1)
{
    // Read the next character.
    fileContents.Append((char)reader.Read());
}
// Store the file contents in a new string variable.
string data = fileContents.ToString();
// Always close the underlying streams release any file handles.
reader.Close();
sourceFile.Close();
```



Reading and Writing Text Files

```
string sourceFilePath = @"C:\. . .\TextDataFile.txt";
string data;
// Create a FileStream object so that you can
// interact with the file system.
FileStream sourceFile = new FileStream(
                            sourceFilePath,  // Pass in the source file path.
                            FileMode.Open,   // Open an existing file.
                            FileAccess.Read);// Read an existing file.
StreamReader reader = new StreamReader(sourceFile);
// Read the entire file into a single string variable.
data = reader.ReadToEnd();
// Always close the underlying streams release any file handles.
reader.Close();
sourceFile.Close();
```



Reading and Writing Text Files

```
string destinationFilePath = @"C:\. . .\TextDataFile.txt";
string data = "Hello, this will be written in plain text";
// Create a FileStream object so that you can interact with the file
// system.
FileStream destFile = new FileStream(
                      destinationFilePath, // Pass in the destination path.
                      FileMode.Create,     // Always create new file.
                      FileAccess.Write);   // Only perform writing.
// Create a new StreamWriter object.
StreamWriter writer = new StreamWriter(destFile);
// Write the string to the file.
writer.WriteLine(data);
// Always close the underlying streams to flush the data to the file
// and release any file handles.
writer.Close();
destFile.Close();
```

Thank you for your attention