Файловый ввод/вывод

Класс FileInfo

Подобно классу DirectoryInfo, класс FileInfo используется для манипуляции с файлами. В таблице приведены его основные члены.
Таблица Основные члены класса FileInfo

Член	Описание
Create()	Создает новый файл и возвращает объект FileStream, который используется для взаимодействия с созданным файлом.
CreateText()	Создает объект StreamWriter, который используется для создания текстового файла.
CopyTo()	Копирует существующий файл в другой файл.
AppendText()	Создает объект StreamWriter для добавления текста в файл
Delete()	Удаляет файл, связанный с экземпляром FileInfo
Directory	Используется для получения экземпляра родительского каталога
DirectoryName	Содержит полный путь к родительскому каталогу
Length	Получает размер текущего файла или каталога
MoveTo()	Используется для перемещения файла
Name	Содержит имя файла
Open()	Открывает файл с различными разрешениями чтения/записи
OpenRead()	Создает доступный только для чтения объект FileStream
OpenText()	Создает объект StreamReader для чтения информации из текстового файла
OpenWrite()	Создает объект FileStream, доступный только для записи

Теперь рассмотрим примеры использования класса FileInfo. Начнем с примеров использования метода Create():

```
FileInfo myFile = new FileInfo(@”C:\temp\file.vdd”);
FileStream fs = myFile.Create();
// производим какие-либо операции с fs
// закрывам поток
fs.Close();
```

Итак, мы создали поток типа FileStream, позволяющий производить манипуляции с содержимым файла, например, читать из него данные, записывать в него данные. Позже этот поток будет подробно рассмотрен, пока разберемся с созданием и открытием файлов.
Для открытия файла используется метод Open(), позволяющий как открывать существующие файлы, так и создавать новые. Причем этот метод принимает несколько параметров, в отличие от метода Create(), что позволяет более гибко управлять процессом

открытия/создания файлов. В результате выполнения метода Open() создается объект типа FileStream, как и в предыдущем случае:

```
FileInfo mf = new FileInfo(@"C:\temp\file.vdd");
FileStream fs = mf.Open(FileMode.OpenOrCreate, FileAccees.Read-
Write, FileShare.None);
```

Методу Open() нужно передать три параметра. Первый из них тип запроса ввода/вывода из перечисления типа FileMode:

```
public enum FileMode
{
CreateNew,
Create,
Open,
OpenOrCreate,
Truncate,
Append
}
```

Рассмотрим члены этого перечисления:

• CreateNew - создать новый файл, если он существует, будет сгенерировано исключение IOException.

• Create - создать новый файл, если он существует, он будет перезаписан.

• Open - открыть существующий файл. Если он не существует, будет сгенерировано исключение FileNotFoundException.

• OpenOrCreate - открывает файл, если он существует. Если файл не существует, он будет создан.

• Truncate - открывает файл и усекает его до нулевой длины. По сути, удаляет все его содержимое.

• Append - открывает файл и переходит в самый его конец. Этот флаг может использоваться только для потоков, которые открыты только для чтения. Если файл не существует, он будет создан.

Второй параметр метода Open() - одно из значений перечисления FileAccess. Используется для определения операций чтения/записи:

```
public enum FileAccess
{
Read,
Write,
ReadWrite
}
```

Думаю, значения членов перечисления FileAccess в комментариях не нуждаются. Третий параметр метода Open() - член перечисления FileShare, задающий тип совместного доступа к этому файлу:

```
public enum FileShare
{
Delete,
Inheritable,
None,
Read,
ReadWrite,
Write
}
```

Методы OpenRead() и OpenWrite() используются для создания потоков, доступных только для чтения и только для записи. Данные методы возвращают поток FileStream,

сконфигурированный соответствующим образом. В принципе, можно было бы обойтись только методом Open(), но использовать эти методы немного удобнее - ведь вам не нужно применять различные значения из перечислений. Примеры:

```
FileInfo f = new FileInfo(@"C:\temp\1.dat");
using (FileStream ro = f.OpenRead())
{
// выполняем операции чтения из файла
}
FileInfo f2 = new FileInfo(@"C:\temp\2.dat");
using (FileStream rw = f2.OpenWrite())
{
// записываем в файл
}
```

Для работы с текстовыми файлами пригодятся методу OpenText(), CreateText() и AppendText(). Первый метод возвращает экземпляр типа StreamReader (в отличие от методов Create(), Open() и OpenRead/Write(), которые возвращают тип FileStream). Пример открытия текстового файла:

```
FileInfo txt = new FileInfo(@"program.log");
using(StreamReader txt_reader = txt.OpenText ())
{
// Читаем данные из текстового файла
}
```

Методы CreateText() и ApendText() возвращают объект типа StreamWriter. Использовать эти методы можно так:

```
FileInfo f = new FileInfo(@"C:\temp\1.txt");
using(StreamWriter sw = f.CreateText())
{
// Записываем данные в файл...
}
FileInfo f = new FileInfo(@"C:\temp\2.txt");
using(StreamWriter swa = f.AppendText())
{
// Добавляем данные в текстовый файл
}
```

Класс File

Тип File поддерживает несколько полезных методов, которые пригодятся при работе с текстовыми файлами. Например, метод ReadAllLines() позволяет открыть указанный файл и прочитать из него все строки - в результате будет возвращен массив строк. После того, как все данные из файла прочитаны, файл будет закрыт.

Аналогично, метод ReadAllBytes() читает все байты из файла, возвращает массив байтов и закрывает файл.

Метод ReadAllText() читает все содержимое текстового файла в одну строку и возвращает ее. Как обычно, файл после чтения будет закрыт.

Существуют и аналогичные методы записи WriteAllBytes(), WriteAllLines() и WriteAlltext(), которые записывают в файл, соответственно, массив байтов, массив строк и строку. После записи файл закрывается.

Пример:

```
string[] myFriends = {"Jane", "Max", "John" };
// Записать все данные в файл
File.WriteAllLines(@”C:\temp\friends.txt”, myFriends);
// Прочитать все обратно и вывести
```

```
foreach (string friend in File.ReadAllLines(@"C:\temp\friends.
txt"))
{
Console.WriteLine("{0}", friend);
}
```