**13.1. РАБОТА С ПОТОКАМИ И ФАЙЛОВОЙ СИСТЕМОЙ**

**Оглавление**

[§13.1 Пространство имен System.IO 1](#_Toc62986138)

[§13.2 Работа с дисками 3](#_Toc62986139)

[§13.3 Работа с каталогами 5](#_Toc62986140)

[§13.4 Работа с файлами. Классы File и FileInfo 8](#_Toc62986141)

[§13.5 Класс FileStream. Чтение и запись файла 13](#_Toc62986142)

[§13.6 Чтение и запись текстовых файлов. StreamReader и StreamWriter 20](#_Toc62986143)

## §13.1 Пространство имен System.IO

Большинство задач в программировании, так или иначе, связаны с работой с файлами и каталогами. Нам может потребоваться прочитать текст из файла или наоборот произвести запись, удалить файл или целый каталог, не говоря уже о более комплексных задачах, как например, создание текстового редактора и других подобных задачах.

Фреймворк .NET предоставляет большие возможности по управлению и манипуляции файлами и каталогами, которые по большей части сосредоточены в пространстве имен System.IO.

В таблице 13.1 представлены важнейшие классы, расположенные в пространстве имен, которые позволяют управлять файловым вводом-выводом.

**Таблица 13.1 – Классы пространства имен System.IO**

|  |  |
| --- | --- |
| **Класс** | **Описание** |
| File | Предоставляет статические методы для создания, копирования, удаления, перемещения и открытия одного файла, |
| FileInfo | Предоставляет свойства и методы экземпляра для создания, копирования, удаления, перемещения и открытия файлов. Тип FileInfo обеспечивает аналогичную типу File функциональность, но через действительную объектную ссылку. Этот класс не наследуется. |
| FileStream | Обеспечивает произвольный доступ к файлу (т.е. возможности поиска) с данными, представленными в виде потока байт. |
| StreamReader | Используется для извлечения текстовой информации из файла. Этот класс не поддерживает произвольный доступ к файлу. |
| StreamWriter | Используется для записи текстовой информации в файл. Этот класс не поддерживает произвольный доступ к файлу. |
| DriveInfo | Предоставляет детальную информацию относительно дисковых устройств, используемых данной машиной. |
| Path | Выполняет операции над типами System.String, содержащими информацию о пути к файлу или каталогу в независимой от платформы манере. |

## §13.2 Работа с дисками

Работу с файловой системой начнем с самого верхнего уровня - дисков. Для представления диска в пространстве имен System.IO имеется класс **DriveInfo**.

Этот класс имеет статический метод GetDrives, который возвращает имена всех логических дисков компьютера. Также он предоставляет ряд полезных свойств:

**Таблица 13.2 – Свойства класса DriveInfo**

|  |  |
| --- | --- |
| **Свойство** | **Описание** |
| AvailableFreeSpace | указывает на объем доступного свободного места на диске в байтах |
| DriveFormat | получает имя файловой системы |
| DriveType | представляет тип диска |
| IsReady | готов ли диск (например, DVD-диск может быть не вставлен в дисковод) |
| Name | получает имя диска |
| TotalFreeSpace | получает общий объем свободного места на диске в байтах |
| TotalSize | общий размер диска в байтах |
| VolumeLabel | получает или устанавливает метку тома - уникальное имя, назначенное жесткому диску |

Получим имена и свойства всех дисков на компьютере:

**Листинг 13.1**

|  |  |
| --- | --- |
| 1 | using System; |
| 2 | using System.IO; |
| 3 | namespace HelloApp |
| 4 | { |
| 5 | class Program |
| 6 | { |
| 7 | static void Main(string[] args) |
| 8 | { |
| 9 | DriveInfo[] drives = DriveInfo.GetDrives(); |
| 10 | foreach (DriveInfo drive in drives) |
| 11 | { |
| 12 | Console.WriteLine($"Название: {drive.Name}"); |
| 13 | Console.WriteLine($"Тип: {drive.DriveType}"); |
| 14 | if (drive.IsReady) |
| 15 | { |
| 16 | Console.WriteLine($"Объем диска: {drive.TotalSize}"); |
| 17 | Console.WriteLine($"Свободное пространство: {drive.TotalFreeSpace}"); |
| 18 | Console.WriteLine($"Метка: {drive.VolumeLabel}"); |
| 19 | } |
| 20 | Console.WriteLine(); |
| 21 | } |
| 22 | } |
| 23 | } |
| 24 | } |

В программе, представленной в листинге 13.1, выводится имя диска (строка 12), тип диска (строка 13), при готовности диска (строка 14 – условие проверки готовности диска) выводится общий размер места для хранения на диске в байтах (строка 16), общий объем свободного места на диске в байтах (строка 17), метка тома (строка 18).

## §13.3 Работа с каталогами

Для работы с каталогами в пространстве имен System.IO предназначены сразу два класса: **Directory** и **DirectoryInfo**.

*Класс Directory*

Класс Directory предоставляет ряд статических методов для управления каталогами. Некоторые из этих методов:

**Таблица 13.3 – Методы класса Directory**

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| CreateDirectory(path) | создает каталог по указанному пути path |
| Delete(path) | удаляет каталог по указанному пути path |
| Exists(path) | определяет, существует ли каталог по указанному пути path. Если существует, возвращается true, если не существует, то false |
| GetDirectories(path) | получает список каталогов в каталоге path |
| GetFiles(path) | получает список файлов в каталоге path |
| Move(sourceDirName, destDirName) | перемещает каталог |
| GetParent(path) | получение родительского каталога |

*Класс DirectoryInfo*

Данный класс предоставляет функциональность для создания, удаления, перемещения и других операций с каталогами. Во многом он похож на Directory. Некоторые из его свойств и методов:

**Таблица 13.4 – Методы и свойства класса DirectoryInfo**

|  |  |
| --- | --- |
| **Свойства и методы** | **Описание** |
| Create() | создает каталог |
| CreateSubdirectory(path) | создает подкаталог по указанному пути path |
| Delete() | удаляет каталог |
| GetDirectories() | получает список каталогов |
| GetFiles() | получает список файлов |
| MoveTo(destDirName) | перемещает каталог |
| Свойство Parent | получение родительского каталога |
| Свойство Root | получение корневого каталога |
| Свойство Exists | определяет, существует ли каталог |

Посмотрим на примерах применение этих классов.

В листинге 13.2 показан пример получения списка файлов и подкаталогов.

**Листинг 13.2 – Получение списка файлов и подкаталогов**

|  |  |
| --- | --- |
| 1 | string dirName = "C:\\"; |
| 2 | if (Directory.Exists(dirName)) |
| 3 | { |
| 4 | Console.WriteLine("Подкаталоги:"); |
| 5 | string[] dirs = Directory.GetDirectories(dirName); |
| 6 | foreach (string s in dirs) |
| 7 | { |
| 8 | Console.WriteLine(s); |
| 9 | } |
| 10 | Console.WriteLine(); |
| 11 | Console.WriteLine("Файлы:"); |
| 12 | string[] files = Directory.GetFiles(dirName); |
| 13 | foreach (string s in files) |
| 14 | { |
| 15 | Console.WriteLine(s); |
| 16 | } |
| 17 | } |

Обратите внимание на использование слешей в именах файлов. Либо мы используем двойной слеш: "C:\\", либо одинарный, но тогда перед всем путем ставим знак @:

@"C:\Program Files".

В листинге 13.3 показан пример создания каталога с помощью метода Create() для объекта класса DirectoryInfo.

**Листинг 13.3 – Создание каталога**

|  |  |
| --- | --- |
| 1 | string path = @"C:\SomeDir"; |
| 2 | string subpath = @"program\avalon"; |
| 3 | DirectoryInfo dirInfo = new DirectoryInfo(path); |
| 4 | if (!dirInfo.Exists) |
| 5 | { |
| 6 | dirInfo.Create(); |
| 7 | } |
| 8 | dirInfo.CreateSubdirectory(subpath); |

Вначале проверяем, а нету ли такой директории, так как если она существует, то ее создать будет нельзя, и приложение выбросит ошибку. В итоге у нас получится следующий путь: "C:\SomeDir\program\avalon".

Если мы просто применим метод Delete к непустой папке, в которой есть какие-нибудь файлы или подкаталоги, то приложение нам выбросит ошибку. Поэтому нам надо передать в метод Delete дополнительный параметр булевого типа, который укажет, что папку надо удалять со всем содержимым:

**Листинг 13.4 – Удаление каталога**

|  |  |
| --- | --- |
| 1 | string dirName = @"C:\SomeFolder"; |
| 2 | try |
| 3 | { |
| 4 | DirectoryInfo dirInfo = new DirectoryInfo(dirName); |
| 5 | dirInfo.Delete(true); |
| 6 | Console.WriteLine("Каталог удален"); |
| 7 | } |
| 8 | catch (Exception ex) |
| 9 | { |
| 10 | Console.WriteLine(ex.Message); |
| 11 | } |

В листинге 13.5 показан пример перемещения каталога из одного расположения в другое.

**Листинг 13.5 – Перемещение каталога**

|  |  |
| --- | --- |
| 1 | string oldPath = @"C:\SomeFolder"; |
| 2 | string newPath = @"C:\SomeDir"; |
| 3 | DirectoryInfo dirInfo = new DirectoryInfo(oldPath); |
| 4 | if (dirInfo.Exists && Directory.Exists(newPath) == false) |
| 5 | { |
| 6 | dirInfo.MoveTo(newPath); |
| 7 | } |

При перемещении надо учитывать, что новый каталог, в который мы хотим перемесить все содержимое старого каталога, не должен существовать.

## §13.4 Работа с файлами. Классы File и FileInfo

Подобно паре Directory/DirectoryInfo для работы с файлами предназначена пара классов File и FileInfo. С их помощью мы можем создавать, удалять, перемещать файлы, получать их свойства и многое другое.

Класс FileInfo предоставляет свойства и методы экземпляра для создания, копирования, удаления, перемещения и открытия файлов, а также позволяет создавать объекты FileStream для операции чтения и записи. Этот класс не наследуется.

Конструктор класса FileInfo выполняет инициализацию нового экземпляра класса FileInfo, который служит оболочкой для пути файла.

Конструктор имеет следующую форму:

public FileInfo (string fileName);

Здесь fileName – полное имя нового файла или относительное имя файла. Путь не должен заканчиваться символом разделителя каталогов.

Некоторые полезные методы и свойства класса FileInfo:

**Таблица 13.5 – Методы и свойства класса FileInfo**

| **Методы и свойства** | **Описание** |
| --- | --- |
| CopyTo(path) | копирует файл в новое место по указанному пути path |
| Create() | создает файл |
| Delete() | удаляет файл |
| MoveTo(destFileName) | перемещает файл в новое место |
| Свойство Directory | получает родительский каталог в виде объекта DirectoryInfo |
| Свойство DirectoryName | получает полный путь к родительскому каталогу |
| Свойство Exists | указывает, существует ли файл |
| Свойство Length | получает размер файла |
| Свойство Extension | получает расширение файла |
| Свойство Name | получает имя файла |
| Свойство FullName | получает полное имя файла |

Класс File предоставляет статические методы для создания, копирования, удаления, перемещения и открытия одного файла, а также помогает при создании объектов FileStream.

В таблице 13.6 перечислены статические методы класса File:

**Таблица 13.6 – Методы класса File**

|  |  |
| --- | --- |
| **Методы и свойства** | **Описание** |
| Copy(string sourceFileName, string destFileName, bool overwrite) | Копирует существующий файл в новый файл. Перезапись файла с тем же именем разрешена.  Параметры метода:   1. sourceFileName – копируемый файл; 2. destFileName – имя целевого файла. Это не может быть имя каталога; 3. overwrite – если равен true, то конечный файл можно перезаписать; в противном случае — false. |
| Create(string path) | Создает или перезаписывает файл в указанном пути path |
| Delete(string path) | Удаляет указанный файл. Параметр path – это имя файла, предназначенного для удаления |
| Move(string sourceFileName, string destFileName) | Перемещает заданный файл в новое местоположение.  Параметры метода:   1. sourceFileName – Имя перемещаемого файла. Может содержать относительный или абсолютный путь; 2. destFileName – Новый путь к файлу и его имя. |
| Exists(string path) | Определяет, существует ли заданный файл. Параметр path – это проверяемый файл. |

Один из способов создания дескриптора файла предусматривает использование метода FileInfo.Create():

**Листинг 13.6 – Получение информации о файле**

|  |  |
| --- | --- |
| 1 | // Создаем новый файл |
| 2 | FileInfo f = new FileInfo(@"C:\Test.dat"); |
| 3 | FileStream fs = f.Create(); |
| 4 | // Закрыть файловый поток |
| 5 | fs.Close(); |

Метод FileInfo.Create() возвращает тип FileStream, который предоставляет операции записи/чтения лежащего в его основе файла. Имейте в виду, что объект FileStream, возвращенный FileInfo.Create() открывает полный доступ по чтению и записи всем пользователям.

Также имейте в виду, что после окончания работы с текущим объектом FileStream следует закрыть его дескриптор, чтобы освободить лежащие в основе потока неуправляемые ресурсы.

**Листинг 13.7 – Получение информации о файле**

|  |  |
| --- | --- |
| 1 | static void Main(string[] args) |
| 2 | { |
| 3 | string path = @"c:\LEC\Test.txt"; |
| 4 | FileInfo fileInf = new FileInfo(path); |
| 5 | if (fileInf.Exists) |
| 6 | { |
| 7 | Console.WriteLine("Имя файла: {0}", fileInf.Name); |
| 8 | Console.WriteLine("Полное имя: {0}", fileInf.FullName); |
| 9 | Console.WriteLine("Директория размещения файла: {0}", fileInf.DirectoryName); |
| 10 | Console.WriteLine("Время создания: {0}", fileInf.CreationTime); |
| 11 | Console.WriteLine("Время последнего изменения: {0}", fileInf.LastWriteTime); |
| 12 | Console.WriteLine("Размер: {0}", fileInf.Length); |
| 13 | } |
| 14 | } |

В листинге 13.8 показана реализация удаления файла с помощью метода Delete() для объекта класса FileInfo.

**Листинг 13.8 - Удаление файла**

|  |  |
| --- | --- |
| 1 | string path = @"c:\LEC\Test.txt"; |
| 2 | FileInfo fileInf = new FileInfo(path); |
| 3 | if (fileInf.Exists) |
| 4 | { |
| 5 | fileInf.Delete(); |
| 6 | // альтернатива с помощью класса File |
| 7 | // File.Delete(path); |
| 8 | } |

В листинге 13.9 показана реализация перемещения файла в новое место с помощью метода MoveTo() для объекта класса FileInfo.

**Листинг 13.9 – Перемещение файла**

|  |  |
| --- | --- |
| 1 | string path = @"c:\LEC\Test.txt"; |
| 2 | string newPath = @"C:\PATH2\Test2.txt"; |
| 3 | FileInfo fileInf = new FileInfo(path); |
| 4 | if (fileInf.Exists) |
| 5 | { |
| 6 | fileInf.MoveTo(newPath); |
| 7 | // альтернатива с помощью класса File |
| 8 | // File.Move(path, newPath); |
| 9 | } |

В листинге 13.10 показана реализация копирование существующего файла в новое расположение с помощью метода Copy() для объекта класса FileInfo.

**Листинг 13.10 – Копирование файла**

|  |  |
| --- | --- |
| 1 | string path = @"c:\LEC\Test.txt"; |
| 2 | string newPath = @"C:\PATH2\Test2.txt"; |
| 3 | FileInfo fileInf = new FileInfo(path); |
| 4 | if (fileInf.Exists) |
| 5 | { |
| 6 | fileInf.CopyTo(newPath, true); |
| 7 | // альтернатива с помощью класса File |
| 8 | // File.Copy(path, newPath, true); |
| 9 | } |

Метод CopyTo класса FileInfo принимает два параметра: путь, по которому файл будет копироваться, и булевое значение, которое указывает, надо ли при копировании перезаписывать файл (если true, как в случае выше, файл при копировании перезаписывается). Если же в качестве последнего параметра передать значение false, то если такой файл уже существует, приложение выдаст ошибку.

Метод Copy класса File принимает три параметра: путь к исходному файлу, путь, по которому файл будет копироваться, и булевое значение, указывающее, будет ли файл перезаписываться.

## §13.5 Класс FileStream. Чтение и запись файла

Класс FileStream представляет возможности по считыванию из файла и записи в файл. Он позволяет работать как с текстовыми файлами, так и с бинарными.

Для создания объекта FileStream можно использовать как конструкторы этого класса, так и статические методы класса File. Конструктор FileStream имеет множество перегруженных версий, из которых отметим лишь одну, самую простую и используемую:

FileStream(string filename, FileMode mode)

Здесь в конструктор передается два параметра: путь к файлу и перечисление FileMode. Данное перечисление указывает на режим доступа к файлу и может принимать следующие значения:

**Таблица 13.7 – Значения перечисления FileMode**

|  |  |
| --- | --- |
| **Режим доступа** | **Описание** |
| Append | Если файл существует, то текст добавляется в конец файл. Если файла нет, то он создается. Файл открывается только для записи. |
| Create | Создается новый файл. Если такой файл уже существует, то он перезаписывается |
| CreateNew | Создается новый файл. Если такой файл уже существует, то он приложение выбрасывает ошибку. |
| Open | Открывает файл. Если файл не существует, выбрасывается исключение. |
| OpenOrCreate | Если файл существует, он открывается, если нет - создается новый. |
| Truncate | Если файл существует, то он перезаписывается. Файл открывается только для записи. |

Другой способ создания объекта FileStream представляют статические методы класса File:

**Листинг 13.11 – Статические методы создания объекта FileStream**

|  |  |
| --- | --- |
| 1 | FileStream File.Open(string file, FileMode mode) |
| 2 | FileStream File.OpenRead(string file) |
| 3 | FileStream File.OpenWrite(string file) |

Первый метод открывает файл с учетом объекта FileMode и возвращает файловой поток FileStream. У этого метода также есть несколько перегруженных версий. Второй метод открывает поток для чтения, а третий открывает поток для записи.

Рассмотрим наиболее важные его свойства и методы класса FileStream:

**Таблица 13.8 – Свойства и методы класса FileStream**

| **Свойства и методы** | **Описание** |
| --- | --- |
| Свойство Length | возвращает длину потока в байтах |
| Свойство Position | возвращает текущую позицию в потоке |
| void CopyTo(Stream destination) | копирует данные из текущего потока в поток destination |
| int Read(byte[] array, int offset, int count) | считывает данные из файла в массив байтов и возвращает количество успешно считанных байтов.  Принимает три параметра:   1. array - массив байтов, куда будут помещены считываемые из файла данные; 2. offset представляет смещение в байтах в массиве array, в который считанные байты будут помещены; 3. count - максимальное число байтов, предназначенных для чтения. Если в файле находится меньшее количество байтов, то все они будут считаны. |
| long Seek(long offset, SeekOrigin origin) | устанавливает позицию в потоке со смещением на количество байт, указанных в параметре offset |
| void Write(byte[] array, int offset, int count) | записывает в файл данные из массива байтов.  Принимает три параметра:   1. array - массив байтов, откуда данные будут записываться в файл 2. offset - смещение в байтах в массиве array, откуда начинается запись байтов в поток 3. count - максимальное число байтов, предназначенных для записи |

*Чтение и запись файлов*

FileStream представляет доступ к файлам на уровне байтов, поэтому, например, если вам надо считать или записать одну или несколько строк в текстовый файл, то массив байтов надо преобразовать в строки, используя специальные методы. Поэтому для работы с текстовыми файлами применяются другие классы.

В то же время при работе с различными бинарными файлами, имеющими определенную структуру, FileStream может быть очень даже полезен для извлечения определенных порций информации и ее обработки.

Посмотрим на примере считывания-записи в текстовый файл:

**Листинг 13.12 – Считывание-запись в текстовый файл**

|  |  |
| --- | --- |
| 1 | using System; |
| 2 | using System.IO; |
| 3 | namespace HelloApp |
| 4 | { |
| 5 | class Program |
| 6 | { |
| 7 | static void Main(string[] args) |
| 8 | { |
| 9 | // создаем каталог для файла |
| 10 | string path = @"C:\Dir1"; |
| 11 | DirectoryInfo dirInfo = new DirectoryInfo(path); |
| 12 | if (!dirInfo.Exists) |
| 13 | { |
| 14 | dirInfo.Create(); |
| 15 | } |
| 16 | Console.WriteLine("Введите строку для записи в файл:"); |
| 17 | string text = Console.ReadLine(); |
| 18 | // запись в файл |
| 19 | using (FileStream fstream = new FileStream($@"{path}\File.txt", FileMode.OpenOrCreate)) |
| 20 | { |
| 21 | // преобразуем строку в байты |
| 22 | byte[] array = System.Text.Encoding.Default.GetBytes(text); |
| 23 | // запись массива байтов в файл |
| 24 | fstream.Write(array, 0, array.Length); |
| 25 | Console.WriteLine("Текст записан в файл"); |
| 26 | } |
| 27 | // чтение из файла |
| 28 | using (FileStream fstream = File.OpenRead($@"{path}\File.txt")) |
| 29 | { |
| 30 | // преобразуем строку в байты |
| 31 | byte[] array = new byte[fstream.Length]; |
| 32 | // считываем данные |
| 33 | fstream.Read(array, 0, array.Length); |
| 34 | // декодируем байты в строку |
| 35 | string textFromFile = System.Text.Encoding.Default.GetString(array); |
| 36 | Console.WriteLine($@"Текст из файла: {textFromFile}"); |
| 37 | } |
| 38 | Console.ReadLine(); |
| 39 | } |
| 40 | } |
| 41 | } |

Разберем этот пример. Вначале создается папка для файла. Кроме того, на уровне операционной системы могут быть установлены ограничения на запись в определенных каталогах, и при попытке создания и записи файла в подобных каталогах мы получим ошибку.

И при чтении, и при записи используется оператор using. Не надо путать данный оператор с директивой using, которая подключает пространства имен в начале файла кода. Оператор using позволяет создавать объект в блоке кода, по завершению которого вызывается метод Dispose у этого объекта, и, таким образом, объект уничтожается. В данном случае в качестве такого объекта служит переменная fstream.

И при записи, и при чтении применяется объект кодировки Encoding.Default из пространства имен System.Text. В данном случае мы используем два его метода: GetBytes для получения массива байтов из строки и GetString для получения строки из массива байтов.

В итоге введенная нами строка записывается в файл file.txt. По сути это бинарный файл (не текстовый), хотя если мы в него запишем только строку, то сможем посмотреть в удобочитаемом виде этот файл, открыв его в текстовом редакторе. Однако если мы в него запишем случайные байты, например:

fstream.WriteByte(13);

fstream.WriteByte(103);

То у нас могут возникнуть проблемы с его пониманием. Поэтому для работы непосредственно с текстовыми файлами предназначены отдельные классы - StreamReader и StreamWriter.

*Произвольный доступ к файлам*

Нередко бинарные файлы представляют определенную структуру. И, зная эту структуру, мы можем взять из файла нужную порцию информации или наоборот записать в определенном месте файла определенный набор байтов. Например, в wav-файлах непосредственно звуковые данные начинаются с 44 байта, а до 44 байта идут различные метаданные - количество каналов аудио, частота дискретизации и т.д.

С помощью метода Seek() мы можем управлять положением курсора потока, начиная с которого производится считывание или запись в файл. Этот метод принимает два параметра: offset (смещение) и позиция в файле. Позиция в файле описывается тремя значениями:

1. SeekOrigin.Begin: начало файла
2. SeekOrigin.End: конец файла
3. SeekOrigin.Current: текущая позиция в файле

Курсор потока, с которого начинается чтение или запись, смещается вперед на значение offset относительно позиции, указанной в качестве второго параметра. Смещение может быть отрицательным, тогда курсор сдвигается назад, если положительное - то вперед.

Рассмотрим на примере:

**Листинг 13.13 – Пример со смещением позиции указателя в файле**

|  |  |
| --- | --- |
| 1 | using System.IO; |
| 2 | using System.Text; |
| 3 | class Program |
| 4 | { |
| 5 | static void Main(string[] args) |
| 6 | { |
| 7 | string text = "hello world"; |
| 8 | // запись в файл |
| 9 | using (FileStream fstream = new FileStream(@"D:\note.dat", FileMode.OpenOrCreate)) |
| 10 | { |
| 11 | // преобразуем строку в байты |
| 12 | byte[] input = Encoding.Default.GetBytes(text); |
| 13 | // запись массива байтов в файл |
| 14 | fstream.Write(input, 0, input.Length); |
| 15 | Console.WriteLine("Текст записан в файл"); |
| 16 | // перемещаем указатель в конец файла, до конца файла- пять байт |
| 17 | fstream.Seek(-5, SeekOrigin.End); // минус 5 символов с конца потока |
| 18 | // считываем четыре символов с текущей позиции |
| 19 | byte[] output = new byte[4]; |
| 20 | fstream.Read(output, 0, output.Length); |
| 21 | // декодируем байты в строку |
| 22 | string textFromFile = Encoding.Default.GetString(output); |
| 23 | Console.WriteLine($"Текст из файла: {textFromFile}"); // worl |
| 24 | // заменим в файле слово world на слово house |
| 25 | string replaceText = "house"; |
| 26 | fstream.Seek(-5, SeekOrigin.End); // минус 5 символов с конца потока |
| 27 | input = Encoding.Default.GetBytes(replaceText); |
| 28 | fstream.Write(input, 0, input.Length); |
| 29 | // считываем весь файл |
| 30 | // возвращаем указатель в начало файла |
| 31 | fstream.Seek(0, SeekOrigin.Begin); |
| 32 | output = new byte[fstream.Length]; |
| 33 | fstream.Read(output, 0, output.Length); |
| 34 | // декодируем байты в строку |
| 35 | textFromFile = Encoding.Default.GetString(output); |
| 36 | Console.WriteLine($"Текст из файла: {textFromFile}"); // hello house |
| 37 | } |
| 38 | Console.Read(); |
| 39 | } |
| 40 | } |

Вызов

fstream.Seek(-5, SeekOrigin.End)

перемещает курсор потока в конец файлов назад на пять символов:



То есть после записи в новый файл строки "hello world" курсор будет стоять на позиции символа "w".

После этого считываем четыре байта начиная с символа "w". В данной кодировке 1 символ будет представлять 1 байт. Поэтому чтение 4 байтов будет эквивалентно чтению четырех символов: "worl".

Затем опять же перемещаемся в конец файла, не доходя до конца пять символов (то есть опять же с позиции символа "w"), и осуществляем запись строки "house". Таким образом, строка "house" заменяет строку "world".

В примерах выше для закрытия потока применяется конструкция using. После того как все операторы и выражения в блоке using отработают, объект FileStream уничтожается. Однако мы можем выбрать и другой способ – это метод Close():

fstream.Close()

## §13.6 Чтение и запись текстовых файлов. StreamReader и StreamWriter

Класс FileStream не очень удобно применять для работы с текстовыми файлами. К тому же для этого в пространстве System.IO определены специальные классы: StreamReader и StreamWriter.

Для записи в текстовый файл используется класс StreamWriter. Некоторые из его конструкторов, которые могут применяться для создания объекта StreamWriter:

**Таблица 13.9 – Конструкторы класса StreamWriter**

|  |  |
| --- | --- |
| **Конструктор** | **Описание** |
| StreamWriter(string path) | через параметр path передается путь к файлу, который будет связан с потоком |
| StreamWriter(string path, bool append) | параметр append указывает, надо ли добавлять в конец файла данные или же перезаписывать файл. Если равно true, то новые данные добавляются в конец файла. Если равно false, то файл перезаписывается заново |
| StreamWriter(string path, bool append, System.Text.Encoding encoding) | параметр encoding указывает на кодировку, которая будет применяться при записи |

Свою функциональность StreamWriter реализует через следующие методы:

**Таблица 13.10 – Методы класса StreamWriter**

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| int Close() | закрывает записываемый файл и освобождает все ресурсы |
| void Flush() | записывает в файл оставшиеся в буфере данные и очищает буфер |
| void Write(string value) | записывает в файл данные простейших типов, как int, double, char, string и т.д. |
| void WriteLine(string value) | также записывает данные, только после записи добавляет в файл символ окончания строки |

Рассмотрим запись в файл на примере:

**Листинг 13.14**

|  |  |
| --- | --- |
| 1 | using System; |
| 2 | using System.IO; |
| 3 | namespace HelloApp |
| 4 | { |
| 5 | class Program |
| 6 | { |
| 7 | static void Main(string[] args) |
| 8 | { |
| 9 | string writePath = @"c:\Dir1\file3.txt"; |
| 10 | string text = "Привет мир!\nПока мир..."; |
| 11 | try |
| 12 | { |
| 13 | using (StreamWriter sw = new StreamWriter(writePath, false, System.Text.Encoding.Default)) |
| 14 | { |
| 15 | sw.WriteLine(text); |
| 16 | } |
| 17 | using (StreamWriter sw = new StreamWriter(writePath, true, System.Text.Encoding.Default)) |
| 18 | { |
| 19 | sw.WriteLine("Дозапись"); |
| 20 | sw.Write(4.5); |
| 21 | } |
| 22 | Console.WriteLine("Запись выполнена"); |
| 23 | } |
| 24 | catch (Exception e) |
| 25 | { |
| 26 | Console.WriteLine(e.Message); |
| 27 | } |
| 28 | } |
| 29 | } |
| 30 | } |

В данном случае два раза создаем объект StreamWriter. В первом случае если файл существует, то он будет перезаписан. Если не существует, он будет создан. И в нее будет записан текст из переменной text. Во втором случае файл открывается для дозаписи, и будут записаны атомарные данные - строка и число. В обоих случаях будет использоваться кодировка по умолчанию.

По завершении программы в папке C://SomeDir мы сможем найти файл file.txt, который будет иметь следующие строки:

**Привет мир!**

**Пока мир...**

**Дозапись**

**4,5**

*Чтение из файла и StreamReader*

Класс StreamReader позволяет нам легко считывать весь текст или отдельные строки из текстового файла.

Некоторые из конструкторов класса StreamReader:

**Таблица 13.11 – Конструкторы класса StreamReader**

|  |  |
| --- | --- |
| **Конструктор** | **Описание** |
| StreamReader(string path) | через параметр path передается путь к считываемому файлу |
| StreamReader(string path, System.Text.Encoding encoding) | параметр encoding задает кодировку для чтения файла |

Среди методов StreamReader можно выделить следующие:

**Таблица 13.12 – Методы класса StreamReader**

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| void Close() | закрывает считываемый файл и освобождает все ресурсы |
| int Peek() | возвращает следующий доступный символ, если символов больше нет, то возвращает -1 |
| int Read() | считывает и возвращает следующий символ в численном представлении. Имеет перегруженную версию:  Read(char[] array, int index, int count)  , где array - массив, куда считываются символы, index - индекс в массиве array, начиная с которого записываются считываемые символы, и count - максимальное количество считываемых символов |
| string ReadLine() | считывает одну строку в файле |
| string ReadToEnd() | считывает весь текст из файла |

Сначала считаем текст полностью из ранее записанного файла:

**Листинг 13.15**

|  |  |
| --- | --- |
| 1 | using System; |
| 2 | using System.IO; |
| 3 | using System.Threading.Tasks; |
| 4 | namespace HelloApp |
| 5 | { |
| 6 | class Program |
| 7 | { |
| 8 | static void Main(string[] args) |
| 9 | { |
| 10 | string path = @"c:\Dir1\file3.txt"; |
| 11 | try |
| 12 | { |
| 13 | using (StreamReader sr = new StreamReader(path)) |
| 14 | { |
| 15 | Console.WriteLine(sr.ReadToEnd()); |
| 16 | } |
| 17 | } |
| 18 | catch (Exception e) |
| 19 | { |
| 20 | Console.WriteLine(e.Message); |
| 21 | } |
| 22 | } |
| 23 | } |
| 24 | } |

Считаем текст из файла построчно:

**Листинг 13.16**

|  |  |
| --- | --- |
| 1 | string path= @"c:\Dir1\file3.txt"; |
| 2 | using (StreamReader sr = new StreamReader(path, System.Text.Encoding.Default)) |
| 3 | { |
| 4 | string line; |
| 5 | while ((line = sr.ReadLine()) != null) |
| 6 | { |
| 7 | Console.WriteLine(line); |
| 8 | } |
| 9 | } |

В данном случае считываем построчно через цикл while:

while ((line = sr.ReadLine()) != null)

- сначала присваиваем переменной line результат функции sr.ReadLine(), а затем проверяем, не равна ли она null. Когда объект sr дойдет до конца файла и больше строк не останется, то метод sr.ReadLine() будет возвращать null.