# Работа с дисками

Так или иначе, но перед вами встанет задача по работе с файлами:

Вам потребуется либо записать результат работы своей программы, либо считать какие-то первичные данные, либо сохранить состояние программы. Для этих целей в C# есть несколько классов и способов работы с фалйами.

Начнём с самого верха с дисков:

Работа с дисками

Работу с файловой системой начнем с самого верхнего уровня - дисков. Для представления диска в пространстве имен System.IO имеется класс **DriveInfo**. Подробнее см. документацию Microsoft [[1]](#_Полезная_литература:)

|  |  |
| --- | --- |
| **Конструкторы** |  |
| DriveInfo(string driverName) |  |
| **Метод** |  |
| static GetDrives() | Статический метод, возвращает массив дисков |
| **Свойства** |  |
| AvailableFreeSpace | Указывает объем доступного свободного места на диске в байтах. |
| DriveFormat | Получает имя файловой системы, например NTFS или FAT32. |
| DriveType | Возвращает тип диска, например компакт-диск, съемный, сетевой или несъемный. |
| IsReady | Возвращает значение, указывающее, готов ли диск. |
| Name | Возвращает имя диска, например C:\. |
| RootDirectory | Возвращает корневой каталог диска. |
| TotalFreeSpace | Возвращает общий объем свободного места, доступного на диске, в байтах. |
| TotalSize | Возвращает общий размер места для хранения на диске в байтах. |
| VolumeLabel | Возвращает или задает метку тома диска. |

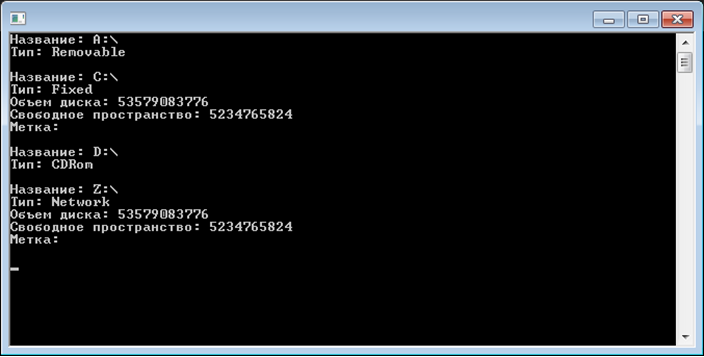
Для получения списка дисков не обязательно создавать объект класса DriveInfo, можно вызвать метод GetDrivers(), который вернёт массив объектов DriveInfo, каждый экземпляр которого будет соответствовать одному диску на компьютере, и предоставлять информацию о диске (т.е. каждый объект будет DriveInfo будет соответствовать одному диску)

Доступ ко всем остальным свойствам возможен только с существующим объектом.

Пример:

|  |
| --- |
| using System;  using System.IO;  namespace FileApp  {  class Program  {  static void Main(string[] args)  {  DriveInfo[] drives = DriveInfo.GetDrives();  foreach (DriveInfo drive in drives)  {  Console.WriteLine("Название: {0}", drive.Name);  Console.WriteLine("Тип: {0}", drive.DriveType);  if (drive.IsReady)  {  Console.WriteLine("Объем диска: {0}", drive.TotalSize);  Console.WriteLine("Свободное пространство: {0}", drive.TotalFreeSpace);  Console.WriteLine("Метка: {0}", drive.VolumeLabel);  }  Console.WriteLine();  }  Console.ReadLine();  }  }  } |

Вывод программы:



Работа с каталогами

Для работы с каталогами в пространстве имён System.IO определено два класса: **Directory** и **DirectoryInfo**.

Класс Directory является статическим и все определённые в нём методы вызываются без создания объекта, т.е. точно так же как вызываются методы WriteLine() у класса Console.

Класс Directory представлен множеством методов, мы рассмотрим лишь часть из них, остальные вы сможете найти на официальном сайте Microsoft [[2]](#_Полезная_литература:)

|  |  |
| --- | --- |
| **Методы** |  |
| CreateDirectory(path) | Создает все каталоги и подкаталоги по указанному пути, если они еще не существуют. |
| Delete(path) | Удаляет пустой каталог по заданному пути. |
| Delete(String, Boolean) | Удаляет заданный каталог и, при наличии соответствующей инструкции, все подкаталоги и файлы в нем. |
| Exists(path) | Определяет, указывает ли заданный путь на существующий каталог на диске. В случае если каталог существует возвращает true, если нет false. |
| GetDirectories(path) | Возвращает имена подкаталогов (включая пути) в указанном каталоге. |
| GetFiles(path) | Возвращает имена файлов (с указанием пути к ним) в указанном каталоге. |
| Move(sourceDirName, destDirName) | Перемещает файл или каталог со всем его содержимым в новое местоположение. |
| GetParent(path) | Извлекает родительский каталог, на который указывает абсолютный или относительный путь. |

Класс DirectoryInfo

Как и в случае с DriverInfo для начала работы с ним требует создать объект класса. Объект будет соответствовать определённому каталогу, указанному при инициализации. Рекомендуется использовать этот класс в случае если требуется детальная и длительная работа с каталогом.

Класс представлен множеством методов, рассмотрим здесь лишь небольшую часть, все остальные методы доступны на официальном сайте Microsoft [[3]](#_Полезная_литература:)

|  |  |
| --- | --- |
| **Конструктор** |  |
| DirectryInfo(path) | Создаёт экземпляр |
| **Свойства** |  |
| Exists | Получает значение, определяющее наличие каталога. |
| Parent | Получает родительский каталог заданного подкаталога. |
| Root | Получает корневую часть каталога. |
| **Методы** |  |
| Create() | Создает каталог. |
| CreateSubdirectory(path) | Создает один или несколько подкаталогов по заданному пути. Путь может быть задан относительно текущего экземпляра класса [DirectoryInfo](https://docs.microsoft.com/ru-ru/dotnet/api/system.io.directoryinfo?view=netframework-4.6). |
| Delete() | Удаляет этот [DirectoryInfo](https://docs.microsoft.com/ru-ru/dotnet/api/system.io.directoryinfo?view=netframework-4.6), если он пуст. |
| Delete(bool) | Удаляет данный экземпляр [DirectoryInfo](https://docs.microsoft.com/ru-ru/dotnet/api/system.io.directoryinfo?view=netframework-4.6), указывая, следует ли также удалить подкаталоги и файлы. |
| GetDirectories() | Возвращает подкаталоги текущего каталога. |
| GetFiles() | Возвращает список файлов текущего каталога. |
| MoveTo(destDirName) | Перемещает экземпляр [DirectoryInfo](https://docs.microsoft.com/ru-ru/dotnet/api/system.io.directoryinfo?view=netframework-4.6) и его содержимое в местоположение, на которое указывает новый путь. |

Примеры использования:

Данная программа выведет все каталоги и файлы которая лежат на в корневом каталоге диска C:\

|  |
| --- |
| string dirName = "C:\\";    if (Directory.Exists(dirName))  {  Console.WriteLine("Подкаталоги:");  string[] dirs = Directory.GetDirectories(dirName);  foreach (string s in dirs)  {  Console.WriteLine(s);  }  Console.WriteLine();  Console.WriteLine("Файлы:");  string[] files = Directory.GetFiles(dirName);  foreach (string s in files)  {  Console.WriteLine(s);  }  } |

Кстати обратите внимание на написание C:\\

Это сделано для того чтобы исключить управляющие символы, которые как раз начинаются с обратного слеша, к примеру «\n – перенос картетки, \t – табуляция и т.д. Или мы можем начинать строку с символа @, @”C:\Temp”. Данный символ говорит компиляторы чтобы следующую строку он обрабатывал как есть, без управляющих последовательностей.

Создание каталога:

|  |
| --- |
| string path = @"C:\SomeDir";  string subpath = @"subDir\subSubDir";  DirectoryInfo dirInfo = new DirectoryInfo(path);  if (!dirInfo.Exists)  {  dirInfo.Create();  }  dirInfo.CreateSubdirectory(subpath); |

Тут мы создаём объект класса DirectoryInfo по адресу C:\SomeDir, проверяем существует ли такой каталог и если нет, то создаём его. Затем создаём в нём два подкаталога.

Вывод информации о каталоге:

|  |
| --- |
| string dirName = "C:\\Program Files";  DirectoryInfo dirInfo = new DirectoryInfo(dirName);  Console.WriteLine("Название каталога: {0}", dirInfo.Name);  Console.WriteLine("Полное название каталога: {0}", dirInfo.FullName);  Console.WriteLine("Время создания каталога: {0}", dirInfo.CreationTime);  Console.WriteLine("Корневой каталог: {0}", dirInfo.Root); |

Удаление каталога:

Для удаления каталога у класса DirectoryInfo есть два метода: Delete() и Delete(bool)

Отличаются они в том как удаляют каталоги:

Delete() удаляет только пустой каталог, Delete(bool) удаляет каталог со всем содержимым. В случае если применять Delete() и каталог не пустой, то программа выдаст ошибку.

|  |
| --- |
| string dirName = @"C:\SomeFolder";    try  {  DirectoryInfo dirInfo = new DirectoryInfo(dirName);  dirInfo.Delete(true);  }  catch (Exception ex)  {  Console.WriteLine(ex.Message);  } |

Или через класс Directory

|  |
| --- |
| string dirName = @"C:\SomeFolder";  Directory.Delete(dirName, true); |

Перемещение каталога:

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

При перемещении надо учитывать, что новый каталог, в который мы хотим переместить всё содержимое старого каталога, не должен существовать (т.е. мы его создаём при перемещении).

# Работа с файлами

Для работы с файлами в C# есть классы подобные тем которые предназначены для работы с каталогами: **File** и **FileInfo.** Ихвозможности в целом похожи, только расширены для работы с файлами. С их помощью можно: создавать, удалять, перемещать файлы, получать их свойства и многое другое.

Как и в случае с Directory и DirectoryInfo, класс File является статическим, а класс FileInfo требует объекта. Здесь рассмотрим наиболее частые методы обоих классов, полный список методов приведён в официальной документации класс File [[4]](#_Полезная_литература:), класс FileInfo [[5]](#_Полезная_литература:)

Методы класса File:

|  |  |
| --- | --- |
| **Методы** |  |
| Copy(sourseFile, destFile) | Копирует существующий файл в новый файл. Перезапись файла с тем же именем не разрешена. |
| Copy(sourseFile, destFile, bool) | Копирует существующий файл в новый файл. Перезапись файла с тем же именем разрешена. |
|  |  |
| Create(PathFile) | Создает или перезаписывает файл в указанном пути. |
| Delete(PathFile) | Удаляет указанный файл. |
| Move(sourseFile, destFile) | Перемещает заданный файл в новое местоположение и разрешает переименование файла. |
| Exists(file) | Определяет, существует ли заданный файл. |

Методы класс FileInfo

|  |  |
| --- | --- |
| **Конструктор** |  |
| FileInfo(file) | Выполняет инициализацию нового экземпляра класса [FileInfo](https://docs.microsoft.com/ru-ru/dotnet/api/system.io.fileinfo?view=netframework-4.6), который служит оболочкой для пути файла. |
| **Свойства** |  |
| Directory | Свойство извлекает объект, представляющий родительский каталог файла. (возвращает объект DirectoryInfo) |
| DirectoryName | Получает строку, представляющую полный путь к каталогу. |
| Exists | Получает значение, показывающее, существует ли файл. |
| Length | Получает размер текущего файла в байтах. |
| Extension | Получает строку, содержащую расширение файла. |
| Name | Возвращает имя файла. |
| FullName | Получает полный путь к каталогу или файлу. |
| **Методы** |  |
| CopyTo(path) | Копирует существующий файл в новый файл и запрещает перезапись существующего файла. |
| CopyTo(path, bool) | Копирует существующий файл в новый файл и разрешает перезапись существующего файла. |
| Create() | Создает файл. |
| Delete() | Удаляет файл без возможности восстановления. |
| MoveTo(destFileName) | Перемещает заданный файл в новое местоположение и разрешает переименование файла. |

Получение информации о файле:

|  |
| --- |
| string path = @"C:\apache\hta.txt";  FileInfo fileInf = new FileInfo(path);  if (fileInf.Exists)  {  Console.WriteLine("Имя файла: {0}", fileInf.Name);  Console.WriteLine("Время создания: {0}", fileInf.CreationTime);  Console.WriteLine("Размер: {0}", fileInf.Length);  } |

Удаление файла:

|  |
| --- |
| string path = @"C:\apache\hta.txt";  FileInfo fileInf = new FileInfo(path);  if (fileInf.Exists)  {  fileInf.Delete();  // альтернатива с помощью класса File  // File.Delete(path);  } |

Перемещение файла:

|  |
| --- |
| string path = @"C:\apache\hta.txt";  string newPath = @"C:\SomeDir\hta.txt";  FileInfo fileInf = new FileInfo(path);  if (fileInf.Exists)  {  fileInf.MoveTo(newPath);  // альтернатива с помощью класса File  // File.Move(path, newPath);  } |

Копирование файла:

|  |
| --- |
| string path = @"C:\apache\hta.txt";  string newPath = @"C:\SomeDir\hta.txt";  FileInfo fileInf = new FileInfo(path);  if (fileInf.Exists)  {  fileInf.CopyTo(newPath, true);  // альтернатива с помощью класса File  // File.Copy(path, newPath, true);  } |

У классов File и FileInfo есть по два метода копирования: CopyTo(path), CopyTo(path, bool) и Copy(soursePath, destPath), Copy(soursePath, destPath, bool). Отличие между ними в булевой переменной, которая указывает возможно перезаписи файла, т.е. при копировании файла, если файл «куда» уже существует, то в первом случае (без переменной bool) будет выдана ошибка, а во втором случае файл перезапишется исходным файлом.

# Чтение и запись файла.

Класс FileStream [[6]](#_Полезная_литература:) предоставляет возможности по считыванию и записи данных в файл.

Рассмотрим наиболее важные методы и свойства:

|  |  |
| --- | --- |
| **Конструктор** |  |
| [FileStream(String, FileMode)](https://docs.microsoft.com/ru-ru/dotnet/api/system.io.filestream.-ctor?view=netframework-4.6#System_IO_FileStream__ctor_System_String_System_IO_FileMode_) | Инициализирует новый экземпляр класса [FileStream](https://docs.microsoft.com/ru-ru/dotnet/api/system.io.filestream?view=netframework-4.6) указанным путем и режимом создания. |
| **Свойства** |  |
| Length | Возвращает длину потока в байтах. |
| Position | Возвращает или задает текущую позицию этого потока. |
| **Методы** |  |
| Read(byte[] array, int offset, int count) | Выполняет чтение блока байтов из потока и запись данных в заданный буфер. |
| **array** - массив байтов, куда будут помещены считываемые из файла данные |
| **offset** представляет смещение в байтах в массиве array, в который считанные байты будут помещены |
| **count** - максимальное число байтов, предназначенных для чтения. Если в файле находится меньшее количество байтов, то все они будут считаны. |
| Write(byte[] array, int offset, int count) | Записывает блок байтов в файловый поток. |
| **array** - массив байтов, откуда данные будут записываться в файла |
| **offset** - смещение в байтах в массиве array, откуда начинается запись байтов в поток |
| **count** - максимальное число байтов, предназначенных для записи |
| Seek(long offset, SeekOrigin origin) | Устанавливает текущее положение этого потока на заданное значение. |

Пример:

|  |
| --- |
| using System;  using System.IO;  namespace Lesson3  {  class Program  {  static void Main(string[] args)  {  Console.WriteLine("Введите строку для записи в файл:");  string text = Console.ReadLine();  // Запись в файл  FileStream fstream = new FileStream(@"C:\Test\test.txt", FileMode.OpenOrCreate);  //Преобразуем строку в байты  byte[] arr = System.Text.Encoding.Default.GetBytes(text);  fstream.Write(arr, 0, arr.Length);  // Закрываем поток  fstream.Close();  //Чтение из файла  fstream = File.OpenRead(@"C:\Test\test.txt");  arr = new byte[fstream.Length];  //Считываем данные  fstream.Read(arr, 0, arr.Length);  // Декодируем из байтов в строку  string textFromFile = System.Text.Encoding.Default.GetString(arr);  // Закрываем поток  fstream.Close();  Console.WriteLine(textFromFile);  Console.Read();  }  }  } |

Рассмотрим подробнее:

И при чтении и при записи происходит открытие потока, при записи открытие происходит через инициализацию объекта и передаче специального аргумента FileMode.OpenOrCreate, а при чтении открытие происходит через File.OpenRead(Path).

FileMode это enum которое указывает на режим доступа к файлу и может принимать следующие значения:

* **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.

Открытие означает, что система блокирует доступ к файлу из других источников, кроме как от этого потока, поэтому в конце как чтения, так и записи стоит закрытие потока fstream.Close();

Если бы мы после записи не закрыли поток, то не смогли бы обратиться к чтению этого файла.

Поэтому есть другая форма записи гарантирующая закрытие потока после чтения/записи:

|  |
| --- |
| using System;  using System.IO;  namespace Lesson3  {  class Program  {  static void Main(string[] args)  {  Console.WriteLine("Введите строку для записи в файл:");  string text = Console.ReadLine();  // запись в файл  using (FileStream fstream = new FileStream(@"C:\SomeDir\noname\note.txt", FileMode.OpenOrCreate))  {  // преобразуем строку в байты  byte[] array = System.Text.Encoding.Default.GetBytes(text);  // запись массива байтов в файл  fstream.Write(array, 0, array.Length);  Console.WriteLine("Текст записан в файл");  }  // чтение из файла  using (FileStream fstream = File.OpenRead(@"C:\SomeDir\noname\note.txt"))  {  byte[] array = new byte[fstream.Length];  // считываем данные  fstream.Read(array, 0, array.Length);  // декодируем байты в строку  string textFromFile = System.Text.Encoding.Default.GetString(array);  Console.WriteLine("Текст из файла: {0}", textFromFile);  }  Console.Read();  }  }  } |

Здесь используется конструкция:

|  |
| --- |
| using (FileStream fstream = File.OpenRead(@"C:\SomeDir\noname\note.txt"))  {  byte[] array = new byte[fstream.Length];  fstream.Read(array, 0, array.Length);  string textFromFile = System.Text.Encoding.Default.GetString(array);  Console.WriteLine("Текст из файла: {0}", textFromFile);  } |

В блоке using объявляем открытие потока и в самом блоке делаем чтение/запись. Как только программа выходит за пределы блока using поток закрывается автоматически. Такая форма является предпочтительней.

Так как FileStream пишет в файл бинарном виде мы используем кодировку Encoding.Default. В противном случае нам будет сложно понять, что записалось в файл, если открыть его текстовым редактором.

Произвольный доступ к файлам

Нередко бинарные файлы представляют определенную стрктуру. И, зная эту структуру, мы можем взять из файла нужную порцию информации или наоброт 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:

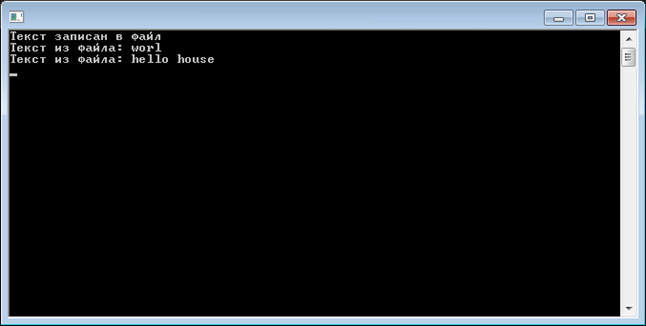
* **SeekOrigin.Begin**: начало файла
* **SeekOrigin.End**: конец файла
* **SeekOrigin.Current**: текущая позиция в файле

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

Рассмотрим на примере:

|  |
| --- |
| static void Main(string[] args)  {  string text = "hello world";  // запись в файл  using (FileStream fstream = new FileStream(@"D:\note.dat", FileMode.OpenOrCreate))  {  // преобразуем строку в байты  byte[] input = Encoding.Default.GetBytes(text);  // запись массива байтов в файл  fstream.Write(input, 0, input.Length);  Console.WriteLine("Текст записан в файл");  // перемещаем указатель в конец файла, до конца файла- пять байт  fstream.Seek(-5, SeekOrigin.End); // минус 5 символов с конца потока  // считываем четыре символов с текущей позиции  byte[] output = new byte[4];  fstream.Read(output, 0, output.Length);  // декодируем байты в строку  string textFromFile = Encoding.Default.GetString(output);  Console.WriteLine("Текст из файла: {0}", textFromFile); // worl  // заменим в файле слово world на слово house  string replaceText = "house";  fstream.Seek(-5, SeekOrigin.End); // минус 5 символов с конца потока  input = Encoding.Default.GetBytes(replaceText);  fstream.Write(input, 0, input.Length);  // считываем весь файл  // возвращаем указатель в начало файла  fstream.Seek(0, SeekOrigin.Begin);  output = new byte[fstream.Length];  fstream.Read(output, 0, output.Length);  // декодируем байты в строку  textFromFile = Encoding.Default.GetString(output);  Console.WriteLine("Текст из файла: {0}", textFromFile); // hello house  }  Console.Read();  } |

Вывод программы:



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 сиволов: "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".

Для работы с текстовыми файлами предназначены классы StreamReader [[7]](#_Полезная_литература:) и StreamWriter [[8]](#_Полезная_литература:)

Методы StreamReader

|  |  |
| --- | --- |
| **Конструктор** |  |
| [StreamReader(String, Encoding)](https://docs.microsoft.com/ru-ru/dotnet/api/system.io.streamreader.-ctor?view=netframework-4.6#System_IO_StreamReader__ctor_System_String_System_Text_Encoding_) | Инициализирует новый экземпляр класса [StreamReader](https://docs.microsoft.com/ru-ru/dotnet/api/system.io.streamreader?view=netframework-4.6) для заданного имени файла, используя указанную кодировку символов. |
| **Методы** |  |
| Close() | Закрывает объект [StreamReader](https://docs.microsoft.com/ru-ru/dotnet/api/system.io.streamreader?view=netframework-4.6) и основной поток и освобождает все системные ресурсы, связанные с устройством чтения. |
| Peek() | Возвращает следующий доступный символ, но не использует его, если символов больше нет, то возвращает -1 |
| Read() | Выполняет чтение следующего символа из входного потока и перемещает положение символа на одну позицию вперед. |
| [Read(Char[], Int32, Int32)](https://docs.microsoft.com/ru-ru/dotnet/api/system.io.streamreader.read?view=netframework-4.6#System_IO_StreamReader_Read_System_Char___System_Int32_System_Int32_) | Считывает заданное максимальное количество символов из текущего потока в буфер начиная с заданного индекса. |
| ReadLine() | Выполняет чтение строки символов из текущего потока и возвращает данные в виде строки. |
| ReadToEnd() | Считывает все символы, начиная с текущей позиции до конца потока. |

|  |
| --- |
| string path = @"C:\Test\hta.txt";  Console.WriteLine("\*\*\*\*\*\*считываем весь файл\*\*\*\*\*\*\*\*");  using (StreamReader sr = new StreamReader(path))  {  Console.WriteLine(sr.ReadToEnd());  }  Console.WriteLine();  Console.WriteLine("\*\*\*\*\*\*считываем построчно\*\*\*\*\*\*\*\*");  using (StreamReader sr = new StreamReader(path, System.Text.Encoding.Default))  {  string line;  while ((line = sr.ReadLine()) != null)  {  Console.WriteLine(line);  }  }  Console.WriteLine();  Console.WriteLine("\*\*\*\*\*\*считываем блоками\*\*\*\*\*\*\*\*");  using (StreamReader sr = new StreamReader(path, System.Text.Encoding.Default))  {  char[] array = new char[4];  // считываем 4 символа  sr.Read(array, 0, 4);  Console.WriteLine(array);  }  Console.Read(); |

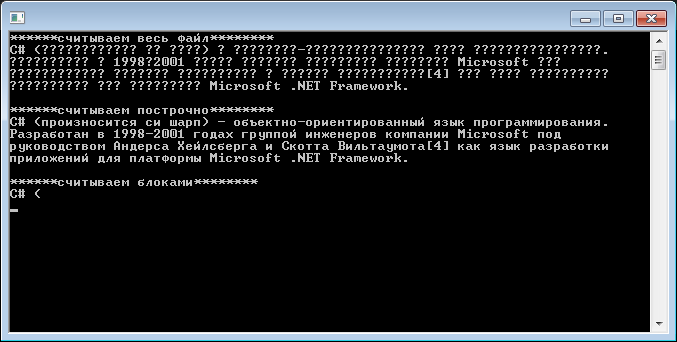
Как и в случае с классом FileStream здесь используется конструкция using.

В первом случае мы разом считываем весь текст с помощью метода ReadToEnd().

Во втором 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.

В третьем случае считываем в массив четыре символа.

Обратите внимание, что в последних двух случаях в конструкторе StreamReader указывалась кодировка System.Text.Encoding.Default. Свойство Default класса Encoding получает кодировку для текущей кодовой страницы ANSI. Также через другие свойства мы можем указать другие кодировки. Если кодировка не указана, то при чтении используется UTF8. Иногда важно указывать кодировку, так как она может отличаться от UTF8, и тогда мы получим некорректный вывод. Например:



Методы StreamWriter

|  |  |
| --- | --- |
| **Конструктор** |  |
| [StreamWriter(String, Encoding)](https://docs.microsoft.com/ru-ru/dotnet/api/system.io.streamreader.-ctor?view=netframework-4.6#System_IO_StreamReader__ctor_System_String_System_Text_Encoding_) | Инициализирует новый экземпляр класса [StreamWriter](https://docs.microsoft.com/ru-ru/dotnet/api/system.io.streamwriter?view=netframework-4.6) для указанного потока, используя заданную кодировку и размер буфера по умолчанию. |
| **Методы** |  |
| Close() | Закрывает текущий объект StreamWriter и базовый поток. |
| Flush() | Очищает все буферы для текущего средства записи и вызывает запись всех данных буфера в основной поток. |
| Write() | Запись в поток, на вход принимает как строки, так и другие типы данных. |
| WriteLine() | также записывает данные, только после записи добавляет в файл символ окончания строки |

|  |
| --- |
| string readPath = @"C:\Test\hta.txt";  string writePath = @"C:\Test\ath.txt";  string text = "";  using (StreamReader sr = new StreamReader(readPath, System.Text.Encoding.Default))  {  text = sr.ReadToEnd();  }  using (StreamWriter sw = new StreamWriter(writePath, false, System.Text.Encoding.Default))  {  sw.WriteLine(text);  }  using (StreamWriter sw = new StreamWriter(writePath, true, System.Text.Encoding.Default))  {  sw.WriteLine("Дозапись");  sw.Write(4.5);  } |

Здесь сначала мы считываем файл в переменную text, а затем записываем эту переменную в файл, а затем через объект StreamWriter записываем в новый файл.

Класс StreamWriter имеет несколько конструкторов. Здесь мы использовали один из них: new StreamWriter(writePath, false, System.Text.Encoding.Default). В качестве первого параметра передается путь к записываемому файлу. Второй параметр представляет булевую переменную, которая определяет, будет файл дозаписываться или перезаписываться. Если этот параметр равен true, то новые данные добавляются в конце к уже имеющимся данным. Если false, то файл перезаписывается. И если в первом случае файл перезаписывается, то во втором делается дозапись в конец файла.

Третий параметр указывает кодировку, в которой записывается файл.

Классы BinaryWriter [[9]](#_Полезная_литература:) и BinaryReader [[10]](#_Полезная_литература:)

Для работы с бинарными файлами предназначена пара классов BinaryWriter и BinaryReader. Эти классы позволяют читать и записывать данные в двоичном формате.

Методы BinaryWriter

|  |  |
| --- | --- |
| **Конструктор** |  |
| [BinaryWriter(Stream)](https://docs.microsoft.com/ru-ru/dotnet/api/system.io.binarywriter.-ctor?view=netframework-4.6#System_IO_BinaryWriter__ctor_System_IO_Stream_) | Инициализирует новый экземпляр класса [BinaryWriter](https://docs.microsoft.com/ru-ru/dotnet/api/system.io.binarywriter?view=netframework-4.6) на основании указанного потока с использованием кодировки UTF-8. |
| **Методы** |  |
| Close() | Закрывает текущий [BinaryWriter](https://docs.microsoft.com/ru-ru/dotnet/api/system.io.binarywriter?view=netframework-4.6) и базовый поток. |
| Flush() | Очищает все буферы текущего модуля записи и вызывает немедленную запись всех буферизованных данных на базовое устройство. |
| Write() | записывает данные в поток |

Методы BinaryReader

|  |  |
| --- | --- |
| **Конструктор** |  |
| [BinaryReader(Stream)](https://docs.microsoft.com/ru-ru/dotnet/api/system.io.binaryreader.-ctor?view=netframework-4.7.2#System_IO_BinaryReader__ctor_System_IO_Stream_) | Инициализирует новый экземпляр класса [BinaryReader](https://docs.microsoft.com/ru-ru/dotnet/api/system.io.binaryreader?view=netframework-4.7.2) на основании указанного потока с использованием кодировки UTF-8. |
| **Методы** |  |
| Close() | Закрывает текущий поток чтения и связанный с ним базовый поток. |
| ReadBoolean() | Считывает значение Boolean из текущего потока и перемещает текущую позицию в потоке на один байт вперед. |
| ReadByte() | Считывает из текущего потока следующий байт и перемещает текущую позицию в потоке на один байт вперед. |
| ReadChar() | Считывает следующий знак из текущего потока и изменяет текущую позицию в потоке в соответствии с используемым значением Encoding и конкретным знаком в потоке, чтение которого выполняется в настоящий момент. |
| ReadDecimal() | Считывает десятичное значение из текущего потока и перемещает текущую позицию в потоке на шестнадцать байтов вперед. |
| ReadDouble() | Считывает число с плавающей запятой длиной 8 байт из текущего потока и перемещает текущую позицию в потоке на восемь байт вперед. |
| ReadInt16() | Считывает целое число со знаком длиной 2 байта из текущего потока и перемещает текущую позицию в потоке на два байта вперед. |
| ReadInt32() | Считывает целое число со знаком длиной 4 байта из текущего потока и перемещает текущую позицию в потоке на четыре байта вперед. |
| ReadInt64() | Считывает целое число со знаком длиной 8 байта из текущего потока и перемещает текущую позицию в потоке на восемь байт вперед. |
| ReadSingle() | Считывает число с плавающей запятой длиной 4 байта из текущего потока и перемещает текущую позицию в потоке на четыре байта вперед. |
| ReadString() | Считывает строку из текущего потока. Строка предваряется значением длины строки, которое закодировано как целое число блоками по семь битов. |

С чтением бинарных данных все просто: соответствующий метод считывает данные определенного типа и перемещает указатель на размер этого типа в байтах, например, значение типа int занимает 4 байта, поэтому BinaryReader считает 4 байта и переместит указать на эти 4 байта.

Посмотрим на реальной задаче применение этих классов. Попробуем с их помощью записывать и считывать из файла массив структур:

|  |
| --- |
| struct State  {  public string name;  public string capital;  public int area;  public double people;  public State(string n, string c, int a, double p)  {  name = n;  capital = c;  people = p;  area = a;  }  }  class Program  {  static void Main(string[] args)  {  State[] states = new State[2];  states[0] = new State("Германия", "Берлин", 357168, 80.8);  states[1] = new State("Франция", "Париж", 640679, 64.7);  string path = @"C:\Test\states.dat";  // создаем объект BinaryWriter  using (BinaryWriter writer = new BinaryWriter(File.Open(path, FileMode.OpenOrCreate)))  {  // записываем в файл значение каждого поля структуры  foreach (State s in states)  {  writer.Write(s.name);  writer.Write(s.capital);  writer.Write(s.area);  writer.Write(s.people);  }  }  // создаем объект BinaryReader  using (BinaryReader reader = new BinaryReader(File.Open(path, FileMode.Open)))  {  // пока не достигнут конец файла  // считываем каждое значение из файла  while (reader.PeekChar() > -1)  {  string name = reader.ReadString();  string capital = reader.ReadString();  int area = reader.ReadInt32();  double population = reader.ReadDouble();  Console.WriteLine("Страна: {0} столица: {1} площадь {2} кв. км численность населения: {3} млн. чел.",  name, capital, area, population);  }  }  Console.ReadLine();  }  } |

Итак, у нас есть структура State с некоторым набором полей. В основной программе создаем массив структур и записываем с помощью BinaryWriter. Этот класс в качестве параметра в конструкторе принимает объект Stream, который создается вызовом File.Open(path, FileMode.OpenOrCreate).

Затем в цикле пробегаемся по массиву структур и записываем каждое поле структуры в поток. В том порядке, в каком эти значения полей записываются, в том порядке они и будут размещаться в файле.

Затем считываем из записанного файла. Конструктор класса BinaryReader также в качестве параметра принимает объект потока, только в данном случае устанавливаем в качестве режима FileMode.Open: new BinaryReader(File.Open(path, FileMode.Open))

В цикле while считываем данные. Чтобы узнать окончание потока, вызываем метод PeekChar(). Этот метод считывает следующий символ и возвращает его числовое представление. Если символ отсутствует, то метод возвращает -1, что будет означать, что мы достигли конца файла.

В цикле последовательно считываем значения поле структур в том же порядке, в каком они записывались.

Таким образом, классы BinaryWriter и BinaryReader очень удобны для работы с бинарными файлами, особенно когда нам известна структура этих файлов. В то же время для хранения и считывания более комплексных объектов, например, объектов классов, лучше подходит другое решение - сериализация.

# Полезная литература:

1. Класс DriveInfo [https://docs.microsoft.com/ru-ru/dotnet/api/system.io.driveinfo](https://docs.microsoft.com/ru-ru/dotnet/api/system.io.driveinfo?view=netframework-4.6)
2. Класс Directory [https://docs.microsoft.com/ru-ru/dotnet/api/system.io.directory](https://docs.microsoft.com/ru-ru/dotnet/api/system.io.directory?view=netframework-4.6)
3. Класс DirectoryInfo [https://docs.microsoft.com/ru-ru/dotnet/api/system.io.directoryinfo](https://docs.microsoft.com/ru-ru/dotnet/api/system.io.directoryinfo?view=netframework-4.6)
4. Класс File [https://docs.microsoft.com/ru-ru/dotnet/api/system.io.file](https://docs.microsoft.com/ru-ru/dotnet/api/system.io.file?view=netframework-4.6)
5. Класс FileInfo [https://docs.microsoft.com/ru-ru/dotnet/api/system.io.fileinfo](https://docs.microsoft.com/ru-ru/dotnet/api/system.io.fileinfo?view=netframework-4.6)
6. Класс FileStream [https://docs.microsoft.com/ru-ru/dotnet/api/system.io.filestream](https://docs.microsoft.com/ru-ru/dotnet/api/system.io.filestream?view=netframework-4.6)
7. Класс StreamReader [https://docs.microsoft.com/ru-ru/dotnet/api/system.io.streamreader](https://docs.microsoft.com/ru-ru/dotnet/api/system.io.streamreader?view=netframework-4.6)
8. Класс StreamWriter <https://docs.microsoft.com/ru-ru/dotnet/api/system.io.streamwriter>
9. Класс BinaryWriter <https://docs.microsoft.com/ru-ru/dotnet/api/system.io.binarywriter>
10. Класс BinaryReader <https://docs.microsoft.com/ru-ru/dotnet/api/system.io.binaryreader>