

# Работа с файловой системой

# Основные понятия

Под **файлом** подразумевается именованная информация на внешнем носителе.

Логически файл можно представить как конечное количество последовательных байтов.

Передача данных с внешнего устройства в оперативную память называется **чтением**, или вводом, обратный процесс — **записью**, или выводом.

Обмен данных реализуется с помощью **потоков**.

**Поток (stream)** — это абстрактное понятие, относящееся к любому переносу данных от источника к приемнику. Поток определяется как последовательность байтов и не зависит от конкретного устройства, с которым производится обмен.

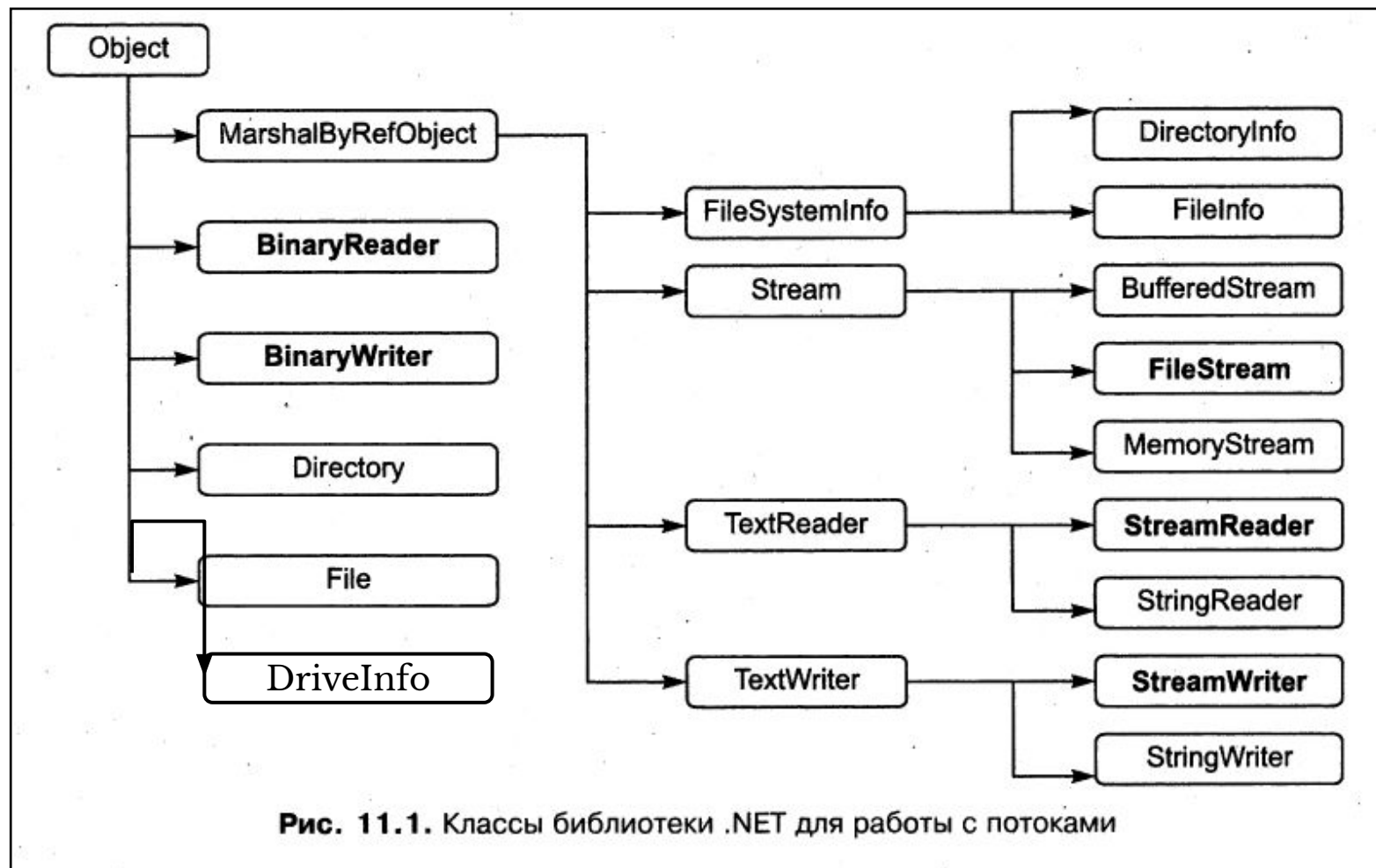
Обмен с потоком для повышения скорости передачи производится через специальную область оперативной памяти — **буфер**.

**При записи** в файл вся **информация** сначала **направляется в буфер** и там накапливается, пока буфер не заполнится. После этого происходит передача данных на внешнее устройство.

**При чтении** из файла **данные** вначале **считываются в буфер**, что позволяет быстро и эффективно обмениваться информацией с внешними устройствами.

# Пространство имен System.IO

Пространство имен System.IO в .NET содержит библиотеки базовых классов, предназначенным для файлового ввода-вывода, а также ввода-вывода из памяти.



# Описание классов для работы с файлами

Класс	Описание
<b>BinaryReader BinaryWriter</b>	Чтение и запись простых встроенных типов (целочисленных, логических, строковых и т.п.) во внутренней форме представления
<b>BufferedStream</b>	Временное хранение потока байтов (например для последующего переноса в постоянное хранилище)
<b>Directory, DirectoryInfo File, FileInfo</b>	Работа с каталогами или физическими файлами: создание, удаление, получение свойств
<b>FileStream</b>	Произвольный доступ к файлу, представленному как поток байтов
<b>MemoryStream</b>	Произвольный доступ к потоку байтов в оперативной памяти
<b>StreamWriter StreamReader</b>	Чтение из файла и запись в файл текстовой информации (произвольный доступ не поддерживается)
<b>DriveInfo</b>	Предоставляет доступ к сведениям на диске

# Уровни чтения и записи файлов

Обмен с внешними устройствами можно выполнять на уровне:

- ☐ Двоичного представления данных (BinaryReader, BinaryWriter).
- ☐ Байтов (FileStream).
- ☐ Текста, то есть символов (StreamWriter, StreamReader).

Использование классов файловых потоков предполагает следующие операции.

1. Создание потока и связывание его с физическим файлом.
2. Обмен (ввод-вывод).
3. Закрытие файла.

Потоки для чтения и записи после завершения работы с ними необходимо закрыть с помощью метода Close(), который вызывает метод Dispose() для освобождения всех ресурсов, связанных с файлом.

# FileStream

Для создания байтового потока при использовании файлов используется класс **FileStream**. Этот класс считается производным от класса **Stream** и использует все его функции.

Классы потоков, в том числе класс `FileStream`, определены в пространстве имен **System.IO**. Поэтому в начале любой программы, использующей файлы, должен стоять следующий программный код:

```
using System.IO;
```

Для формирования байтового потока, связанного с файлом, создается объект класса `FileStream`. В этом классе определено несколько конструкторов.

# FileStream

Ниже приведена запись для байтового потока данных:

**FileStream** (**string** *путь*, **FileMode** *режим*, **FileAccess**  
*режим*)

где *путь* адрес и имя открываемого файла, т.е. можно указать полный маршрут к файлу;

а *режим* — способ открытия файла, он должен соответствовать одному из режимов **FileMode**, которые приведены далее, на сл. странице.

Эти режимы класса **FileStream** открывают файл для чтения или записи данных. Отличительным от них способом является режим **FileMode.Append**, который открывает файл только для дозаписи данных к нему.

# Доступ к

**Последовательный** – **файтам** очередной элемент можно прочитать (записать) только после аналогичной операции с предыдущим элементом.

**Произвольный (прямой)** – выполняется чтение (запись) произвольного элемента по заданному адресу.

Режимы доступа к файлу в перечислении **FileAccess**

Значение	Описание
Read	Открыть файл только для чтения
ReadWrite	Открыть файл для чтения и записи
Write	Открыть файл только для записи

Режимы открытия файла в перечислении **FileMode**

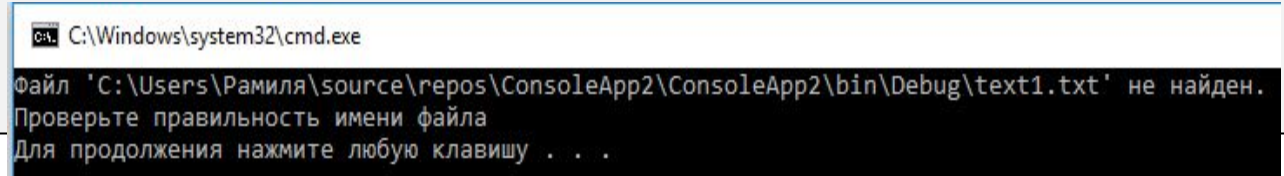
Значение	Описание
Append	Открыть файл, если он существует, и установить текущий указатель в конец файла. Если файл не существует, создать новый файл
Create	Создать новый файл. Если в каталоге уже существует файл с таким же именем, он будет стерт
CreateNew	Создать новый файл. Если в каталоге уже существует файл с таким же именем, возникает исключение <code>IOException</code>
Open	Открыть существующий файл
OpenOrCreate	Открыть файл, если он существует. Если нет, создать файл с таким именем
Truncate	Открыть существующий файл. После открытия он должен быть обрезан до нулевой длины



# Исключения

Исключение	Описание
<b>FileNotFoundException</b>	файл с указанным именем в указанном каталоге не существует
<b>DirectoryNotFoundException</b>	не существует указанный каталог
<b>ArgumentException</b>	неверно задан режим открытия файла
<b>IOException</b>	Файл не открывается из-за ошибок ввода-вывода

```
static void Main(string[] args)
{
    try
    {
        FileStream f = new FileStream("text1.txt", FileMode.Open, FileAccess.ReadWrite);
        f.Close();
    }
    catch (FileNotFoundException e)
    {
        Console.WriteLine(e.Message);
        Console.WriteLine("Проверьте правильность имени файла");
        return;
    }
    catch (Exception e)
    {
        Console.WriteLine("Error: " + e.Message);
        return;
    }
}
```



The screenshot shows a Windows command prompt window with the title bar "C:\Windows\system32\cmd.exe". The command prompt displays the following text: "Файл 'C:\Users\Рафия\source\repos\ConsoleApp2\ConsoleApp2\bin\Debug\text1.txt' не найден. Проверьте правильность имени файла. Для продолжения нажмите любую клавишу . . .". This is the output of the C# program shown in the code block above, which attempts to open a file that does not exist.

```
namespace ConsoleApp1
```

```
{ class Program
```

```
{ static void Main(string[] args)
```

```
{
```

```
    FileStream f = new FileStream("text.txt", FileMode.Create, FileAccess.ReadWrite);
```

```
    f.WriteByte(100); // записывает число 100 в начало файла
```

```
    byte[] x = new byte[10];
```

```
    for (byte i=0;i<10;++i)
```

```
    {
```

```
        x[i] = (byte)(10 - i);
```

```
        f.WriteByte(i); // записывает 10 чисел от 0 до 9
```

```
    }
```

```
    f.Write(x, 0, 5); // записывает первые 5 элементов массива
```

```
    byte[] y = new byte[20];
```

```
    f.Seek(0, SeekOrigin.Begin); // установка указателя на начало
```

```
    f.Read(y, 0, 20); // чтение из файла в массив
```

```
    foreach (byte elem in y) Console.Write(" " + elem);
```

```
    Console.WriteLine();
```

```
    f.Seek(5, SeekOrigin.Begin); // установка указателя на 5-й элемент
```

```
    int a = f.ReadByte(); // чтение 5-го элемента
```

```
    Console.WriteLine(a);
```

```
    a = f.ReadByte(); // чтение 6-го элемента
```

```
    Console.WriteLine(a);
```

```
    Console.WriteLine("Текущая позиция в потоке " + f.Position);
```

```
    f.Close();
```

```
} }
```

## Пример работы с потоком байтов

cmd C:\Windows\system32\cmd.exe

100 0 1 2 3 4 5 6 7 8 9 10 9 8 7 6 0 0 0 0

4

5

Текущая позиция в потоке 7

Для продолжения нажмите любую клавишу . . .

# Потоки байтов. Класс FileStream

Элемент	Описание
CanRead, CanSeek, CanWrite	Свойства, определяющие, какие операции поддерживает поток: чтение, прямой доступ и/или запись
Close	Закрывает текущий поток и освобождает связанные с ним ресурсы (сокеты, указатели на файлы и т. п.)
EndRead, EndWrite	Ожидать завершения асинхронного ввода; закончить асинхронный вывод
Flush	Записать данные из буфера в связанный с потоком источник данных и очистить буфер. Если для данного потока буфер не используется, то этот метод ничего не делает
Length	Возвратить длину потока в байтах
Position	Возвратить текущую позицию в потоке
Read, ReadByte	Считать последовательность байтов (или один байт) из текущего потока и переместить указатель в потоке на количество считанных байтов
Seek	Установить текущий указатель потока на заданную позицию
SetLength	Установить длину текущего потока
Write, WriteByte	Записать последовательность байтов (или один байт) в текущий поток и переместить указатель в потоке на количество записанных байтов

# Работа с файлами. Классы File и FileInfo

Методы	Класс FileInfo . Описание
<b>CopyTo(path)</b>	копирует файл в новое место по указанному пути path
<b>Create()</b>	создает файл
<b>Delete():</b>	удаляет файл
<b>MoveTo(destFileName)</b>	перемещает файл в новое место
<b>Свойство Directory</b>	получает родительский каталог в виде объекта DirectoryInfo
<b>Свойство DirectoryName</b>	получает полный путь к родительскому каталогу
<b>Свойство Exists</b>	указывает, существует ли файл
<b>Свойство Length</b>	получает размер файла
<b>Свойство Extension</b>	получает расширение файла
<b>Свойство Name</b>	получает имя файла
<b>Свойство FullName</b>	получает полное имя файла

Методы	Класс File . Описание
<b>Copy()</b>	копирует файл в новое место
<b>Create()</b>	создает файл
<b>Delete()</b>	удаляет файл
<b>Move</b>	перемещает файл в новое место
<b>Exists(file)</b>	определяет, существует ли файл

# 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);
}
```

# 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);
}
```

Метод CopyTo класса FileInfo принимает два параметра: путь, по которому файл будет копироваться, и булево 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.

# Чтение и запись файлов

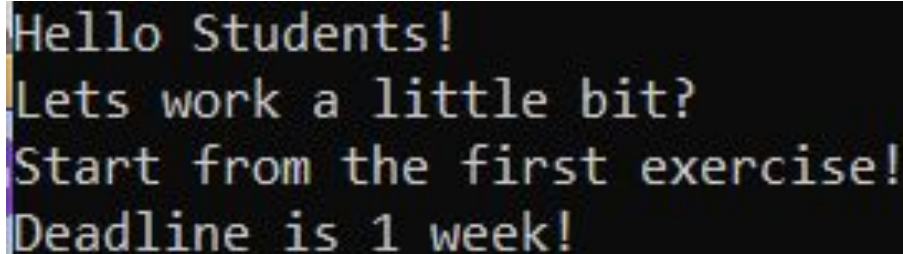
<code>AppendAllLines(String, IEnumerable&lt;String&gt;)</code>	добавляют в файл набор строк. Если файл не существует, то он создается
<code>AppendAllText(String, String)</code>	добавляют в файл строку. Если файл не существует, то он создается
<code>byte[] ReadAllBytes (string path)</code>	считывают содержимое бинарного файла в массив байтов
<code>string[] ReadAllLines (string path)</code>	считывают содержимое текстового файла в массив строк
<code>string ReadAllText (string path)</code>	считывают содержимое текстового файла в строку
<code>IEnumerable&lt;string&gt; ReadLines (string path)</code>	считывают содержимое текстового файла в коллекцию строк
<code>void WriteAllBytes (string path, byte[] bytes)</code>	записывают массив байт в бинарный файл. Если файл не существует, он создается. Если существует, то перезаписывается
<code>void WriteAllLines (string path, string[] contents)</code>	записывают массив строк в текстовый файл. Если файл не существует, он создается. Если существует, то перезаписывается
<code>WriteAllText (string path, string contents)</code>	записывают строку в текстовый файл. Если файл не существует, он создается. Если существует, то перезаписывается

# Пример чтения и запись в файл

```
string path = @"c:\app\content.txt";

string originalText = "Hello Students!";
// запись строки
File.WriteAllText(path, originalText);
// дозапись в конец файла
File.AppendAllText(path, "\nLets work a little bit?\n");

File.AppendAllLines(path, new[] { "Start from the first exercise!",
"Deadline is 1 week!" });
// чтение файла
string fileText = File.ReadAllText(path);
Console.WriteLine(fileText);
```



```
Hello Students!
Lets work a little bit?
Start from the first exercise!
Deadline is 1 week!
```



# StreamWriter

Методы	Чтение из файла. Класс StreamReader Описание
<b>Close</b>	закрывает считываемый файл и освобождает все ресурсы
<b>Peek</b>	возвращает следующий доступный символ, если символов больше нет, то возвращает -1
<b>Read</b>	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
<b>ReadLine</b>	считывает одну строку в файле
<b>ReadToEnd</b>	считывает весь текст из файла

Методы	Запись в файл. . Класс StreamWriter Описание
<b>Close</b>	закрывает записываемый файл и освобождает все ресурсы
<b>Flush</b>	записывает в файл оставшиеся в буфере данные и очищает буфер
<b>Write</b>	записывает в файл данные простейших типов, как int, double, char, string и т.д.
<b>WriteLine</b>	также записывает данные, только после записи добавляет в файл символ окончания строки

# Пример чтения из файла

```
string path= @"C:\SomeDir\hta.txt";
try
{
    Console.WriteLine("***считываем весь файл***");
    using (StreamReader sr = new StreamReader(path))
    {
        Console.WriteLine(sr.ReadToEnd());
    }
}
catch (Exception e)
{
    Console.WriteLine(e.Message);
}
```

```
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("*****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);
}
```

# Пример записи в текстовый файл

```
string readPath= @"C:\SomeDir\hta.txt";
string writePath = @"C:\SomeDir\ath.txt";

string text = "";
try
{
    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);
    }
}
catch (Exception e)
{
    Console.WriteLine(e.Message);
}
```

**true** - новые данные добавляются в  
конце к уже имеющимся данным.  
**false**, то файл перезаписывается

Кодировка, в которой записывается файл.

# Работа с дисками

## Методы:

GetDrives()	Возвращает имена всех логических дисков на компьютере.
GetType()	Возвращает объект Type для текущего экземпляра.
ToString()	Возвращает имя диска в виде строки.

## 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.

# Пример работы с классом DriveInfo

```
using System.IO;  
using static System.Console;
```

```
DriveInfo[] drives = DriveInfo.GetDrives();
```

```
foreach (DriveInfo drive in drives)
```

```
{
```

```
    WriteLine($"Название: {drive.Name}");
```

```
    WriteLine($"Корневой каталог диска: {drive.RootDirectory}");
```

```
    WriteLine($"Тип: {drive.DriveType}");
```

```
    WriteLine($"Имя файловой системы: {drive.DriveFormat}");
```

```
    if (drive.IsReady)
```

```
    { WriteLine($"Объем диска: {drive.TotalSize}");
```

```
      WriteLine($"Свободное пространство: {drive.TotalFreeSpace}");
```

```
      WriteLine($"Метка диска: {drive.VolumeLabel}");
```

```
}}
```

```
Название: C:\  
Корневой каталог диска: C:\  
Тип: Fixed  
Имя файловой системы: NTFS  
Объем диска: 510913409024  
Свободное пространство: 210627727360  
Метка диска: Acer
```

# Работа с каталогами. Класс Directory и DirectoryInfo

Методы	Класс Directory . Описание
<b>CreateDirectory(path)</b>	создает каталог по указанному пути path
<b>Delete(path)</b>	удаляет каталог по указанному пути path
<b>Exists(path)</b>	определяет, существует ли каталог по указанному пути path. Если существует, возвращается true, если не существует, то false
<b>GetDirectories(path)</b>	получает список каталогов в каталоге path
<b>GetFiles(path)</b>	получает список файлов в каталоге path
<b>Move(sourceDirName, destDirName):</b>	перемещает каталог
<b>GetParent(path)</b>	получение родительского каталога

Методы	Класс DirectoryInfo. Описание
<b>Create()</b>	создает каталог
<b>CreateSubdirectory(path)</b>	создает подкаталог по указанному пути path
<b>Delete()</b>	удаляет каталог
Свойство <b>Exists</b>	определяет, существует ли каталог
<b>GetDirectories()</b>	получает список каталогов
<b>GetFiles()</b>	получает список файлов
<b>MoveTo(destDirName)</b>	перемещает каталог
Свойство <b>Parent</b>	получение родительского каталога
Свойство <b>Root</b>	получение корневого каталога

# Создание каталога

## Класс

### DirectoryInfo

```
string path = @"C:\SomeDir";  
string subpath = @"program\avalon";  
DirectoryInfo dirInfo = new DirectoryInfo(path);  
if (!dirInfo.Exists)  
{  
    dirInfo.Create();  
}  
dirInfo.CreateSubdirectory(subpath);
```

## Класс

### Directory

```
string path = @"C:\SomeDir";  
string subpath = @"program\avalon";  
if (!Directory.Exists(path))  
{  
    Directory.CreateDirectory(path);  
}  
Directory.CreateDirectory($"{path}/{subpath}");
```

# Удаление каталога

```
string dirName = @"C:\SomeDir";

DirectoryInfo dirInfo = new DirectoryInfo(dirName);
if (dirInfo.Exists)
{
    dirInfo.Delete(true);
    Console.WriteLine("Каталог удален");
}
else
{
    Console.WriteLine("Каталог не существует");
}
```

```
string dirName = @"C:\SomeDir";
try
{
    Directory.Delete(dirName, true);
    Console.WriteLine("Каталог удален");
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
}
```



# 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);  
}
```

# Получение информации о каталоге

```
using System.IO;
using static System.Console;

string dirName = "C:\\\\Program Files";

DirectoryInfo dirInfo = new DirectoryInfo(dirName);

WriteLine("Название каталога: {0}", dirInfo.Name);
WriteLine("Полное название каталога: {0}", dirInfo.FullName);
WriteLine("Время создания каталога: {0}", dirInfo.CreationTime);
WriteLine("Время последнего доступа: {0}", dirInfo.LastAccessTime);
WriteLine("Время последнего изменения: {0}", dirInfo.LastWriteTime);
WriteLine("Корневой каталог: {0}", dirInfo.Root);
```

```
Название каталога: Program Files
Полное название каталога: C:\\Program Files
Время создания каталога: 07.12.2019 12:14:52
Время последнего доступа: 20.09.2022 12:57:47
Время последнего изменения: 07.09.2022 19:22:31
Корневой каталог: C:\\
```

# Получение списка файлов и подкаталогов

```
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);
    }
}
```

```
Подкаталоги:
C:\$Recycle.Bin
C:\$WinREAgent
C:\Config.Msi
C:\Documents and Settings
C:\MSOCache
C:\OEM
C:\OneDriveTemp
C:\PABCWork.NET
C:\PerfLogs
C:\Program Files
C:\Program Files (x86)
C:\ProgramData
C:\Python
C:\Recovery
C:\System Volume Information
C:\TempPatchCD
C:\Users
C:\Windows

Файлы:
C:\DumpStack.log.tmp
C:\E146354949C9
C:\hiberfil.sys
C:\PageFile.sys
C:\swapfile.sys
```

# Фильтрация папок и файлов

В качестве фильтра в методы **GetDirectories** и **GetFiles** передается шаблон, который может содержать два плейсхолдера:

\* - соответствует любому количеству символов

? соответствует одному символу

```
string dirName = @"C:\";  
// класс Directory  
string[] dirs1 = Directory.GetDirectories(dirName, "*on");  
  
foreach (string dir in dirs1)  
    Console.WriteLine(dir);
```

```
C:\Python  
C:\System Volume Information
```

# BinaryReader

Методы	Класс BinaryWriter Описание
<b>Close()</b>	закрывает поток и освобождает ресурсы
<b>Flush()</b>	очищает буфер, дописывая из него оставшиеся данные в файл
<b>Seek()</b>	устанавливает позицию в потоке
<b>Write()</b>	записывает данные в поток

Методы	Класс BinaryReader Описание
<b>Close()</b>	закрывает поток и освобождает ресурсы
<b>ReadBoolean()</b>	считывает значение bool и перемещает указатель на один байт
<b>ReadDecimal()</b>	считывает значение decimal и перемещает указатель на 16 байт
<b>ReadByte()</b>	считывает один байт и перемещает указатель на один байт
<b>ReadChar()</b>	считывает значение char, то есть один символ, и перемещает указатель на столько байтов, сколько занимает символ в текущей кодировке
<b>ReadDouble()</b>	считывает значение double и перемещает указатель на 8 байт
<b>ReadInt16()</b>	считывает значение short и перемещает указатель на 2 байта
<b>ReadInt32()</b>	считывает значение int и перемещает указатель на 4 байта
<b>ReadSingle()</b>	считывает значение float и перемещает указатель на 4 байта
<b>ReadString()</b>	считывает значение string. Каждая строка предваряется значением длины строки, которое представляет 7-битное целое число

# Пример формирования двоичного файла

```
class Program
{
    static void Main(string[] args)
    {
        try
        {
            BinaryWriter f = new BinaryWriter(new FileStream(@"D:\Информационные
технологии\states.txt", FileMode.Create));
            double d = 0;
            while (d<4)
            {
                f.Write(d);
                d +=0.33;
            }

            f.Seek(16, SeekOrigin.Begin); // второй элемент файла
            f.Write(888d);
            f.Close();
        }
        catch (FileNotFoundException e)
        {
            Console.WriteLine(e.Message);
            Console.WriteLine("Проверьте правильность имени файла");
            return;
        }
        catch (Exception e)
        {
            Console.WriteLine("Error: "+ e.Message);
            return;
        }
    }
}
```

# Пример чтения из двоичного файла

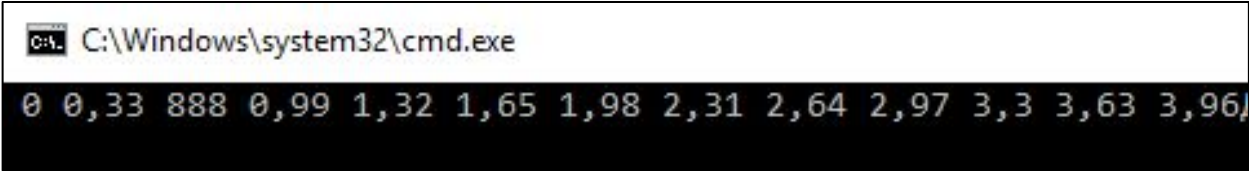
```
FileStream f=new FileStream(@"D:\Информационные технологии\states.txt", FileMode.Open);
BinaryReader fin = new BinaryReader(f);

long n = f.Length / 8; //количество чисел в файле
double[] x = new double[n];

long i = 0;
try
{
    while (true) x[i++] = fin.ReadDouble();
}

catch (EndOfStreamException e) { }

foreach (double d in x) Console.Write(" " + d);
fin.Close();
f.Close();
```



C:\Windows\system32\cmd.exe

0 0,33 888 0,99 1,32 1,65 1,98 2,31 2,64 2,97 3,3 3,63 3,96