

<ерат>

Работа с файловой системой в .NET



Работа с файловой системой

Множество задач в программировании связаны с работой с каталогами и файлами. Часто требуется хранить информацию в файловой системе, для чего требуется создавать различные файлы и директории, записывать в них информацию, или что-то читать из них.

.NET Предоставляет множество возможностей по управлению файловой системой

Работа с дисками

DriveInfo – класс для работы с дисками в .NET

```
DriveInfo[] drives = DriveInfo.GetDrives();
foreach (DriveInfo drive in drives)
{
    Console.WriteLine("Имя: {0}", drive.Name);
    Console.WriteLine("Тип: {0}", drive.DriveType);
    Console.WriteLine("Готовность: {0}", drive.IsReady);
}
```

```
Имя: C:\
Тип: Fixed
Готовность: True
Имя: D:\
Тип: Fixed
Готовность: True
Имя: E:\
Тип: CDRom
Готовность: False
```

Методы и свойства

- **GetDrives()** - Возвращает имена всех логических дисков на компьютере `DriveInfo[] allDrives = DriveInfo.GetDrives();`
- **AvailableFreeSpace** - Указывает объем доступного свободного места на диске в байтах.
- **DriveFormat** - Получает имя файловой системы, например NTFS или FAT32.
- **DriveType** - Получает тип диска, такой как компакт-диск, съемный, сетевой или жесткий.
- **IsReady** - Получает 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** - Получает общий размер места для хранения на диске в байтах.
- **VolumeLabel** - Получает или задает метку тома диска.

Работа с каталогами

Directory – Статические методы для создания, перемещения и перечисления в каталогах и вложенных каталогах. Этот класс не наследуется.

Пример создания каталога

```
// Пытаемся создать директорию
Directory.CreateDirectory(filename);
Console.WriteLine("The directory was created successfully at {0}.",
    Directory.GetCreationTime(filename));
```

Методы

- **CreateDirectory(String)** - Создает все каталоги и подкаталоги по указанному пути, если они еще не существуют.
- **Delete(String)** - Удаляет пустой каталог по заданному пути.
- **Exists(String)** - Определяет, указывает ли заданный путь на существующий каталог на диске.
- **GetCurrentDirectory()** - Получает текущий рабочий каталог приложения.
- **GetDirectories(String)** - Возвращает имена подкаталогов (включая пути) в указанном каталоге.
- **GetFiles(String)** - Возвращает имена файлов (с указанием пути к ним) в указанном каталоге.
- **Move(String, String)** - Перемещает файл или каталог со всем его содержимым в новое местоположение.

Работа с каталогами

DirectoryInfo – Методы экземпляра класса для создания, перемещения и перечисления в каталогах и подкаталогах.

Пример создания и удаления каталога

```
DirectoryInfo di = new DirectoryInfo(path:@"c:\MyDir");

// Пытаемся создать директорию.
di.Create();
Console.WriteLine("The directory was created successfully.");

// Удаляем ее.
di.Delete();
Console.WriteLine("The directory was deleted successfully.");
```

Методы и свойства

- **Create()** - Создает каталог.
- **Delete()** - Удаляет каталог если он пуст.
- **Exists**- Свойство, позволяет узнать, существует ли каталог.
- **GetDirectories()** - Возвращает подкаталоги текущего каталога.
- **GetFiles()** - Возвращает список файлов текущего каталога.
- **MoveTo(String)** - Перемещает экземпляр DirectoryInfo и его содержимое в местоположение, на которое указывает новый путь.
- **Parent** - Получает родительский каталог.

Работа с файлами

File – Предоставляет статические методы для создания, копирования, удаления, перемещения и открытия одного файла.

Пример создания и записи в файл

```
if (!File.Exists(filename))
{
    // Создаем файл для записи.
    using (StreamWriter sw = File.CreateText(filename))
    {
        sw.WriteLine("Hello there");
    }
}
```

Методы

- **Copy(String, String)** - Копирует существующий файл в новый файл. Перезапись файла с тем же именем не разрешена.
- **Create(String)** - Создает или перезаписывает файл в указанном пути.
- **Exists(String)** - Определяет, существует ли заданный файл.
- **Move(String, String)** - Перемещает заданный файл в новое местоположение и разрешает переименование файла.

Работа с файлами

FileInfo – Предоставляет свойства и методы экземпляра для создания, копирования, удаления, перемещения и открытия файлов

Пример создания и записи в файл

```
var fileInfo = new FileInfo(filename);
```

```
// Создаем файл для записи.  
using (StreamWriter sw = fileInfo.CreateText())  
{  
    sw.WriteLine("Hello there");  
}
```

Методы и свойства

- **Length** - Получает размер текущего файла в байтах.
- **Name** - Возвращает имя файла.
- **DirectoryName** - Получает строку, представляющую полный путь к каталогу.
- **Exists** - Получает значение, показывающее, существует ли файл.
- **Extension** - Получает строку, содержащую расширение файла.
- **Create()** - Создает файл.
- **Delete()** - Удаляет файл без возможности восстановления
- **MoveTo(String)** - Перемещает заданный файл в новое местоположение и разрешает переименование файла.
- **CopyTo(String)** - Копирует существующий файл в новый файл и запрещает перезапись существующего файла.

Чтение и запись в файл

FileStream – Предоставляет возможность чтения и записи как в текстовые так и в бинарные файлы.

```
string filename = @"c:\Users\MyFile.txt";  
FileStream fs = File.Open(filename, FileMode.OpenOrCreate);
```

FileMode – Задаёт режим открытия файла.

Методы и свойства

- **Append** – Открывает существующий, или создает. Если файл существует, то помещает новый текст в конец файла.
- **Create** – Создает новый файл или перезаписывает старые.
- **CreateNew** – создает новый файл, если такой файл уже существует, то выбрасывает `IOException`
- **Open** – открывает существующий файл. Если файл не существует – вызывается исключение.
- **OpenOrCreate** – открывает файл или создает новый
- **Truncate** – открывает существующий файл, и очищает его. Новый не создается

Чтение и запись в файл

FileStream – Предоставляет возможность чтения и записи как в текстовые так и в бинарные файлы.

Пример записи и чтения из файла

```
string text = "Some text to write";

//Создаем файл с использованием FileStream.
using (FileStream fileStream = File.Create(filename))
{
    fileStream.Write(array: new UTF8Encoding().GetBytes(text), offset: 0, count: text.Length);
}

//Открываем FileStream для чтения из файла.
using (FileStream fileStream = File.OpenRead(filename))
{
    var buffer = new byte[1024];
    var encoding = new UTF8Encoding();
    while (fileStream.Read(buffer, offset: 0, count: buffer.Length) > 0)
    {
        Console.WriteLine(encoding.GetString(buffer));
    }
}
```

Методы и свойства

- **Read(Byte[] array, Int32 offset, Int32 count)** - Выполняет чтение блока байтов из файла в массив байт.
 - **array Byte[]** - массив в который будут помещены считанные байты.
 - **offset Int32** - Смещение в байтах в массиве array, в который будут помещены считанные байты.
 - **count Int32** - Максимальное число байтов, которые будут считаны.
- **Write (byte[] array, int offset, int count)** - записывает в файл данные из массива байтов.
 - **array Byte[]** - Массив, предназначенный для записи.
 - **offset Int32** - Смещение байтов (начиная с нуля) массива array, с которого начинается копирование байтов в поток.
 - **count Int32** - Максимальное число байтов для записи.

Чтение и запись в файл

StreamReader – удобно использовать для чтения из текстовых файлов. Считывает символы из потока байтов в определенной кодировке.

```
StreamReader streamReader = new StreamReader(path);

string path = "D://MyWonderfulProject//test.txt";
// Построчно читаем и выводим в консоль содержимое файла
using (StreamReader sr = new StreamReader(path))
{
    string line;
    while ((line = sr.ReadLine()) != null)
    {
        Console.WriteLine(line);
    }
}
```

Методы и свойства

- **Close()** - Закрывает объект StreamReader и основной поток и освобождает все системные ресурсы, связанные с устройством чтения.
- **Peek()** - Возвращает следующий доступный символ, но не использует его.
- **Read()** - Выполняет чтение следующего символа из входного потока и перемещает положение символа на одну позицию вперед.
- **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.

Чтение и запись в файл

StreamWriter – используется для записи в текстовые файлы.

```
StreamWriter streamWriter = new StreamWriter(path);
```

Получение имен всех каталогов и их запись в файл

```
string path = "D://MyWonderfulProject//test.txt";  
//Получаем директории, расположенны на диске C://  
DirectoryInfo[] cDirs = new DirectoryInfo(path@"C:\").GetDirectories();  
  
//Запишем полученные имена в файл  
using (StreamWriter sw = new StreamWriter(path))  
{  
    foreach (DirectoryInfo dir in cDirs)  
    {  
        sw.WriteLine(dir.Name);  
    }  
}
```

Методы и свойства

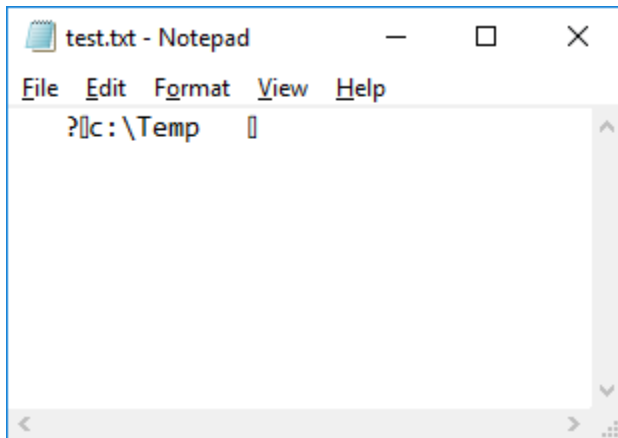
- **Close()** - Закрывает текущий объект StreamWriter и базовый поток.
- **Write(String)** - Записывает в поток строку.
- **WriteLine(Char[])** - Записывает в текстовый поток массив символов, за которыми следует признак конца строки.

Чтение и запись в файл

BinaryWriter – Записывает примитивные типы в двоичный поток и поддерживает запись строк в заданной кодировке.

```
string path = "D://MyWonderfulProject//test.txt";
using (BinaryWriter writer = new BinaryWriter(output: File.Open(path,
    FileMode.Create)))
{
    writer.Write(1.250F);
    writer.Write(@"c:\Temp");
    writer.Write(10);
    writer.Write(true);
}
```

Результат:



Методы и свойства

- **Close()** - Закрывает текущий BinaryWriter и базовый поток.
- **Write(String)** - Записывает в текущий поток строку, предваряемую ее длиной, используя текущую кодировку BinaryWriter, и перемещает позицию в потоке вперед в соответствии с используемой кодировкой и количеством записанных в поток символов.

Чтение и запись в файл

BinaryReader – Для работы с данными в бинарном формате предназначена пара классов BinaryReader и BinaryWriter.

Методы и свойства

- **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.
- **ReadInt32()** - Считывает целое число со знаком длиной 4 байта из текущего потока и перемещает текущую позицию в потоке на четыре байта вперед.
- **ReadDouble()** - Считывает число с плавающей запятой длиной 8 байт из текущего потока и перемещает текущую позицию в потоке на восемь байт вперед.
- **ReadString()** - Считывает строку из текущего потока. Строка предваряется значением длины строки, которое закодировано как целое число блоками по семь битов.

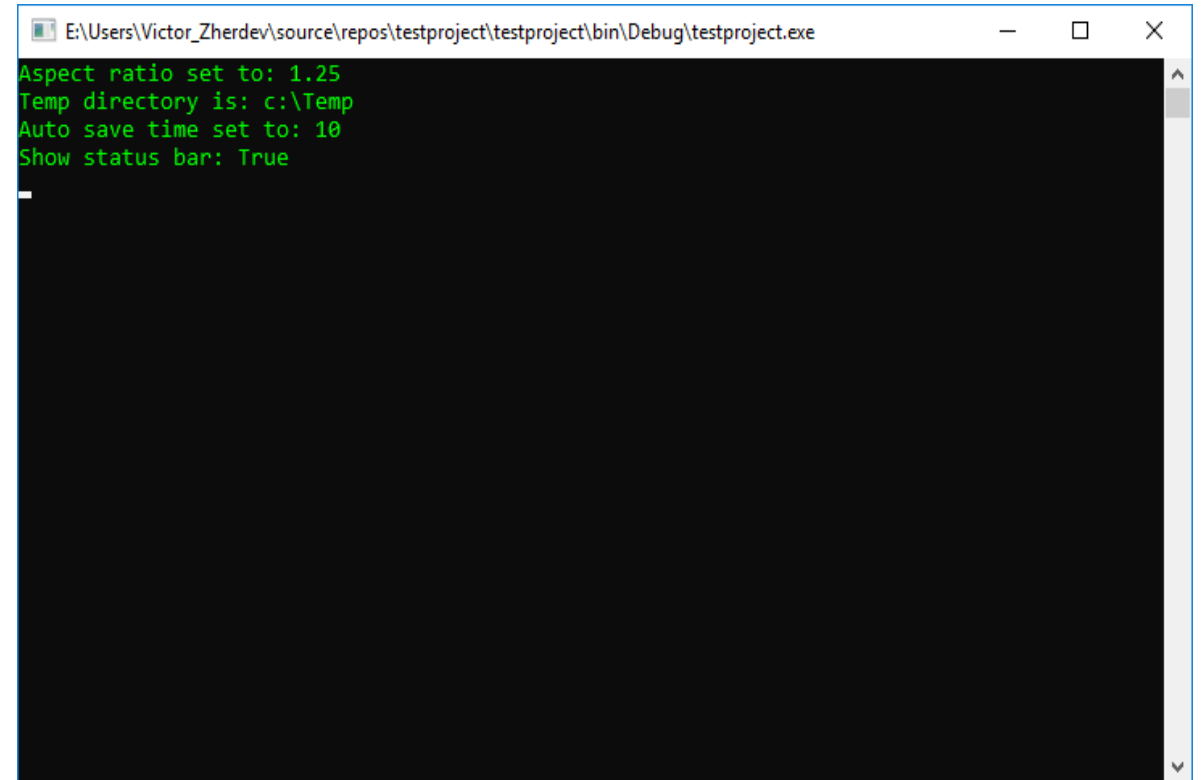
Чтение и запись в файл

```
string path = "D://MyWonderfulProject//test.txt";
if (File.Exists(path))
{
    float aspectRatio;
    string tempDirectory;
    int autoSaveTime;
    bool showStatusBar;

    using (BinaryReader reader = new BinaryReader(input: File.Open(path,
        FileMode.Open)))
    {
        aspectRatio = reader.ReadSingle();
        tempDirectory = reader.ReadString();
        autoSaveTime = reader.ReadInt32();
        showStatusBar = reader.ReadBoolean();
    }

    Console.WriteLine("Aspect ratio set to: " + aspectRatio);
    Console.WriteLine("Temp directory is: " + tempDirectory);
    Console.WriteLine("Auto save time set to: " + autoSaveTime);
    Console.WriteLine("Show status bar: " + showStatusBar);
}
```

Результат:

A screenshot of a Windows console window titled "E:\Users\Victor_Zherdev\source\repos\testproject\testproject\bin\Debug\testproject.exe". The console output, displayed in green text on a black background, shows the following lines: "Aspect ratio set to: 1.25", "Temp directory is: c:\Temp", "Auto save time set to: 10", and "Show status bar: True". There is a small white cursor line below the last line of output.

```
E:\Users\Victor_Zherdev\source\repos\testproject\testproject\bin\Debug\testproject.exe
Aspect ratio set to: 1.25
Temp directory is: c:\Temp
Auto save time set to: 10
Show status bar: True
```

Немного про память

IDisposable - Предоставляет механизм для освобождения неуправляемых ресурсов.

- При работе с файлами задействуются неуправляемые ресурсы (например - дескриптор открытого файла в операционной системе)
- .NET Framework понятия не имеет о том, что происходит там, где его нет.
- А если он ничего об этом не знает, он не освободит память.
- Два пути решения – Dispose и Using

```
BinaryWriter writer = new BinaryWriter(output: File.Open(path,  
    FileMode.Create));  
  
writer.Dispose();
```

Немного про память

Оператор **using** - Предоставляет удобный синтаксис, обеспечивающий правильное использование объектов IDisposable.

- Экземпляр оборачивается в using(...) {...}
- По окончании блока using объект

должен быть уничтожен и будет вызван Dispose().

```
// До
using (BinaryWriter binaryWriter = new BinaryWriter(output: File.Open(path, FileMode.Create)))
{
    binaryWriter.Write("text");
}

//После
BinaryWriter binaryWriter2 = new BinaryWriter(output: File.Open(path, FileMode.Create));
try
{
    binaryWriter2.Write("text");
}
finally
{
    if (binaryWriter2 != null)
    {
        ((IDisposable)binaryWriter2).Dispose();
    }
}
```