

Технология программирования

Файлы

TL; DR

```
using (var reader =  
    new StreamReader("file.txt"))  
{  
    string line;  
    while ((line = reader.ReadLine()) != null)  
    {  
        string[] parts = line.Split(',');  
    }  
}
```

Потоки

- **Поток**, это абстрактное понятие, которое обозначает динамическую изменяющуюся во времени последовательность чего-либо.
- В программах на С#, **поток** – это последовательность байтов, связанная с конкретными устройствами компьютера (дисками, дисплеями, принтерами, клавиатурами) посредством системы ввода/вывода.

ПОТОКИ

Потоки делятся на 3 категории:

1. Байтовые потоки – **FileStream**, **MemoryStream** и т.д.
2. Символьные – **StringReader**, **StringWriter**, **StreamReader**, **StreamWriter**.
3. Двоичные – **BinaryReader** и **BinaryWriter**.

Потоки

Большинство устройств, предназначенных для выполнения операций ввода/вывода, являются байт-ориентированными. Этим и объясняется тот факт, что на самом низком уровне все операции ввода/вывода манипулируют с байтами в рамках байтовых потоков.

Потоки

Символьно-ориентированные потоки, предназначенные для манипулирования символами, а не байтами, являются потоками ввода/вывода более высокого уровня.

Потоки

Двоичные потоки используются для чтения и записи типов данных в виде двоичных значений. Они содержат статические методы для каждого стандартного типа (**ReadBoolean**, **ReadByte**, **ReadBytes**, **ReadChar**, **ReadChars**, **ReadDecimal**, **ReadDouble** и т.д.).

Потоки

Классы для работы с потоками
прописаны в пространстве имен
System.IO

Потоки

Все классы потоков C# наследуют от базового абстрактного класса **Stream**. Он предоставляет следующие методы и свойства:

- **bool CanRead** - можно ли читать из потока
- **bool CanWrite** - можно ли писать в поток

ПОТОКИ

- **bool CanSeek** - можно ли задать в потоке текущую позицию
- **long Position** - позиция текущего элемента потока.
- **long Length** - общее количество символов потока.
- **long Seek (long index, SeekOrigin origin)** - позиционирование в потоке.

ПОТОКИ

- Методы для работы с байтами данных:
 - int **ReadByte()**
 - int **Read**(byte[] buff, int index, int count)
 - void **WriteByte**(byte b)
 - int **Write**(byte[] buff, int index, int count)
- **void Close()** – закрытие потока.

Потоки

Наследники класса **Stream**:

- **BufferedStream** – обеспечивает буферизацию байтового потока. Как правило, буферизованные потоки являются более производительными по сравнению с небуферизованными.

Потоки

- **FileStream** – байтовый поток, обеспечивающий файловые операции ввода/вывода.
- **MemoryStream** – байтовый поток, использующий в качестве источника и хранилища информации оперативную память.

Потоки

Для записи данных в текстовом формате используются специальные классы – **StringReader** и **StringWriter**. Они являются наследниками классов **TextReader / TextWriter** и реализуют следующие операции:

- **int Peek()** - считывает следующий знак, не изменяя состояние средства чтения или источника знака.

ПОТОКИ

- **int Read(...)** - Читает значения из ВХОДНОГО ПОТОКА.
- **string ReadLine()** – читает строку символов из текущего потока.
- **string ReadToEnd()** - читает все символы, начиная с текущей позиции символьного потока.
- **void Write(...)** – записывает символьные представления значений базовых типов.

Потоки

- **Standard input stream** – по умолчанию ориентирован на получение информации с клавиатуры.
- **Standard output stream** – по умолчанию обеспечивает вывод информации на дисплей.
- **Standard error stream** – по умолчанию выводит информацию на дисплей.

Потоки

Для работы со стандартными потоками используется класс **Console**. Он содержит свойства, предоставляющие доступ к объектам потоков типа **TextReader** и **TextWriter**.

- **Console.In**
- **Console.Out**
- **Console.Error**

Потоки

С их помощью можно выводить значения непосредственно в нужный поток:

- **Console.In.ReadLine(...);**
по умолчанию для ввода
- **Console.Out.WriteLine(...);**
по умолчанию для вывода
- **Console.Error.WriteLine(...);**

Пример

```
class Program
{
    static string[] str = {
        "1234567890",
        "abcdefghij",
        "#####",
        "+++++++",
    };
};
```

Пример

```
static void Main(string[] args)
{
    char[] buff = new char[32];
    for (int i = 0; i < buff.Length; i++)
        buff[i] = (char)25;
```



Пример

```
FileStream stream = new FileStream("test.txt",  
    FileMode.Create, FileAccess.Write);  
  
BufferedStream buffered =  
    new BufferedStream(stream);  
  
StreamWriter writer =  
    new StreamWriter(buffered);  
  
for (int i = 0; i < str.Length; i++)  
    writer.WriteLine(str[i]);  
  
writer.Close();
```

Пример

1	1234567890
2	qwertyuiop
3	asdfghjkl
4	zxcvbnm
5	

FileMode

- **Append** - открывает файл, если он существует, и находит конец файла; либо **создает новый файл**.
- **Create** - указывает, что операционная система должна **создавать новый файл**. Если файл уже существует, он будет переписан.
- **CreateNew** - указывает, что операционная система должна создавать **новый файл**.

FileMode

- **Open** - указывает, что операционная система должна открыть **существующий** файл.
- **OpenOrCreate** - указывает, что операционная система должна **открыть файл**, если он существует, в противном случае должен быть **создан новый файл**.
- **Truncate** - указывает, что операционная система должна **открыть** существующий файл. Если файл открыт, он должен быть **усечен** таким образом, чтобы его размер стал равен нулю байтов.

FileAccess

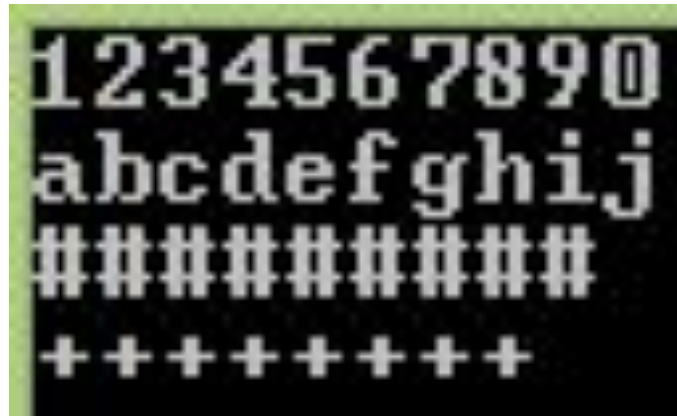
- **Read** - доступ для **чтения** файла. Данные можно прочитать из файла.
- **ReadWrite** - доступ для **чтения и записи** файла. Данные можно записать в файл и прочитать из файла.
- **Write** - доступ для **записи** в файл. Данные можно записать в файл.

Пример

```
stream = new FileStream("test.txt",  
                        FileMode.Open,  
                        FileAccess.Read);  
  
StreamReader reader =  
    new StreamReader(stream);
```

Пример

```
string line = "";  
while (line != null) {  
    line = reader.ReadLine();  
    WriteLine(line);  
}  
WriteLine();
```



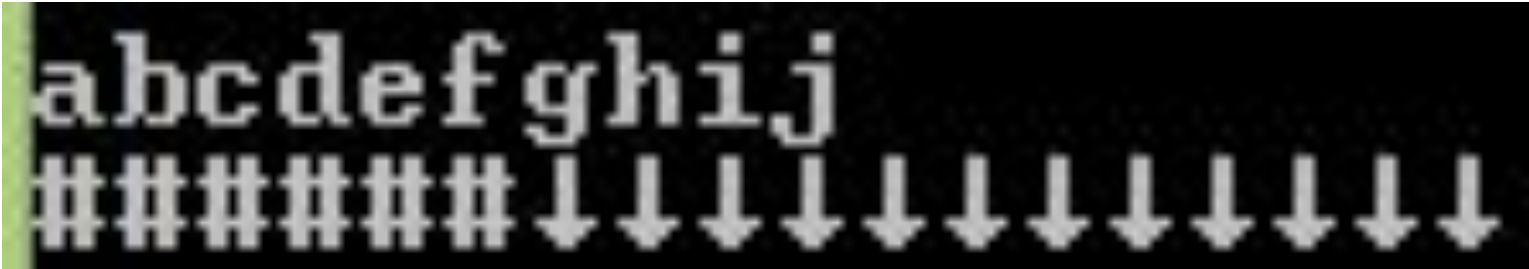
Пример

```
stream.Seek(0, SeekOrigin.Begin);  
reader.Read(buff, 0, 10);  
WriteLine(new string(buff));  
WriteLine();
```

1234567890↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓

Пример

```
reader.Read(buff, 0, 20);  
WriteLine(new string(buff));
```



```
abcdefghij
```

```
#####↓↓↓↓↓↓↓↓↓↓
```

Пример записи в файл фигур

```
abstract class Shape
{
    public abstract
        void Save(StreamWriter writer);

    public abstract double Area();
}
```

Пример

```
class Rectangle : Shape
{
    int w, h;

    public Rectangle(string[] line) {
        w = int.Parse(line[1]);
        h = int.Parse(line[2]);
    }
}
```

Пример

```
public override
    void Save(StreamWriter writer)
{
    writer.WriteLine("Прямоугольник, ширина:
                        + w + ", высота: " + h);
}
public override double Area()
{
    return w*h;
}
}
```


Пример

```
class Triangle : Shape
{
    int a, b, c;

    public Triangle(string[] line)
    {
        a = int.Parse(line[1]);
        b = int.Parse(line[2]);
        c = int.Parse(line[3]);
    }
}
```

Пример

```
public override void Save(StreamWriter writer)
{
    writer.WriteLine("Треугольник, a: " + a
                     + ", b: " + b + ", c: " + c);
}

public override double Area()
{
    double p = (a + b + c)/2.0;
    return Math.Sqrt(p*(p-a)*(p-b)*(p-c));
}
```

Файл с данными

1	1	10	10	
2	1	20	5	
3	2	3	4	5
4	2	5	4	3

Пример

```
static int StringsNum(StreamReader reader)
{
    int num = 1;
    int ch;
    do {
        ch = reader.Read();
        if (ch == '\n')
            num++;
    } while (ch != -1);
    reader.BaseStream.Position = 0;
    return num;
}
```

Пример

```
static void Main(string[] args)
{
    Shape[] arr;

    StreamReader reader =
        new StreamReader("in.txt");

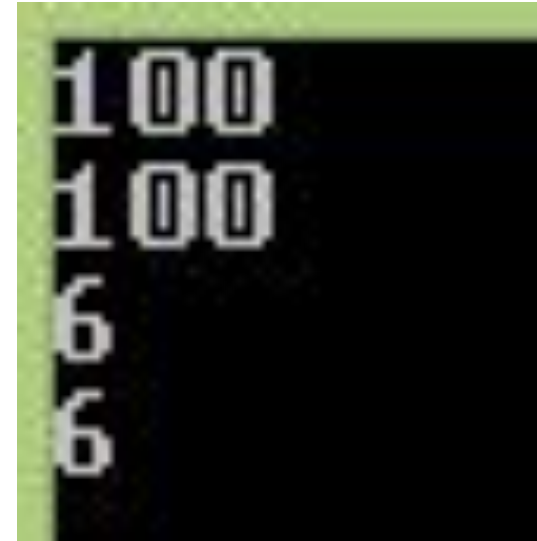
    arr = new Shape[StringsNum(reader)];
```

Пример

```
int i = 0;
while (!reader.EndOfStream) {
    string[] line =
        reader.ReadLine().Split(' ');
    int id = int.Parse(line[0]);
    if (id == 1)
        arr[i] = new Rectangle(line);
    else if (id == 2)
        arr[i] = new Triangle(line);
    i++;
}
reader.Close();
```

Пример

```
using (StreamWriter writer =  
        new StreamWriter("out.txt"))  
{  
    foreach (Shape s in arr)  
        s.Save(writer);  
}  
  
foreach (Shape s in arr)  
    Console.WriteLine(s.Area());  
  
Console.ReadKey();  
}
```



Пример

```
1 Прямоугольник, ширина: 10, высота: 10
2 Прямоугольник, ширина: 20, высота: 5
3 Треугольник, a: 3, b: 4, c: 5
4 Треугольник, a: 5, b: 4, c: 3
5
```


Двоичные файлы

Двоичный файл — последовательность произвольных байтов.

В некотором смысле двоичные файлы противопоставляются текстовым файлам.

При этом с точки зрения технической реализации на уровне аппаратуры, текстовые файлы являются частным случаем двоичных файлов, и, таким образом, в широком значении слова под определение «двоичный файл» подходит любой файл.

Двоичные файлы

Двоичные файлы хранят информацию в том виде, в каком она представлена в памяти компьютера, и потому неудобны для человека. Заглянув в такой файл, невозможно понять, что в нем записано; его нельзя создавать или исправлять вручную - в каком-нибудь текстовом редакторе - и т.п. Однако все эти неудобства компенсируются скоростью работы с данными.

Двоичные файлы

Для работы с двоичными файлами используются классы **BinaryReader** и **BinaryWriter**.

Класс **BinaryReader** содержит методы для чтения данных из двоичных файлов, а **BinaryWriter** соответственно, для записи.

Запись данных

- **Write(bool value)** – записывает в файл однобайтовое значение типа **bool**. При этом 0 будет соответствовать false, а 1 будет соответствовать true
- **Write(byte value)** – записывает в файл значение типа **byte**
- **Write(double value)** – записывает в файл 8-ми байтовое значение типа **double**
- **Write(int value)** – записывает в файл 4-х байтовое значение типа **int**

Запись данных

- **Write(char ch)** – записывает в файл СИМВОЛ В СООТВЕТСТВИИ С ТЕКУЩЕЙ КОДИРОВКОЙ
- **Write(char[] chars)** – записывает в файл МАССИВ СИМВОЛОВ В СООТВЕТСТВИИ С ТЕКУЩЕЙ КОДИРОВКОЙ
- **Write(string value)** – записывает в файл строку. Сначала в файл записывается размер строки, и после этого массив из СИМВОЛОВ

Чтение данных

- **bool ReadBoolean()** – читает из файла одно значение типа **bool**
- **byte ReadByte()** – читает из файла одно значение типа **byte**
- **byte[] ReadBytes(int count)** – читает из файла несколько значений типа **byte**
- **char ReadChar()** – читает из файла один символ **char** в соответствии с текущей установленной кодировкой

Чтение данных

- **double ReadDouble()** – читает из файла одно 8-ми байтовое значение типа **double**
- **int ReadInt32()** – читает из файла одно 4-х байтовое значение типа **int**
- **string ReadString()** – читает из файла строку символов. Считываемая строка в файле представлена своей длиной и набором символов **char**

Пример

```
const string fileName = "Settings.dat";

public static void WriteFile() {
    using (var writer = new BinaryWriter(
        File.Open(fileName, FileMode.Create)))
    {
        writer.Write(1.250F);
        writer.Write(@"c:\Temp");
        writer.Write(10);
        writer.Write(true);
    }
}
```


Пример

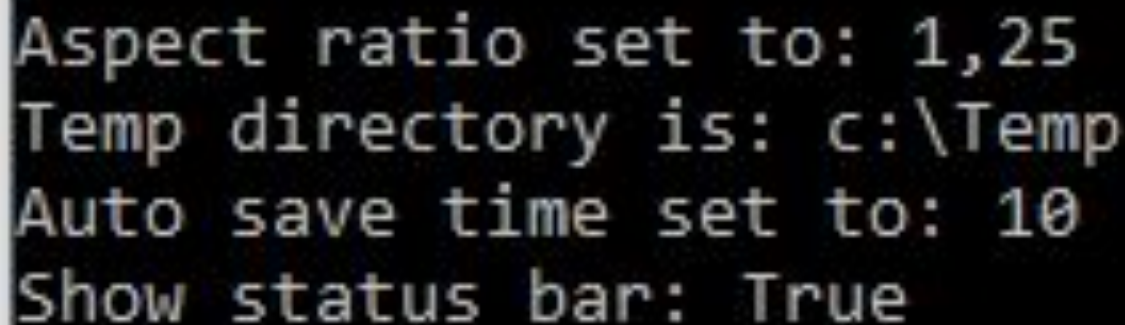
```
public static void ReadFile() {  
    float aspectRatio;  
    string tempDirectory;  
    int autoSaveTime;  
    bool showStatusBar;  
    if (File.Exists(fileName)) {  
        using (var reader = new BinaryReader(  
            File.Open(fileName, FileMode.Open))) {  
            aspectRatio = reader.ReadSingle();  
            tempDirectory = reader.ReadString();  
            autoSaveTime = reader.ReadInt32();  
            showStatusBar = reader.ReadBoolean();  
        }  
    }  
}
```

Пример

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

Пример

```
static void Main() {  
    WriteFile();  
    ReadFile();  
    ReadKey();  
}
```



A screenshot of a console window with a black background and white text. The text is displayed in four lines, showing the output of a program. The first line is 'Aspect ratio set to: 1,25', the second is 'Temp directory is: c:\Temp', the third is 'Auto save time set to: 10', and the fourth is 'Show status bar: True'.

```
Aspect ratio set to: 1,25  
Temp directory is: c:\Temp  
Auto save time set to: 10  
Show status bar: True
```

Пример №2

```
int i = 25;
double d = 3.14157;
bool b = true;
string s = "I am happy";

var writer = new BinaryWriter(
    new FileStream("mydata", FileMode.Create)
);
writer.Write(i);
writer.Write(d);
writer.Write(b);
writer.Write(s);
writer.Close();
```

Пример №2

```
var reader = new BinaryReader(  
    new FileStream("mydata", FileMode.Open)  
);  
WriteLine("Integer data: {0}",  
    reader.ReadInt32());  
WriteLine("Double data: {0}",  
    reader.ReadDouble());  
WriteLine("Boolean data: {0}",  
    reader.ReadBoolean());  
WriteLine("String data: {0}",  
    reader.ReadString());  
reader.Close();
```

Пример №2

```
Integer data: 25  
Double data: 3,14157  
Boolean data: True  
String data: I am happy
```

Дополнительные классы

Пространство имен `System.IO` содержит множество различных классов для работы с файловой системой, выполнения различных часто встречающихся операций и работы с различными популярными форматами файлов.

Класс File

```
string file = File.ReadAllText("C:\\file.txt");  
WriteLine(file);
```


Класс File

Статический класс файл содержит набор методов для выполнения часто встречающихся действий с файлами:

```
string file =  
    File.ReadAllText("C:\\file.txt");  
WriteLine(file);
```

Класс File

Метод **ReadAllLines** читает содержимое **ТЕКСТОВОГО** файла

```
string[] lines =  
    File.ReadAllLines("file.txt");  
  
foreach (string line in lines) {  
    if (line.Length > 80) {  
        // ...  
    }  
}
```

Класс File

Можно использовать его для подсчета количества строк в файле:

```
int lineCount =  
    File.ReadAllLines("file.txt").Length;
```

Класс File

Использование LINQ для поиска текста в файле:

```
bool exists = (  
    from line in File.ReadAllLines("file.txt")  
    where line == "Search pattern"  
    select line).Count() > 0;
```

Класс File

Метод **WriteAllLines** можно использовать для записи текста в файл:

```
string[] stringArray = new string[] {  
    "cat",  
    "dog",  
    "arrow"  
};  
File.WriteAllLines("file.txt", stringArray);
```

Класс File

Метод **AppendAllLines** позволяет дописывать текст в конец файла:

```
File.AppendAllText("C:\\perl.txt",  
                  "first part\n");  
File.AppendAllText("C:\\perl.txt",  
                  "second part\n");  
  
string third = "third part\n";  
File.AppendAllText("C:\\perl.txt", third);
```

Класс File

В классе File есть и методы для работы с двоичными файлами, например метод **ReadAllBytes**:

```
byte[] _logoBytes =  
    File.ReadAllBytes("Logo.png");
```

Класс File

Ряд методов вызывает функции операционной системы для действий с файлами. Например можно создать копию файла при помощи метода **Copy**:

```
File.Copy("file-a.txt", "file-new.txt");
```

```
WriteLine(File.ReadAllText("file-a.txt"));
```

```
WriteLine(File.ReadAllText("file-new.txt"));
```


Класс Directory

Класс Directory предназначен для работы с папками. Его методы позволяют выполнять различные действия, например получить список файлов из определенного каталога:

```
string[] array1 = Directory.GetFiles(@"C:\");  
WriteLine("--- Files: ---");  
foreach (string name in array1) {  
    WriteLine(name);  
}
```

Класс Directory

```
string[] array1 = Directory.GetFiles(@"C:\");  
string[] array2 =  
    Directory.GetFiles(@"C:\", "*.BIN");  
  
WriteLine("--- Files: ---");  
foreach (string name in array1) {  
    WriteLine(name);  
}  
WriteLine("--- BIN Files: ---");  
foreach (string name in array2) {  
    WriteLine(name);  
}
```

Класс Directory

Создание новых папок:

```
Directory.CreateDirectory("C:\\newfolder");  
Directory.CreateDirectory(@"C:\newfolder2");  
Directory.CreateDirectory(@"C:\newfolder2");
```

Класс Path

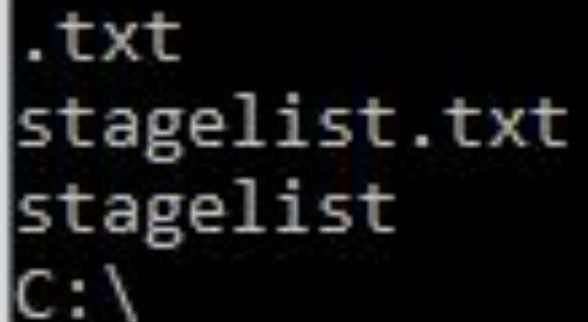
Работа с путями в файловой системе:

```
string path = "C:\\stagelist.txt";  
string extension = Path.GetExtension(path);  
string filename = Path.GetFileName(path);  
string filenameNoExtension =  
    Path.GetFileNameWithoutExtension(path);  
string root = Path.GetPathRoot(path);
```

Класс Path

Работа с путями в файловой системе:

```
WriteLine("{0}\n{1}\n{2}\n{3}",  
    extension,  
    filename,  
    filenameNoExtension,  
    root);
```



```
.txt  
stagelist.txt  
stagelist  
C:\
```

Информация о файле

```
FileInfo info = new FileInfo("C:\\file.txt");  
FileAttributes attributes = info.Attributes;  
WriteLine(attributes);
```

```
info = new FileInfo("C:\\");  
attributes = info.Attributes;  
WriteLine(attributes);
```

Archive

Hidden, System, Directory