|  |
| --- |
| , RD Dep. |
| Конспект и раздаточный материал  NET.C#.09 Работа с файловой системой |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| REVISION HISTORY | | | | | |
| Ver. | Description of Change | Author | Date | Approved | |
| Name | Effective Date |
| 1.0 | Initial version | Анжелика Кравчук |  |  |  |
| 1.1 | Review and corrections. Home work tasks | Владимир Тихон |  |  |  |

Contents

[1. Урок 1. Доступ к файловой системе 3](#_Toc300749138)

[1.1. Управление файлами 3](#_Toc300749139)

[1.2. Чтение из файлов и запись в файлы 4](#_Toc300749140)

[1.3. Управление директориями 6](#_Toc300749141)

[1.4. Управление путями 7](#_Toc300749142)

[1.5. Использование общих диалоговых окон файловой системы 8](#_Toc300749143)

[2. Урок 2. Чтение и запись файлов с помощью потоков 9](#_Toc300749144)

[2.1. Что такое потоки? 9](#_Toc300749145)

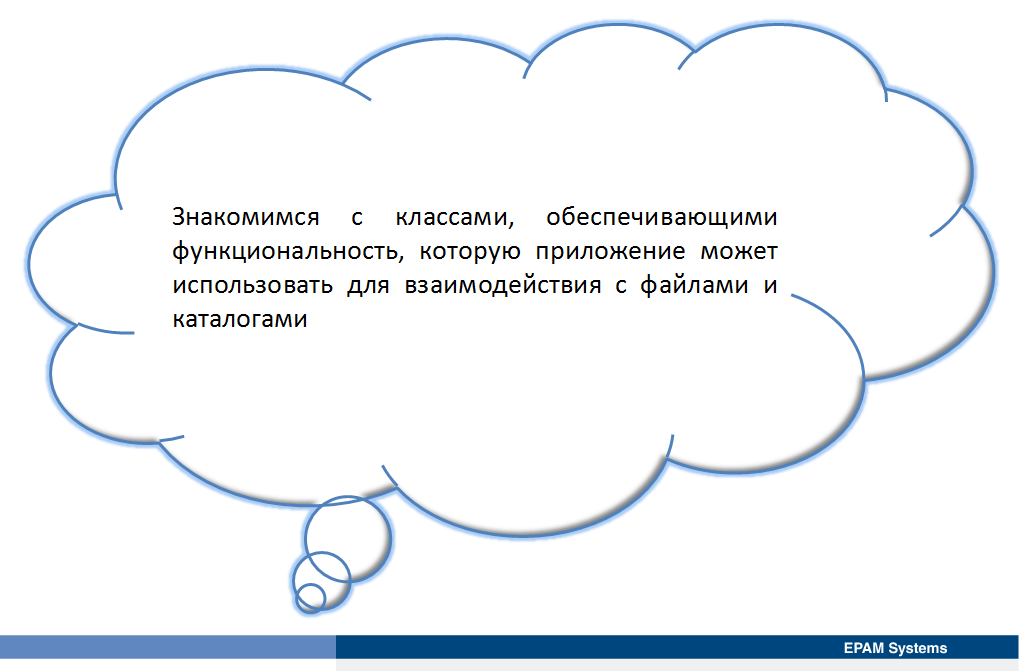
[2.2. Чтение и запись двоичных данных 11](#_Toc300749146)

[2.3. Чтение и запись текста 13](#_Toc300749147)

[2.4. Чтение и запись примитивных типов данных 15](#_Toc300749148)

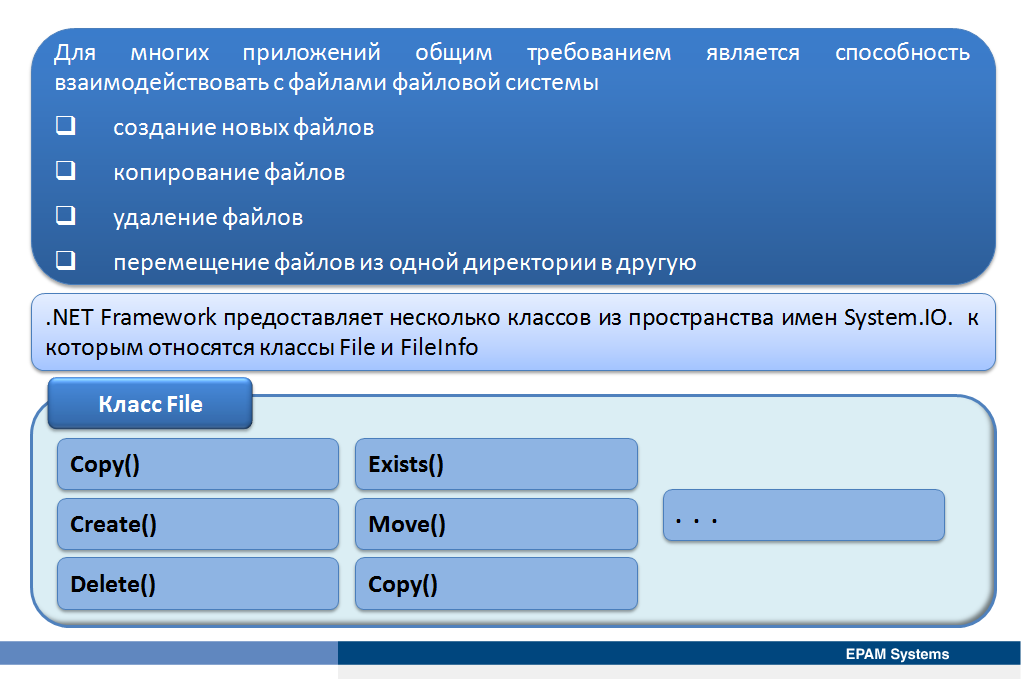
[2.5. Демонстрация: Чтение и запись в файлы 17](#_Toc300749149)

# Урок 1. Доступ к файловой системе



Урок знакомит с классами, обеспечивающими функциональность, которую приложение может использовать для взаимодействия с файлами и каталогами.

## Управление файлами



Для многих приложений общим требованием является способность взаимодействовать с файлами, которые хранятся в файловой системе. Это может включать в себя создание нового файла, копирование или удаление файлов, а также перемещение файла из одной директории в другую. Для упрощения этих взаимодействий .NET Framework предоставляет несколько классов из пространства имен System.IO. к которым относятся классы File и FileInfo.

**Класс File.** Класс File это служебный класс, предоставляющий различные связанные с файлами функции, доступные через статические методы. В следующей таблице описаны некоторые из ключевых методов, предоставляемые классом File, а также показаны некоторые примеры кода.

|  |  |  |
| --- | --- | --- |
| **Метод** | **Описание** | **Пример кода** |
| **AppendAllText** | Позволяет открыть существующий файл, добавить текст в этот файл, а затем закрыть файл, выполнив все действия в одну операцию. | string filePath = "...";  string fileContents = "...";  File.AppendAllText(filePath,  fileContents); |
| **Copy** | Позволяет cкопировать существующий файл в новое место. | string sourceFile = "...";  string destFile = "...";  bool overwrite = false;  File.Copy(sourceFile,destFile,  overwrite); |
| **Create** | Позволяет создать новый файл файловой системы Windows. Метод возвращает объект FileStream, который позволяет взаимодействовать с файлом с помощью потоковой модели. | string filePath = "...";  int bufferSize = 128;  FileStream file = File.Create(filePath,bufferSize,  FileOptions.None); |
| **Delete** | Позволяет удалить файл из файловой системы Windows. | string filePath = "...";  File.Delete(filePath); |
| **Exists** | Позволяет определить, существует ли указанный файл. | string filePath = "...";  bool exists = File.Exists(filePath); |
| **GetCreationTime** | Позволяет получить время создания файла. | string filePath = "...";  DateTime time = File.GetCreationTime(filePath); |
| **GetLastAccessTime** | Позволяет получить время последнего доступа к файлу. | string filePath = "...";  DateTime time = File.GetLastAccessTime(filePath); |
| **Move** | Позволяет переместить файл в новое место. Этот метод можно также использовать для переименования файлов. | string sourceFile = "...";  string destFile = "...";  File.Move(sourceFile,destFile); |
| **ReadAllText** | Позволяет читать весь текст из файла в строковую переменную. | string filePath = "...";  string fileContents = File.ReadAllText(filePath); |
| **SetCreationTime** | Позволяет установить время создания файла. | string filePath = "...";  File.SetCreationTime(filePath,  DateTime.Now); |
| **SetLastAccessTime** | Позволяет установить время последнего доступа к файлу | string filePath = "...";  File.SetLastAccessTime(filePath,  DateTime.Now); |
| **WriteAllText** | Позволяет создать новый файл, записать текст в этот файл, а затем закрыть файл, выполнив все действия в одну операцию. | string filePath = "...";  string fileContents = "...";  File.WriteAllText(filePath,  fileContents); |

**Класс FileInfo.** Класс FileInfo предоставляет несколько свойств и экземплярных методов, которые позволяют создавать, копировать, перемещать файлы и обрабатывать их содержимое. При создании экземпляра класса FileInfo, необходимо указать путь к файлу в файловой системе. В следующем примере кода показано, как создать новый объект класса FileInfo для управления файлом myFile.txt в папке C:\Temp.

string filePath = @"C:\Temp\myFile.txt";

FileInfo file = new FileInfo(filePath);

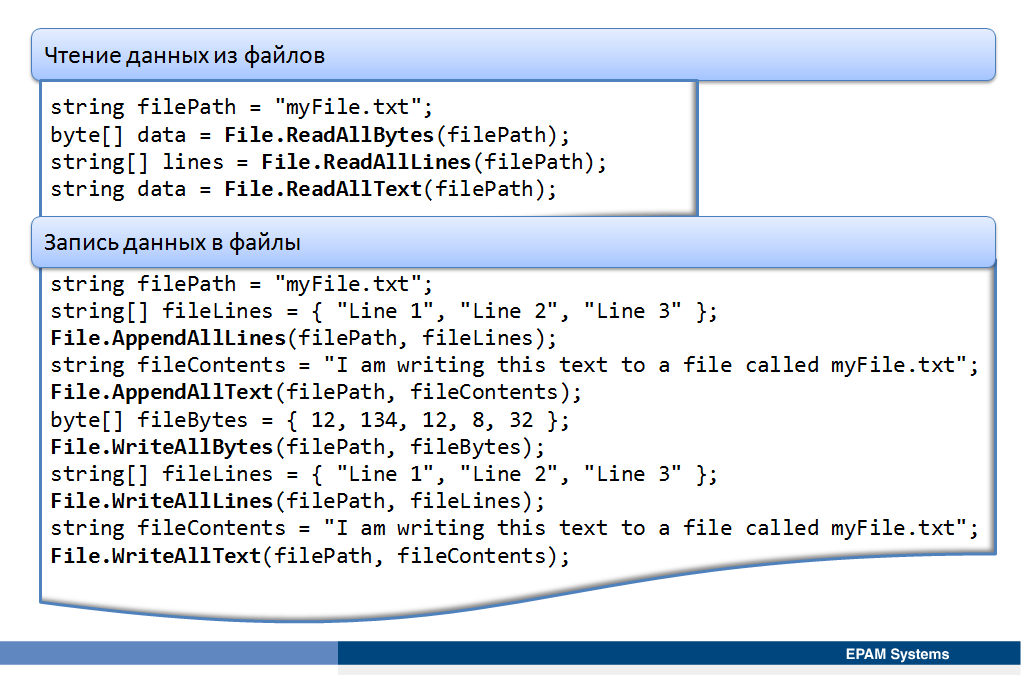
Объект FileInfo можно использовать в качестве оболочки для файла, который предоставляет различные данные и функции через свойства и методы. Класс FileInfo можно также использовать для создания новых файлов. В следующей таблице описаны некоторые из ключевых свойств и методов, а также приведены некоторые примеры кода.

|  |  |  |
| --- | --- | --- |
| **Член** | **Описание** | **Пример кода** |
| **CreationTime (свойство)** | Позволяет получить или установить время создания для конкретного файла. | string filePath = "...";  FileInfo file = new FileInfo(filePath);  file.CreationTime = DateTime.Now;  ...  DateTime time = file.CreationTime; |
| **CopyTo**  **(метод)** | Позволяет копировать файл в новое место в файловой системе. | string filePath = "...";  FileInfo file = new FileInfo(filePath);  string destPath = "...";  file.CopyTo(destPath); |
| **Delete**  **(свойство)** | Позволяет удалить файл. | string filePath = "...";  FileInfo file = new FileInfo(filePath);  file.Delete(); |
| **DirectoryName (свойство)** | Позволяет получить путь к каталогу файла. | string filePath = "...";  FileInfo file = new FileInfo(filePath);  string dirPath = file.DirectoryName; |
| **Exists**  **(свойство)** | Позволяет определить, существует ли заданный файл. | string filePath = "...";  FileInfo file = new FileInfo(filePath);  bool exists = file.Exists; |
| **Extension (свойство)** | Позволяет получить расширение файла. | string filePath = "...";  FileInfo file = new FileInfo(filePath);  string ext = file.Extension; |
| **Length**  **(свойство)** | Позволяет получить длину файла в байтах. | string filePath = "...";  FileInfo file = new FileInfo(filePath);  long length = file.Length; |
| **Name**  **(свойство)** | Позволяет получить имя файла. | string filePath = "...";  FileInfo file = new FileInfo(filePath);  string name = file.Name; |
| **Open**  **(метод)** | Позволяет открыть файл файловой системы Windows. Метод возвращает объект FileStream, который позволяет взаимодействовать с файлом с помощью потоковой модели. | string filePath = "...";  FileInfo file = new FileInfo(filePath);  FileStream stream = file.Open(FileMode.OpenOrCreate); |

<http://go.microsoft.com/fwlink/?LinkId=192915>

<http://go.microsoft.com/fwlink/?LinkId=192916>

## Чтение из файлов и запись в файлы



Классы File и FileInfo предоставляют несколько методов, которые можно использовать для чтения и записи в файл.

Класс File содержит статические методы, которые можно использовать для выполнения атомарных операций прямого чтения и записи в файлы. Эти методы атотомарны, поскольку оборачивают несколько основных функций в один вызов метода. Например, метод AppendAllLines оборачивает операции получения дескриптора файла, открытия потока файла, записи данных в файл, а затем освобождение дескриптора файла.

Класс FileInfo содержит экземплярные методы, которые при чтении и записи в файлы полагаются на классы FileStream и StreamReader. В этом разделе основное внимание уделяется статическим методам, которые предоставлены классом File, которые не используют потоки и обеспечивают единые атомарные операции.

**Чтение из файлов.** При использовании класса File для чтения данных из файла, существует много альтернативных методов, которые можно использовать, каждый из которых предлагает различное поведение. Ниже перечислены некоторые из этих методов:

* Метод ReadAllBytes позволяет прочитать содержимое файла в виде данных типа byte, которые сохраняются в массив. В следующем примере показано, как прочитать содержимое файла myFile.txt в массив data.

string filePath = "myFile.txt";

byte[] data = File.ReadAllBytes(filePath);

* Метод ReadAllLines позволяет прочитать текстовый файл от начала до конца, строка за строкой, и сохранить каждую строку в массив строк. В следующем примере показано, как прочитать содержимое файла myFile.txt и сохранить каждую строку в массиве строк lines.

string filePath = "myFile.txt";

string[] lines = File.ReadAllLines(filePath);

* Метод ReadAllText позволяет читать файл от начала до конца и сохранять данные из файла в строковую переменную. В следующем примере оказано, как прочитать содержимое файла myFile.txt и сохранить данные в строку data.

string filePath = "myFile.txt";

string data = File.ReadAllText(filePath);

**Запись в файлы.** При записи данных в файл с помощью класса File, доступны несколько вариантов методов в зависимости от типа записываемых данных. С каждой опцией можно добавлять данные в существующий файл или создавать новый файл, а затем выполнять операцию записи. Ниже перечислены некоторые из методов.

* Метод AppendAllLines используется для записи содержимого массива строк в текстовый файл. Если указанный путь не существует, будет создан новый файл. В следующем примере показано, как записать содержание массива строк fileLines в файл myFile.txt.

string filePath = "myFile.txt";

string[] fileLines = { "Line 1", "Line 2", "Line 3" };

File.AppendAllLines(filePath, fileLines);

* Метод AppendAllText используется для записи содержимого строковой переменной в текстовый файл. Как и в методе AppendAllLines, если файл не существует, будет создан новый файл, а затем выполнится операция записи. В следующем примере кода показано, как записать содержимое строковой переменной fileContents в файл myFile.txt.

string filePath = "myFile.txt";

string fileContents = "I am writing this text to a file called myFile.txt";

File.AppendAllText(filePath, fileContents);

* Метод WriteAllBytes используется для записи содержимого массива в двоичный файл. Если файл уже существует, содержимое файла будет перезаписано. В следующем примере кода показано, как записать содержимое массива fileBytes в новый файл с именем myFile.txt.

string filePath = "myFile.txt";

byte[] fileBytes = { 12, 134, 12, 8, 32 };

File.WriteAllBytes(filePath, fileBytes);

* Метод WriteAllLines ведет себя аналогично методу AppendAllLines, т.е. позволяет записывать содержимое массива строк в текстовый файл. Основным отличием является то, что, если файл существует, его содержимое будет записано заново. Если файл не существует, будет создан новый файл. В следующем примере кода показано, как записать содержимое массива строк fileLines в новый файл с именем myFile.txt.

string filePath = "myFile.txt";

string[] fileLines = { "Line 1", "Line 2", "Line 3" };

File.WriteAllLines(filePath, fileLines);

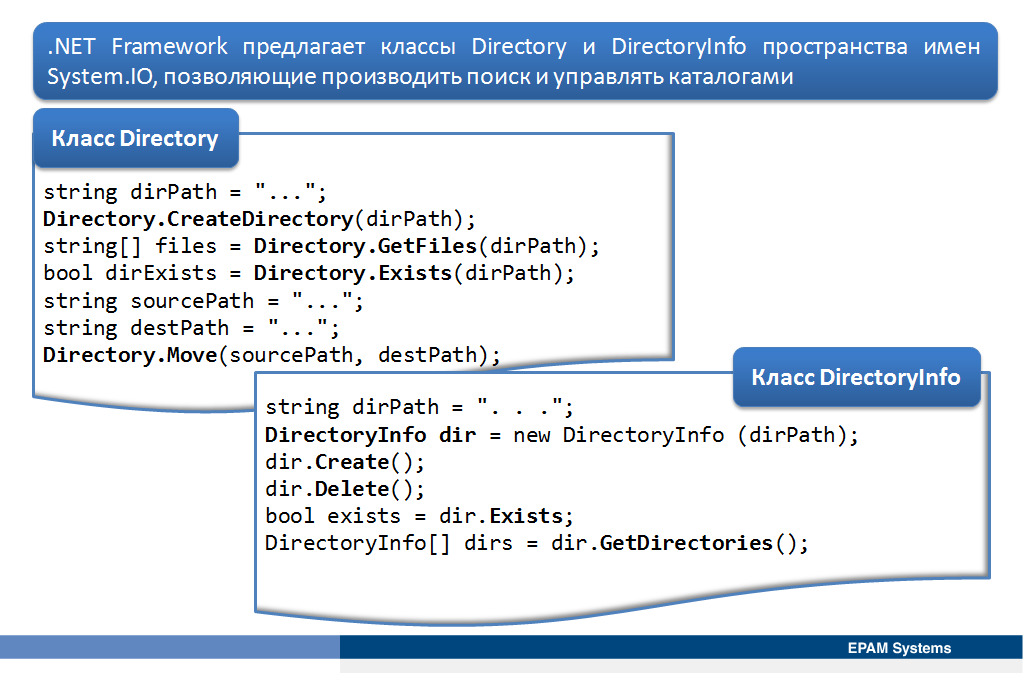
* Метод WriteAllText ведет себя аналогично методу AppendAllText, т.е. записывает содержимое строки переменной в текстовый файл. Основным отличием является то, что, если файл существует, файл будет перезаписан. Если файл не существует, будет создан новый файл. В следующем примере кода показано, как записать содержимое строковой переменной fileContents в новый файл с именем myFile.txt.

string filePath = "myFile.txt";

string fileContents = "I am writing this text to a file called myFile.txt";

File.WriteAllText(filePath, fileContents);

## Управление директориями



.NET Framework предлагает пару классов, подобных классам File и FileInfo, позволяющих производить поиск и управлять каталогами. Если необходимо создать новую папку, удалить существующий каталог или перечислить его содержимое, можно воспользоваться классами Directory и DirectoryInfo пространства имен System.IO.

**Класс Directory.** Класс Directory, как и класс File, является служебным классом, который обеспечивает различные операции, позволяющие управлять папками и каталогами. Класс Directory предоставляет свои функции в рамках статических методов. В следующей таблице приведены некоторые из методов класса и даны примеры кода.

|  |  |  |
| --- | --- | --- |
| **Метод** | **Описание** | **Пример кода** |
| **CreateDirectory** | Позволяет создавать все еще не существующие каталоги, указаные в пути. | string dirPath = @"C:\NewFolder\SubFolder";  Directory.CreateDirectory(dirPath); |
| **DeleteDirectory** | Позволяет удалить один или несколько каталогов файловой системы. | string dirPath = @"C:\Users\Student\" + "MyDirectory";  bool deleteSubFolders = true;  Directory.Delete(dirPath,deleteSubFolders); |
| **GetDirectories** | Позволяет получить все подкаталоги по указанному пути. | string dirPath = "...";  string[] dirs = Directory.GetDirectories(dirPath); |
| **GetFiles** | Позволяет получить все файлы по указанному пути. | string dirPath = "...";  string[] files = Directory.GetFiles(dirPath); |
| **Exists** | Позволяет определить, существует ли каталог по указанному пути. | string dirPath = "...";  bool dirExists = Directory.Exists(dirPath); |
| **Move** | Позволяет переместить каталог. Нельзя использовать метод для перемещения каталогов с разных носителей. | string sourcePath = "...";  string destPath = "...";  Directory.Move(sourcePath, destPath); |

**Класс DirectoryInfo.** Класс DirectoryInfo предоставляет несколько свойств и экземплярных методов, которые позволяют работать с каталогами. Как и в классе FileInfo, при создании экземпляра класса DirectoryInfo, обычно уазывается путь к каталогу файловой системы. В следующем примере кода показано, создание экземпляра класса DirectoryInfo.

string dirPath = @"C:\Users\Student\Music\";

DirectoryInfo dir = new DirectoryInfo (dirPath);

Объект DirectoryInfo можно использовать в качестве обертки для каталога, который предоставляет различные данные и функции через свойства и методы. Класс DirectoryInfo можно также использовать для создания нового каталога. Например, в следующем примере показано, как можно определить существование каталога, и если его не существует, как его можно создать.

string dirPath = @"C:\Users\Student\Music\";

DirectoryInfo dir = new DirectoryInfo(dirPath);

if (!dir.Exists)

{

dir.Create();

}

В следующей таблице описаны некоторые из ключевых свойств и методов, даны некоторые примеры кода.

|  |  |  |
| --- | --- | --- |
| **Член** | **Описание** | **Пример кода** |
| **Create**  **(метод)** | Позволяет создавать каталоги по указанному пути. Если каталог уже существует, метод игнорируется. | string dirPath = "...";  DirectoryInfo dir = new DirectoryInfo(dirPath);  dir.Create(); |
| **Delete**  **(метод)** | Позволяет удалить несколько каталогов. Если каталог не может быть найден, генерируется исключение DirectoryNotFoundException. | string dirPath = "...";  DirectoryInfo dir = new DirectoryInfo(dirPath);  dir.Delete(); |
| **Exists**  **(свойство)** | Позволяет определить, существует ли каталог по указанному пути. | string dirPath = "...";  DirectoryInfo dir = new DirectoryInfo(dirPath);  bool exists = dir.Exists; |
| **FullName**  **(свойство)** | Позволяет получить полный путь к каталогу. | string dirPath = "...";  DirectoryInfo dir = new DirectoryInfo(dirPath);  string fullName = dir.FullName; |
| **GetDirectories**  **(метод)** | Позволяет получить все подкаталоги по указанному пути. Этот метод возвращает массив DirectoryInfo, который позволяет использовать каждый из членов DirectoryInfo во всех подкаталогах. | string dirPath = "...";  DirectoryInfo dir = new DirectoryInfo (dirPath);  DirectoryInfo[] dirs = dir.GetDirectories(); |
| **GetFiles**  **(метод)** | Позволяет получить все файлы по указанному пути. Этот метод возвращает массив FileInfo, который позволяет использовать каждый из членов FileInfo во всех файлах каталога. | string dirPath = "...";  DirectoryInfo dir = new DirectoryInfo(dirPath);  FileInfo[] files = dir.GetFiles(); |
| **MoveTo**  **(метод)** | Позволяет переместить каталог. Нельзя использовать метод для перемещения каталогов с разных носителей. | string dirPath = "...";  DirectoryInfo dir = new DirectoryInfo(dirPath);  string destPath = "...";  dir.MoveTo(destPath); |
| **Name**  **(свойство)** | Позволяет получить имя каталога. | string dirPath = "...";  DirectoryInfo dir = new DirectoryInfo(dirPath);  string dirName = dir.Name; |
| **Parent**  **(свойство)** | Позволяет получить родительский каталог. | string dirPath = @"C:\Users\Student\Music\";  DirectoryInfo dir = new DirectoryInfo(dirPath);  DirectoryInfo parentDir = dir.Parent; |

**Перечисление содержимого каталога.** В следующем примере показано, как можно перечислить и показать подробную информацию о всех подкаталогах и файлах, которые они содержат.

string dirPath = @"C:\Users\Student\Documents";

// Get all sub directories in the Documents directory.

string[] subDirs = Directory.GetDirectories(dirPath);

foreach (string dir in subDirs)

{

// Display the directory name.

Console.WriteLine("{0} contains the following files:", dir);

// Get all the files in each directory.

string[] files = Directory.GetFiles(dir);

foreach (string file in files)

{

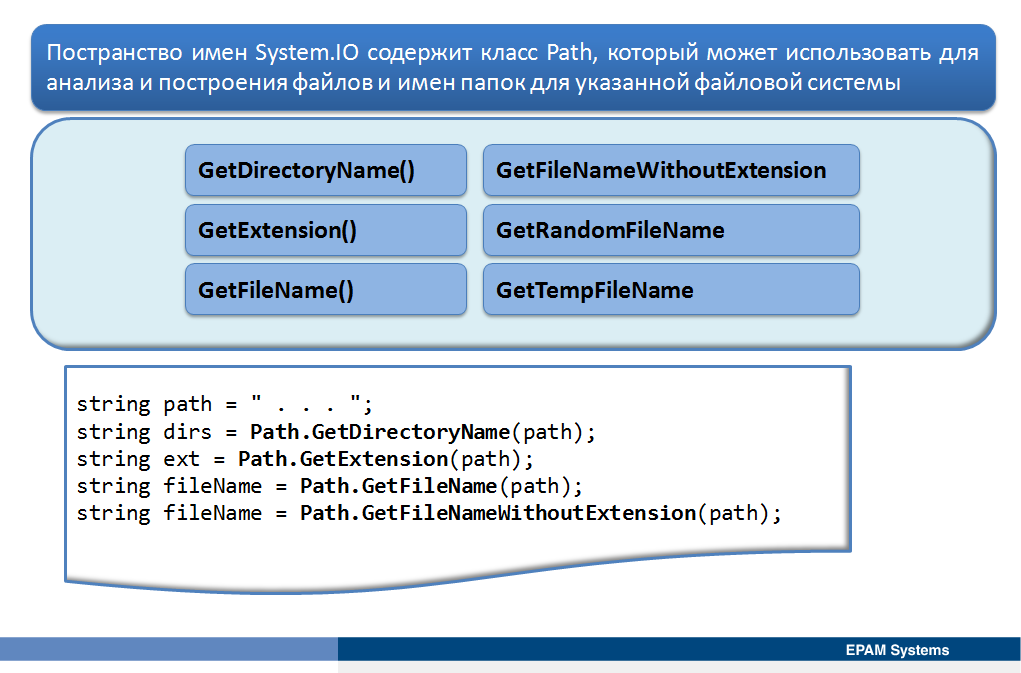
// Display the file name.

Console.WriteLine(file);

}

}

## Управление путями



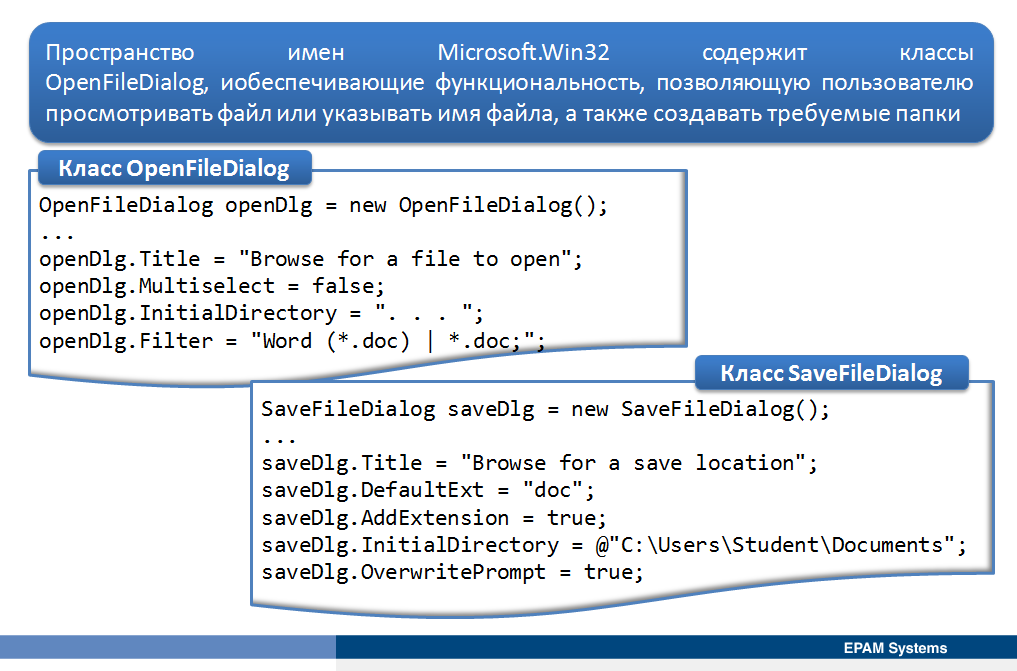
Файлы хранятся в папках. Все файлы и папки имеют названия. Сочетание имени файла и папки, в которой он находится, представляют собой путь к этому файлу. Различные файловые системы могут иметь различные соглашения и правила, согласно которым установлены легальные файл и путь к файлу. Класс Path содержит методы, которые можно использовать для анализа и построения файлов и имен папок для указанной файловой системы.

**Класс Path.** Класс Path предоставляет свою функциональность с помощью различных статических методов. В следующей таблице приведены некоторые из методов и приведены некоторые примеры кода.

|  |  |  |
| --- | --- | --- |
| **Метод** | **Описание** | **Пример кода** |
| **GetDirectoryName** | Позволяет получить все каталоги по указанному пути. | string path = @"C:\Temp\SubFolder\MyFile.txt";  string dirs = Path.GetDirectoryName(path); |
| **GetExtension** | Позволяет получить расширение указанного файла. | string path = @"C:\Temp\SubFolder\MyFile.txt";  string ext = Path.GetExtension(path); |
| **GetFileName** | Позволяет получить имя файла, включая расширение по указанному пути. | string path = @"C:\Temp\SubFolder\MyFile.txt";  string fileName = Path.GetFileName(path); |
| **GetFileNameWithoutExtension** | Позволяет получить имя файла без расширения по указанному пути. | string path = @"C:\Temp\SubFolder\MyFile.txt";  string fileName = Path.GetFileNameWithoutExtension(path); |
| **GetRandomFileName** | Позволяет генерировать случайные папки или файлы. | string fileName = Path.GetRandomFileName(); |
| **GetTempFileName** | Позволяет создать новый временный файл в локальной временной папке Windows. Метод возвращает абсолютный путь к файлу. | string tempFilePath = Path.GetTempFileName(); |
| **GetTempPath** | Позволяет получить путь к локальной временной папке Windows. | string tempPath = Path.GetTempPath(); |

<http://go.microsoft.com/fwlink/?LinkId=192919>

## Использование общих диалоговых окон файловой системы



При создании приложения с графическим интерфейсом неразумно заставлять пользователей вводить длинные, громоздкие пути и имена файлов. Пользователи ожидают возможности переходить к файлам и каталогам через диалоговые окна. Создание диалогового окна открытия или сохранения файла, имеющегося в любом приложении, потребует значительного количества усилий для разработки и тестирования. Для этой цели .NET Framework предоставляет классы OpenFileDialog и SaveFileDialog пространства имен Microsoft.Win32[[1]](#footnote-1).

Классы OpenFileDialog и SaveFileDialog обеспечивают функциональность, позволяющую пользователю просмотривать файл или указывать имя файла, а также создавать любые требуемые папки. Функциональные возможности этих классов доступны через различные свойства и методы, которые можно использовать для настройки поведения диалоговых окон в соответствии с требованиями. Однако следует отметить, что ни диалоговое окно фактически открывает или сохраняет указанный файл; все это делается построением пути и имени файла, которые приложение может использовать, чтобы открыть или сохранить файл.

В следующей таблице описаны некоторые из ключевых свойств, которые являются общими для классов OpenFileDialog и SaveFileDialog.

|  |  |
| --- | --- |
| **Свойства** | **Описание** |
| **CheckFileExists** | Позволяет поручить диалоговое окно, отображающее предупреждение, если пользователь указывает не существующий файл. |
| **FileName** | Позволяет получить или задать путь к файлу, который выбран в диалоговом окне. |
| **Filter** | Позволяет ограничить типы файлов, которые пользователь может выбрать в диалоговом окне. |
| **InitialDirectory** | Позволяет получить или установить каталог по умолчанию, который отображается, когда диалоговое окно показывается впервые. |
| **Title** | Позволяет указать заголовок для диалогового окна. |

Классы OpenFileDialog и SaveFileDialog можно использовать таким же образом, как и любой другой класс .NET Framework. Первым шагом является создание экземпляра класса, как показано в следующем примере.

OpenFileDialog openDlg = new OpenFileDialog();

...

SaveFileDialog saveDlg = new SaveFileDialog();

После создания экземпляра любого класса диалога, можно использовать свойства для настройки его поведения. Большинство свойств, которые предоставляются через оба классы одинаковы, однако есть некоторые исключения, такие как свойство Multiselect класса OpenFileDialog и свойство OverwritePrompt класса SaveFileDialog, показанные в следующем примере.

OpenFileDialog openDlg = new OpenFileDialog();

...

SaveFileDialog saveDlg = new SaveFileDialog();

...

openDlg.Title = "Browse for a file to open";

openDlg.Multiselect = false;

openDlg.InitialDirectory = @"C:\Users\Student\Documents";

openDlg.Filter = "Word (\*.doc) | \*.doc;";

...

saveDlg.Title = "Browse for a save location";

saveDlg.DefaultExt = "doc";

saveDlg.AddExtension = true;

saveDlg.InitialDirectory = @"C:\Users\Student\Documents";

saveDlg.OverwritePrompt = true;

Для появления диалогового окна во время работы приложения необходимо вызвать метод ShowDialog:

...

openDlg.ShowDialog();

...

saveDlg.ShowDialog();

И, наконец, чтобы получить путь, выбранный пользователем, запросить свойство FileName:

...

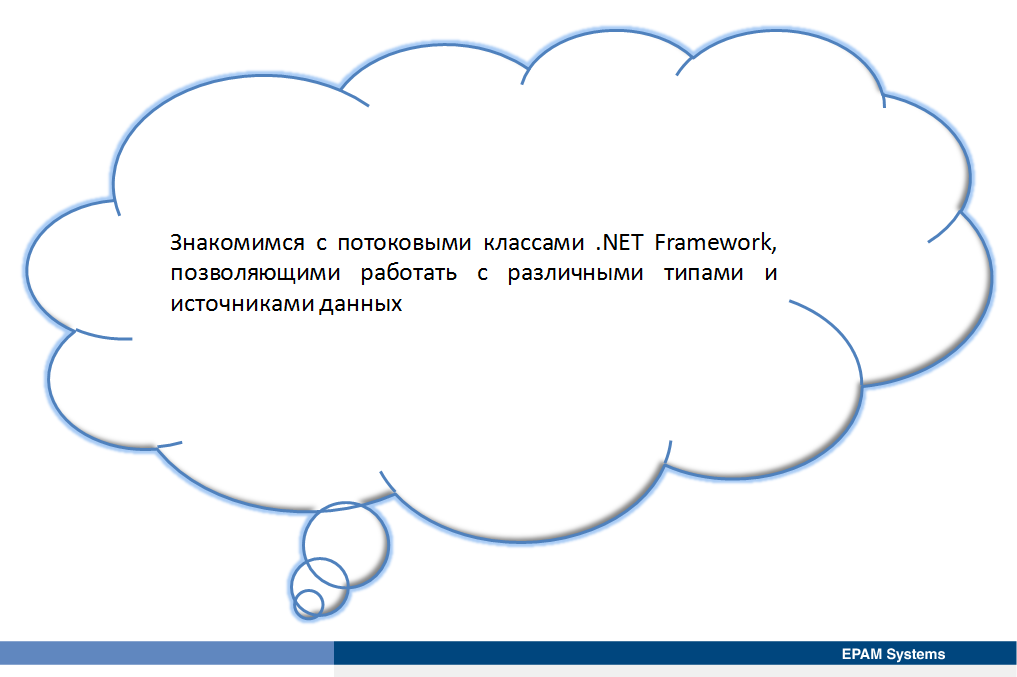
string selectedFileName = openDlg.FileName;

...

string selectedFileName = saveDlg.FileName;

В зависимости от того, выбрал пользователь файл или просто закрыл окно, значение, которое возвращается из свойства FileName может быть действительным абсолютным путем или пустой строкой. Таким образом, перед использованием результата следует выполнить соответствующую проверку.

# Урок 2. Чтение и запись файлов с помощью потоков



Чтение и запись данных в одной атомарной операции, как описано в предыдущем уроке является возможным с небольшим количеством данных. Однако, когда необходимо работать с большими объемами данных, такие операции являются неэффективными и могут потреблять слишком много ресурсов.

## Что такое потоки?



При работе с данными, будь то данные хранимые в файле файловой системы или на веб-сервере, доступном через соединение HTTP, данные иногда становится слишком большими, чтобы загружать их в память и передавать одной атомарной операцией. Например, можно представить попытку загрузить в память видео файл в 100-гигабайт в одну операцию. Мало того, что такая операция занимает много времени, она также потребляют большой объем памяти.

Для решения обозначенной проблемы .NET Framework позволяет использовать потоки. Поток представляет собой последовательность байтов, которые могут поступать из файла файловой системы, сети связи или памяти. Потоки позволяют считывать или записывать данные в источник данных посредством небольших управляемых пакетов данных. Как правило, потоки обеспечивают следующие операции:

* Чтение из потока – это перенос информации из потока в структуру данных, такую как массив байтов.
* Запись в поток – это передача данных из структуры данных в поток.
* Поиск в потоке – это выяснение и изменение текущей позиции внутри потока. Возможность поиска зависит от вида резервного хранилища потока. Например, в сетевых потоках отсутствует унифицированное представление текущего положения, поэтому обычно они не поддерживают поиск.

.NET Framework предоставляет несколько потоковых классов, позволяющих работать с различными данными и источниками данных. Для выбора какого-либо из них, необходимо учитывать следующее:

* Какой тип данных читается или записывается, например, двоичный или буквенно-цифровой.
* Где хранятся данные, например, в локальной файловой системе, в памяти или на веб-сервере в сети.

Библиотека классов .NET Framework предоставляет несколько классов из пространства имен System.IO, которые можно использовать для чтения и записи файлов с помощью потоков. На самом высоком уровне абстракции (Рис. 1.), класс Stream определяет общие функциональные возможности, которые обеспечивают все потоки; он представляет универсальное представление последовательности байтов вместе с операциями и свойствами, которые обеспечивают все потоки. Класс Stream и его производные классы предоставляют универсальное представление различных типов ввода и вывода, изолируя программиста от отдельных сведений операционной системы и базовых устройств.

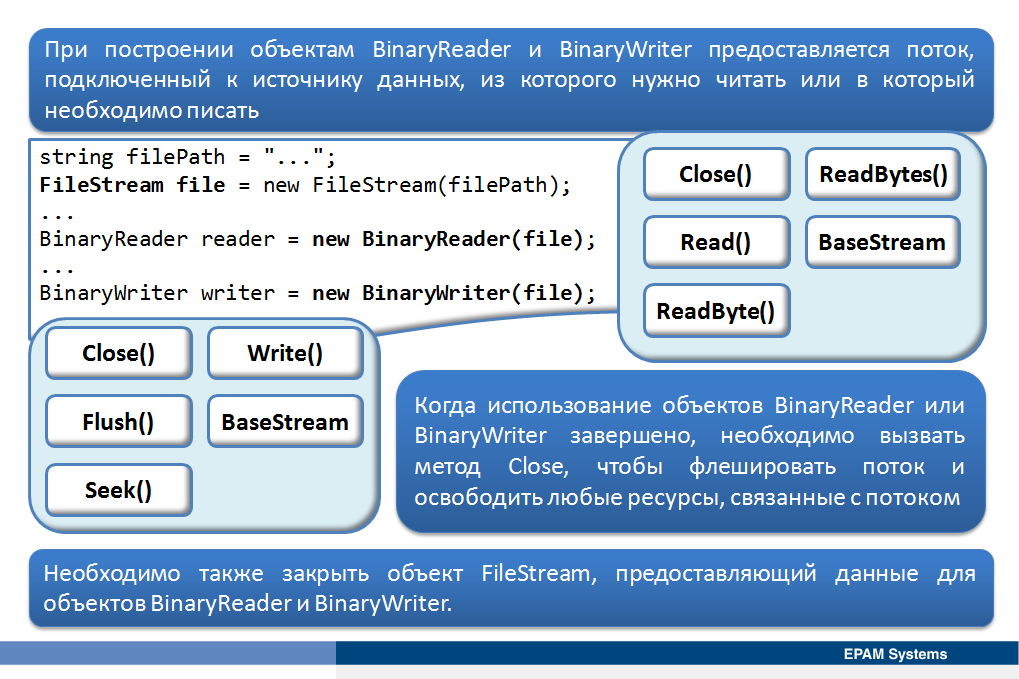
Рис. 1.

Внутренне, объект Stream поддерживает указатель, который ссылается на текущее местоположение в источнике данных. Когда впервые строится объект Stream для источника данных, этот указатель позиционируется перед первым байтом. Когда происходит чтение или запись данных, класс Stream продвигает этот указатель к концу данных, которые читаются или записываются. Класс Stream нельзя использовать напрямую. Вместо этого, нужно реализовать специализации этого класса, которые оптимизированы для выполнения потокового ввода/вывода для конкретных типов источников данных. Таким образом, в каждом резервном хранилище (резервное хранилище – устройство хранения информации, например диск или память) используется собственный поток как реализация класса [Stream](http://msdn.microsoft.com/ru-ru/library/system.io.stream(v=VS.90).aspx). Например, класс FileStream[[2]](#footnote-2) реализует поток, который использует файл на диске в качестве источника данных, класс MemoryStream реализует поток, который использует блок памяти в качестве источника данных. Потоки, связанные с резервными хранилищами, называются базовыми. В конструкторах базовых потоков есть параметры, необходимые для присоединения потока к резервному хранилищу. Например, класс [FileStream](http://msdn.microsoft.com/ru-ru/library/system.io.filestream(v=VS.90).aspx) имеет конструктор, задающий путь, который определяет порядок совместного использования файла процессами и т. д.

Структура классов [System.IO](http://msdn.microsoft.com/ru-ru/library/system.io(v=VS.90).aspx) предоставляет упрощенное составление потоков. Базовые потоки могут быть присоединены к одному или нескольким сквозным потокам, которые обеспечивают требуемую функциональность. Модуль чтения или записи может быть присоединен к концу цепочки, что обеспечит легкое считывание или запись предпочитаемых типов.

http://go.microsoft.com/fwlink/?LinkId=192920

## Чтение и запись двоичных данных



Поток, который устанавливается с помощью объекта FileStream всего лишь сырая последовательность байтов. Если файл содержит структурированные данные, необходимо преобразовать последовательность байтов в соответствующие типы. Это может быть трудоемкой, подверженной ошибкам задачей. Библиотека классов .NET Framework содержит классы, используемые для чтения и записи текстовых данных и примитивных типов в поток, который открыт с помощью объекта FileStream. Эти классы включают StreamReader, StreamWriter, BinaryReader и BinaryWriter.

**Классы BinaryReader и BinaryWriter.** Многие приложения хранят данные в сырой бинарной форме, поскольку двоичная запись быстая и занимает мало места на диске. Однако такая форма не читаема для человека. Воспользоваться мощью бинарного формата в .NET приложениях позволяют классы BinaryReader и BinaryWriter. При построении объектам BinaryReader и BinaryWriter предоставляется поток, подключенный к источнику данных, из которого нужно читать или в который необходимо писать. В следующем примере показана инициализация объектов классов BinaryReader и BinaryWriter.

string filePath = "...";

FileStream file = new FileStream(filePath);

...

BinaryReader reader = new BinaryReader(file);

...

BinaryWriter writer = new BinaryWriter(file);

После того как создан объект BinaryReader, можно использовать его члены для чтения бинарных данных. В следующей таблице описаны некоторые из ключевых членов этого класса.

|  |  |
| --- | --- |
| **Член** | **Описание** |
| **BaseStream**  **(свойство)** | Получает доступ к базовому потоку, используемому объектом BinaryReader. |
| **Close**  **(метод)** | Закрывает объект BinaryReader и базовый поток. |
| **Read**  **(метод)** | Выполняет чтение символов из базового потока и перемещает текущую позицию в потоке вперед в соответствии с используемой кодировкой и конкретным символом в потоке, чтение которого выполняется в настоящий момент. |
| **ReadByte**  **(метод)** | Считывает из текущего потока следующий байт и перемещает текущую позицию в потоке на один байт вперед. |
| **ReadBytes**  **(метод)** | Считывает указанное количество байтов из текущего потока в массив байтов и перемещает текущую позицию на это количество байтов. |

Класс BinaryReader содержит еще 16 методов (ReadInt16, ReadDouble, ReadChar и т.д.), которые могут читать бинарный поток и преобразовывать данные в различные примитивные типы данных, доступные в C#.

Аналогичным образом, объект BinaryWriter предоставляет различные члены для записи данные в основной поток. В следующей таблице описаны некоторые из ключевых членов.

|  |  |
| --- | --- |
| **Член** | **Описание** |
| **BaseStream**  **(свойство)** | Получает доступ к базовому потоку, который используется объектом BinaryWriter. |
| **Close**  **(метод)** | Закрывает объект BinaryWriter и базовый поток. Любые данные из буфера сбрасываются в базовый поток. |
| **Flush**  **(метод)** | Явно флеширует какие-либо данные из текущего буфера в основной поток. |
| **Seek**  **(метод)** | Установливает позицию в текущем потоке, таким образом, чтобы писать в конкретные байты. |
| **Write**  **(метод)** | Записывает данные в поток, продвигаясь в нем. Метод предоставляет несколько перегрузок, позволяющих записывать все примитивные типы данных в поток. |

Когда использование объектов BinaryReader или BinaryWriter завершено, необходимо вызвать метод Close, чтобы флешировать поток и освободить любые ресурсы, связанные с потоком. Необходимо также закрыть объект FileStream, предоставляющий данные для объектов BinaryReader и BinaryWriter.

**Запись двоичных данных.** В следующем примере показано использование классов BinaryWriter и FileStream для записи набора из четырех байтов целых чисел в файл.

string destinationFilePath = @"C:\Users\Student\Documents\BinaryDataFile.bin";

// Collection of bytes.

byte[] dataCollection = { 1, 4, 6, 7, 12, 33, 26, 98, 82, 101 };

// Create a FileStream object so that you can interact with the file system.

FileStream destFile = new FileStream(

destinationFilePath, // Pass in the destination path.

FileMode.Create, // Always create new file.

FileAccess.Write); // Only perform writing.

// Create a BinaryWriter object passing in the FileStream object.

BinaryWriter writer = new BinaryWriter(destFile);

// Write each byte to stream.

foreach (byte data in dataCollection)

{

writer.Write(data);

}

// Close both streams to flush the data to the file.

writer.Close();

destFile.Close();

Код создает файл со следующим содержимым.



**Чтение двоичных данных.** В следующем примере показано использование классов BinaryReader и FileStream для чтения из файла, содержащего набор байтов. В этом примере используется метод Read для продвижения через поток байтов в файле.

// Source file path.

string sourceFilePath = @"C:\Users\Student\Documents\BinaryDataFile.bin";

// Create a FileStream object so that you can interact with the file system.

FileStream sourceFile = new FileStream(sourceFilePath, // Pass in the source file path.

FileMode.Open, // Open an existing file.

FileAccess.Read);// Read an existing file.

// Create a BinaryWriter object passing in the FileStream object.

BinaryReader reader = new BinaryReader(sourceFile);

// Store the current position of the stream.

int position = 0;

// Store the length of the stream.

int length = (int)reader.BaseStream.Length;

// Create an array to store each byte from the file.

byte[] dataCollection = new byte[length];

int returnedByte;

while ((returnedByte = reader.Read()) != -1)

{

// Set the value at the next index.

dataCollection[position] = (byte)returnedByte;

// Advance our position variable.

position += sizeof(byte);

}

// Close the streams to release any file handles.

reader.Close();

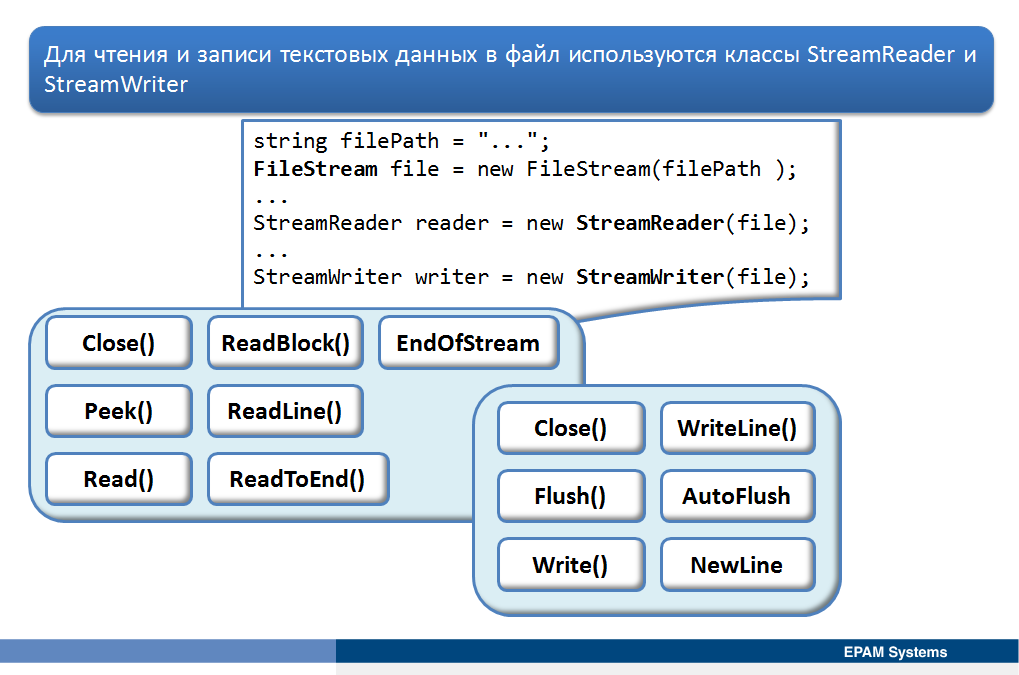
sourceFile.Close();

Если операции чтения из файла или записи в файл генерируют исключение, следует убедиться, что потоки и дескрипторы файлов освобождены. Для освобождения ресурсов можно использовать блок **try**/**finally**. Как правило, в блок **try** следует поместить логику, которая осуществляет чтение или запись, и расположить логику, которая закрывает потоки и высвобождает дискриптор файла, в блок **finally**.

<http://msdn.microsoft.com/en-us/library/system.io.binarywriter.aspx>

<http://msdn.microsoft.com/en-us/library/system.io.binaryreader.aspx>

## Чтение и запись текста



В дополнение к хранению в двоичном виде данные можно также хранить в виде обычного текста. Процесс для чтения и записи текстовых данных в файл очень похож на чтение и запись двоичных данных, только при этом используются классы StreamReader и StreamWriter.

Класс Console, который используется для чтения и записи на консоль содержит свойство In StreamReader и свойство Out StreamWriter. Метод Console.ReadLine читает текстовые данные из потока, который определяет свойство In, а метод Console.WriteLine записывает текстовые данные в поток, который идентифицирует свойство Out.

**Классы StreamReader и StreamWriter.** Аналогично использованию классов BinaryReader и BinaryWriter, при инициализации классов StreamReader или StreamWriter следует предоставить объект потока для обработки взаимодействия с источником данных, как показано в следующем примере.

string destinationFilePath = "...";

FileStream file = new FileStream(destinationFilePath);

...

StreamReader reader = new StreamReader(file);

...

StreamWriter writer = new StreamWriter(file);

В следующей таблице описаны некоторые из ключевых членов, которые обеспечивает класс StreamReader для чтения текста из основного потока.

|  |  |
| --- | --- |
| **Член** | **Описание** |
| **Close**  **(метод)** | Закрывает объект StreamReader и базовый поток. |
| **EndOfStream (свойство)** | Определяет, достигнут ли конец потока. |
| **Peek**  **(метод)** | Получает следующий доступный символ в потоке, но не использует его. |
| **Read**  **(метод)** | Выполняет чтение следующего символа из входного потока и перемещает положение символа на одну позицию вперед. |
| **ReadBlock**  **(метод)** | Выполняет чтение из текущего потока блок символов начиная с указанного индекса. |
| **ReadLine**  **(метод)** | Выполняет чтение строки символов из текущего потока и возвращает данные в виде строки. |
| **ReadToEnd**  **(метод)** | Считывает поток от текущего положения до конца. |

В следующей таблице приведены некоторые из ключевых членов, которые обеспечивает класс StreamWriter для записи текста в поток.

|  |  |
| --- | --- |
| **Член** | **Описание** |
| **AutoFlush**  **(свойство)** | Получает или задает значение, определяющее, будет ли StreamWriter флешировать буфер в основной поток после каждого вызова [StreamWriter.Write](http://msdn.microsoft.com/ru-ru/library/system.io.streamwriter.write.aspx). |
| **Close**  **(метод)** | Закрывает текущий объект StreamWriter и базовый поток. |
| **Flush**  **(метод)** | Очищает все буферы для текущего средства записи и вызывает запись всех данных буфера в основной поток. |
| **NewLine**  **(свойство)** | Получает или задает признак конца строки, используемой текущим TextWriter. |
| **Write**  **(метод)** | Записывать данные в поток и продвигатся в нем. |
| **WriteLine**  **(метод)** | Позволяет записывать данные в поток, за которыми следует признак конца строки, и продвинуться в потоке. |

Методы Write и WriteLine обеспечивают несколько перегрузок, которые позволяют создавать различные типы данных, кроме текстовых.

**Чтение текста.** В следующем примере кода показано использование классов StreamReader и FileStream для чтения текстового файла. В этом примере используются методы Peek и Read для получения вручную каждого символа в файле.

string sourceFilePath = @"C:\Users\Student\Documents\TextDataFile.txt";

// Create a FileStream object so that you can interact with the file system.

FileStream sourceFile = new FileStream(

sourceFilePath, // Pass in the source file path.

FileMode.Open, // Open an existing file.

FileAccess.Read);// Read an existing file.

StreamReader reader = new StreamReader(sourceFile);

StringBuilder fileContents = new StringBuilder();

// Check to see if the end of the file has been reached.

while (reader.Peek() != -1)

{

// Read the next character.

fileContents.Append((char)reader.Read());

}

// Store the file contents in a new string variable.

string data = fileContents.ToString();

// Always close the underlying streams release any file handles.

reader.Close();

sourceFile.Close();

Следующий пример кода обеспечивает альтернативный подход к получению вручную каждого символа из потока, с помощью метода ReadToEnd.

string sourceFilePath = @"C:\Users\Student\Documents\TextDataFile.txt";

string data;

// Create a FileStream object so that you can interact with the file system.

FileStream sourceFile = new FileStream(

sourceFilePath, // Pass in the source file path.

FileMode.Open, // Open an existing file.

FileAccess.Read);// Read an existing file.

StreamReader reader = new StreamReader(sourceFile);

// Read the entire file into a single string variable.

data = reader.ReadToEnd();

// Always close the underlying streams release any file handles.

reader.Close();

sourceFile.Close();

**Запись текста.** В следующем примере кода показано использование классов StreamWriter и FileStream для записи строки в новый файл файловой системы.

string destinationFilePath = @"C:\Users\Student\Documents\TextDataFile.txt";

string data = "Hello, this will be written in plain text";

// Create a FileStream object so that you can interact with the file

// system.

FileStream destFile = new FileStream(

destinationFilePath, // Pass in the destination path.

FileMode.Create, // Always create new file.

FileAccess.Write); // Only perform writing.

// Create a new StreamWriter object.

StreamWriter writer = new StreamWriter(destFile);

// Write the string to the file.

writer.WriteLine(data);

// Always close the underlying streams to flush the data to the file

// and release any file handles.

writer.Close();

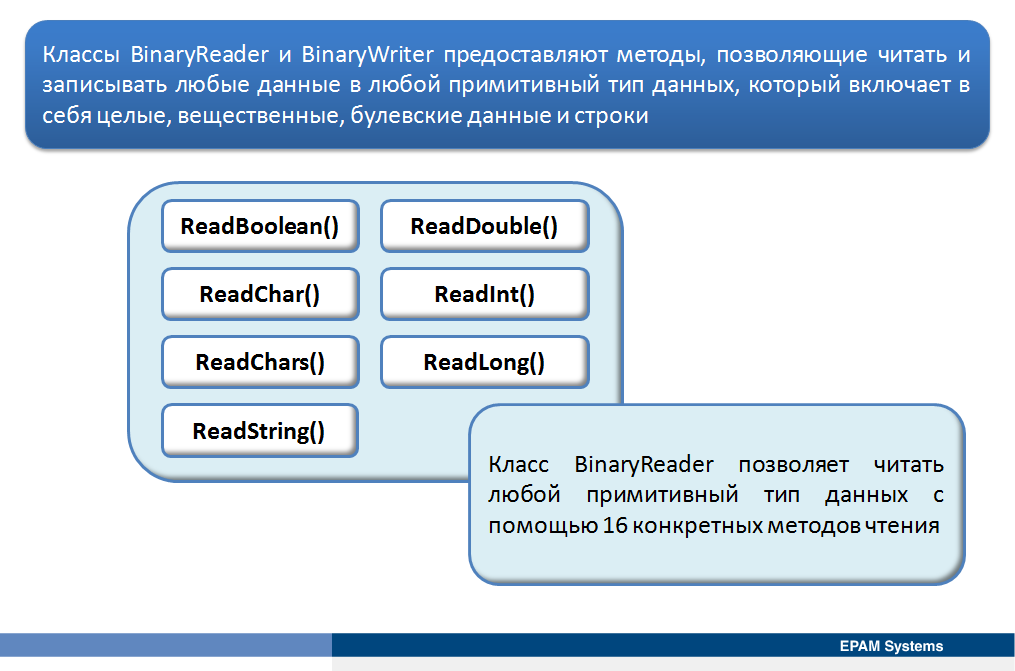
destFile.Close();

Когда закончено использование объектов StreamReader и StreamWriter, необходимо вызвать метод Close, чтобы флешировать поток и освободить все ресурсы, связанные с потоком. Необходимо также закрыть объект FileStream, предоставляющий данные для объектов StreamReader и StreamWriter.

<http://go.microsoft.com/fwlink/?LinkId=192921>

<http://go.microsoft.com/fwlink/?LinkId=192922>

## Чтение и запись примитивных типов данных



Использование классов BinaryReader и BinaryWriter не ограничено использованием неструктурированных массивов. Эти классы также предоставляют методы, позволяющие читать и записывать любые данные в любой примитивный тип данных, который включает в себя целые, вещественные, булевские данные и строки.

Потоковая модель, которую реализует. NET Framework, также поддерживает концепцию потоков непримитивных типов, таких как классы и структуры. Эти типы должны быть сериализованы, и можно использовать форматтеры, такие как объект BinaryFormatter с объектом FileStream, указывающим, как читать и писать данные.

**Чтение примитивных типов данных.** Класс BinaryReader позволяет читать любой примитивный тип данных с помощью 16 конкретных методов чтения. В следующей таблице приведены некоторые из методов чтения, предоставляемых классом BinaryReader.

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| **ReadBoolean** | Читает из потока значение **true/false**. |
| **ReadChar** | Читает из потока один символ. |
| **ReadChars** | Читает из потока набор символов. При использовании этого метода, нужно указать число символов, которые должен вернуть метод. |
| **ReadDouble** | Читает из потока значение типа double. |
| **ReadInt** | Читает из потока значение типа **int**. |
| **ReadLong** | Читает из потока значение типа **long**. |
| **ReadString** | Читает из потока значение типа **string**. |

Каждый из методов чтения предназначен для работы с определенным типом данных. Метод считывает необходимое количество байт для данного типа[[3]](#footnote-3), а затем перемещает указатель в потоке на следующий блок байтов. В следующем примере кода показано, как прочитать файл, содержащий множество примитивных типов.

// Source file path.

string sourceFilePath = @"C:\Users\Student\Documents\PrimitiveDataTypeFile.txt";

// Create a FileStream object so that you can interact with the file system.

FileStream sourceFile = new FileStream(

sourceFilePath, // Pass in the source file path.

FileMode.Open, // Open an existing file.

FileAccess.Read);// Read an existing file.

// Create a BinaryWriter object passing in the FileStream object.

BinaryReader reader = new BinaryReader(sourceFile);

bool boolValue = reader.ReadBoolean();

byte byteValue = reader.ReadByte();

byte[] byteArrayValue = reader.ReadBytes(4);

char charValue = reader.ReadChar();

char[] charArrayValue = reader.ReadChars(4);

decimal decimalValue = reader.ReadDecimal();

double doubleValue = reader.ReadDouble();

float floatValue = reader.ReadSingle();

int intValue = reader.ReadInt32();

long longValue = reader.ReadInt64();

sbyte sbyteValue = reader.ReadSByte();

short shortValue = reader.ReadInt16();

string stringValue = reader.ReadString();

uint unintValue = reader.ReadUInt32();

ulong ulongValue = reader.ReadUInt64();

ushort ushortValue = reader.ReadUInt16();

// Close the streams to release any file handles.

reader.Close();

sourceFile.Close();

**Запись примитивных типов данных.** Класс BinaryWriter позволяет записать любой примитивный тип данных методом записи, который обеспечивает несколько перегрузок. В следующем примере кода показано использование класса BinaryWriter для записи в текстовый файл различных примитивных типов данных.

string destinationFilePath = @"C:\Users\Student\Documents\PrimitiveDataTypeFile.txt";

// Create a FileStream object so that you can interact with the file

// system.

FileStream destFile = new FileStream(

destinationFilePath, // Pass in the destination path.

FileMode.Create, // Always create new file.

FileAccess.Write); // Only perform writing.

// Create a BinaryWriter object passing in the FileStream object.

BinaryWriter writer = new BinaryWriter(destFile);

bool boolValue = true;

writer.Write(boolValue);

byte byteValue = 1;

writer.Write(byteValue);

byte[] byteArrayValue = { 1, 4, 6, 8 };

writer.Write(byteArrayValue);

char charValue = 'a';

writer.Write(charValue);

char[] charArrayValue = { 'a', 'b', 'c', 'd' };

writer.Write(charArrayValue);

decimal decimalValue = 1.00m;

writer.Write(decimalValue);

double doubleValue = 2.5;

writer.Write(doubleValue);

float floatValue = 4.5f;

writer.Write(floatValue);

int intValue = 999999999;

writer.Write(intValue);

long longValue = 999999999999999999;

writer.Write(longValue);

sbyte sbyteValue = 99;

writer.Write(sbyteValue);

short shortValue = 9999;

writer.Write(shortValue);

string stringValue = "MyString";

writer.Write(stringValue);

uint unintValue = 999999999;

writer.Write(unintValue);

ulong ulongValue = 999999999999999999;

writer.Write(ulongValue);

ushort ushortValue = 9999;

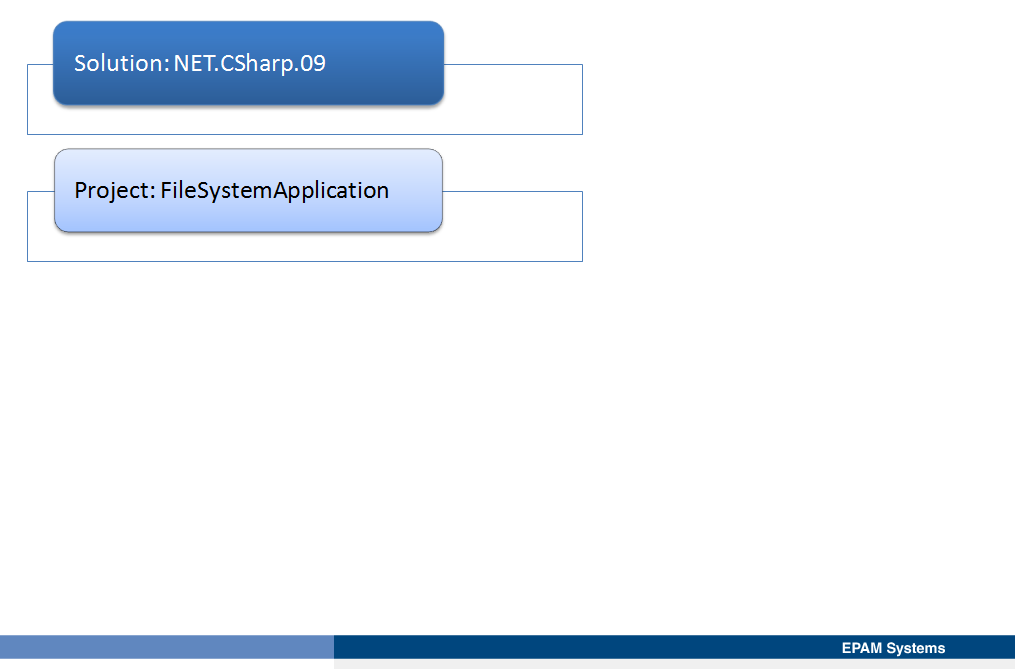
writer.Write(ushortValue);

// Close both streams to flush the data to the file.

writer.Close();

destFile.Close();

## Демонстрация: Чтение и запись в файлы



1. В пространстве имен System.Windows.Forms также можно найти классы OpenFileDialog и SaveFileDialog. До появления Windows Presentation Foundation (WPF) Windows Forms было основной технологией для реализации клиентских Windows-приложений в .NET Framework, отсюда включение пространства имен System.Windows.Forms. [↑](#footnote-ref-1)
2. Темы данного урока сосредоточены на чтении и записи данных в файлы файловой системы, поэтому используется класс FileStream. При этом будут рассмотрены чтение и запись различных данных, поэтому также будут рассмотрены классы, такие как BinaryReader, BinaryWriter, StreamReader и StreamWriter [↑](#footnote-ref-2)
3. При считывании массива необходимо указать количество элементов в нем. [↑](#footnote-ref-3)