

Файловий ввід- вивід та серіалізація об'єктів

ЛЕКЦІЯ 6

План лекції

Файловий ввід-вивід

Потоки вводу-виводу

Серіалізація об'єктів

Робота з файлами та документами за допомогою WPF-додатків (у планах на 2020 рік)

Файловий ввід- вивід

ПИТАННЯ 1

Исследование пространства имен System.IO

Пространство имен System.IO в .NET — это область библиотек базовых классов, посвященная службам файлового ввода-вывода, а также ввода-вывода из памяти.

- В System.IO определен набор классов, интерфейсов, перечислений, структур и делегатов, большинство из которых находятся в mscorlib.dll.
- В дополнение в сборке System.dll определены дополнительные члены пространства имен System.IO.
- Обратите внимание, что во всех проектах Visual Studio автоматически устанавливаются ссылки на обе сборки.

Многие типы из пространства имен System.IO сосредоточены на программной манипуляции физическими каталогами и файлами.

- Дополнительные типы предоставляют поддержку чтения и записи данных в строковые буферы, а также области памяти.

Ключевые члены пространства имен System.IO

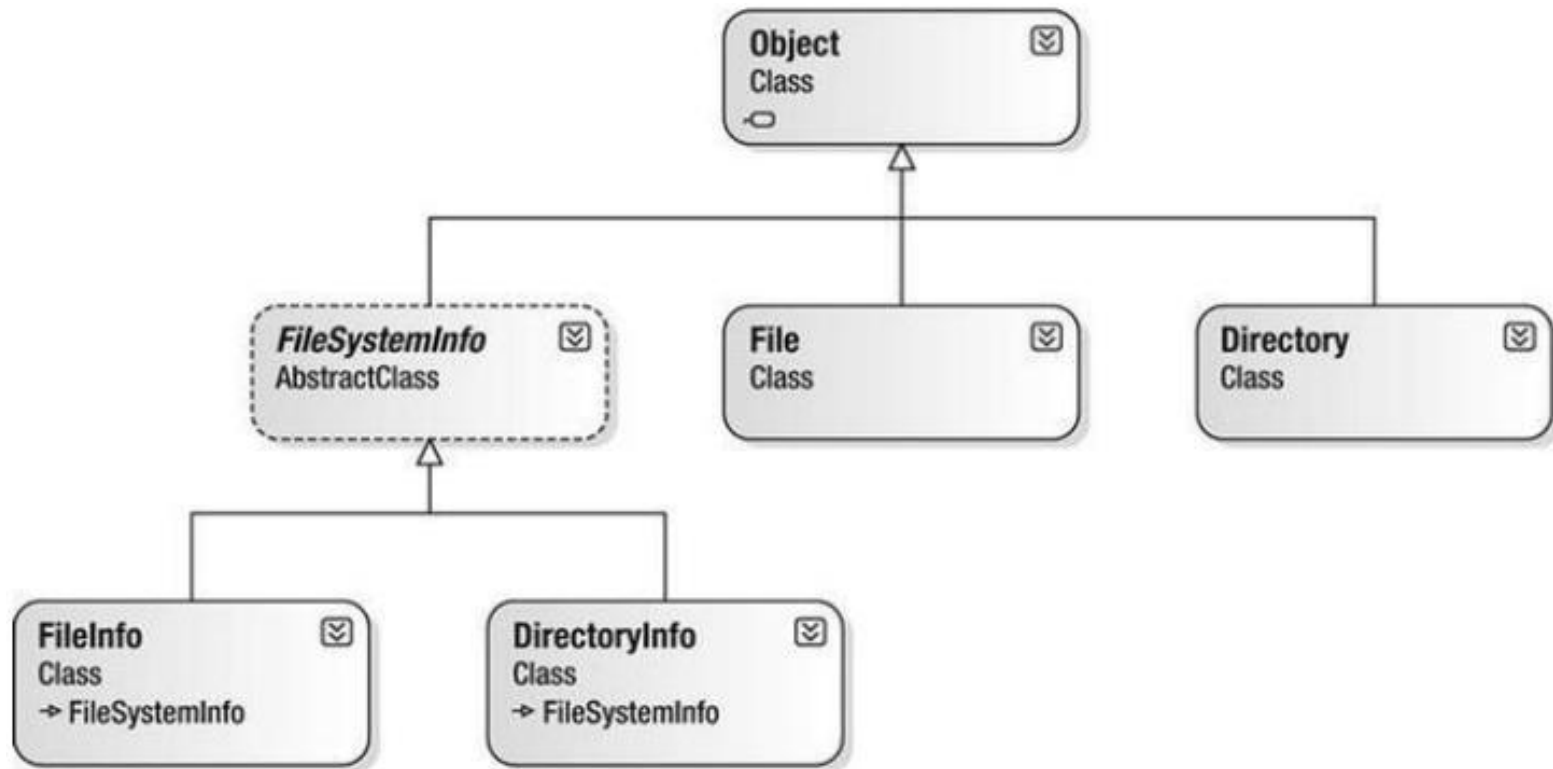
Неабстрактные классы ввода-вывода	Описание
BinaryReader BinaryWriter	Эти классы позволяют сохранять и извлекать элементарные типы данных (целочисленные, булевские, строковые и т.п.) в двоичном виде
BufferedStream	Этот класс предоставляет временное хранилище для потока байтов, который может быть зафиксирован в постоянном хранилище в более позднее время
Directory DirectoryInfo	Эти классы используются для манипуляций структурой каталогов машины. Тип <code>Directory</code> открывает функциональность с использованием <i>статических членов</i> . Тип <code>DirectoryInfo</code> обеспечивает аналогичную функциональность через действительную <i>объектную ссылку</i>
DriveInfo	Этот класс предоставляет детальную информацию относительно дисковых устройств, используемых на заданной машине

В дополнение к этим конкретным типам классов в System.IO определено несколько перечислений, а также набор абстрактных классов (например, `Stream`, `TextReader` и `TextWriter`), которые определяют разделяемый полиморфный интерфейс для всех наследников.

File FileInfo	Эти классы служат для манипулирования множеством файлов на машине. Тип <i>File</i> открывает функциональность через <i>статические члены</i> . Тип <i>FileInfo</i> обеспечивает аналогичную функциональность через <i>действительную объектную ссылку</i>
FileStream	Этот класс предоставляет произвольный доступ к файлу (например, возможности поиска) с данными, представленными в виде потока байт
FileSystemWatcher	Этот класс позволяет отслеживать модификации внешних файлов в определенном каталоге
MemoryStream	Этот класс обеспечивает произвольный доступ к данным, хранящимся в памяти, а не в физическом файле
Path	Этот класс выполняет операции над типами <i>System.String</i> , содержащими информацию о пути к файлу или каталогу в независимой от платформы манере
StreamWriter StreamReader	Эти классы используются для хранения (и извлечения) текстовой информации из файла. Они не поддерживают произвольный доступ к файлу
StringWriter StringReader	Подобно <i>StreamWriter/StreamReader</i> , эти классы также работают с текстовой информацией. Однако лежащим в основе хранилищем является строковый буфер, а не физический файл

Классы Directory (DirectoryInfo) и File (FileInfo)

В System.IO предоставляются 4 класса, которые позволяют манипулировать индивидуальными файлами, а также взаимодействовать со структурой каталогов машины.



Опис архітектури

Первые два класса, `Directory` и `File`, предлагают операции создания, удаления, копирования и перемещения с использованием различных статических членов.

- Родственные им классы `FileInfo` и `DirectoryInfo` обеспечивают похожую функциональность в виде методов уровня экземпляра (и потому должны размещаться в памяти с помощью ключевого слова `new`).

Классы `Directory` и `File` напрямую расширяют `System.Object`, в то время как `DirectoryInfo` и `FileInfo` являются производными от абстрактного класса `FileSystemInfo`.

Обычно `FileInfo` и `DirectoryInfo` представляют собой лучший выбор для получения полных деталей о файле или каталоге (например, время создания, возможности чтения/записи и т.п.), поскольку их члены возвращают строго типизированные объекты.

- В отличие от этого, члены классов `Directory` и `File`, как правило, возвращают простые строковые значения, а не строго типизированные объекты.
- Тем не менее, это лишь руководящий принцип; во многих случаях одну и ту же работу можно делать, используя `File/FileInfo` или `Directory/DirectoryInfo`.

Абстрактный базовый класс FileSystemInfo

Классы DirectoryInfo и FileInfo наследуют значительную часть своего поведения от абстрактного базового класса FileSystemInfo. По большей части члены класса FileSystemInfo используются для выяснения общих характеристик (таких как время создания, различные атрибуты и т.д.) определенного файла или каталога.

Свойство	Описание
Attributes	Получает или устанавливает ассоциированные с текущим файлом атрибуты, которые представлены перечислением FileAttributes (например, доступный только для чтения, зашифрованный, скрытый или сжатый)
CreationTime	Получает или устанавливает время создания текущего файла или каталога
Exists	Может использоваться для определения, существует ли данный файл или каталог
Extension	Извлекает расширение файла
FullName	Получает полный путь к файлу или каталогу
LastAccessTime	Получает или устанавливает время последнего доступа к текущему файлу или каталогу
LastWriteTime	Получает или устанавливает время последней записи в текущий файл или каталог
Name	Получает имя текущего файла или каталога

Работа с типом DirectoryInfo

содержит набор членов, используемых для создания, перемещения, удаления и перечисления каталогов и подкаталогов.

- В дополнение к функциональности, предоставленной его базовым классом (FileSystemInfo), класс DirectoryInfo предлагает ключевые члены

Член	Описание
Create() CreateSubdirectory()	Создает каталог (или набор подкаталогов) по заданному путевому имени
Delete()	Удаляет каталог и все его содержимое
GetDirectories()	Возвращает массив объектов DirectoryInfo, представляющих все подкаталоги в текущем каталоге
GetFiles()	Извлекает массив объектов FileInfo, представляющий набор файлов в заданном каталоге
MoveTo()	Перемещает каталог со всем содержимым по новому пути
Parent	Извлекает родительский каталог данного каталога
Root	Получает корневую часть пути

Работа с типом `DirectoryInfo` начинается с указания определенного пути в качестве параметра конструктора.

Если требуется получить доступ к текущему рабочему каталогу (т.е. каталогу выполняющегося приложения), применяйте обозначение в виде точки (.).

- `DirectoryInfo dir1 = new DirectoryInfo(".");`
- `DirectoryInfo dir2 = new DirectoryInfo(@"C:\Windows");`

Во втором примере предполагается, что переданный конструктору путь (`C:\Windows`) уже существует на физической машине.

- Однако при попытке взаимодействия с несуществующим каталогом генерируется исключение `System.IO.DirectoryNotFoundException`.
- Таким образом, чтобы указать каталог, который пока еще не создан, сначала придется вызвать метод `Create()`:

```
DirectoryInfo dir3 = new DirectoryInfo(@"C:\MyCode\Testing");  
dir3.Create();
```

Вывод ряда интересных статистических данных

```
class Program {
    static void Main(string[] args) {
        ShowWindowsDirectoryInfo();
        Console.ReadLine();
    }

    static void ShowWindowsDirectoryInfo() {
        // Dump directory information.
        DirectoryInfo dir = new DirectoryInfo(@"C:\Windows");
        Console.WriteLine("***** Directory Info *****");
        Console.WriteLine("Повна назва: {0}", dir.FullName);
        Console.WriteLine("Назва: {0}", dir.Name);
        Console.WriteLine("Батьківський каталог: {0}", dir.Parent);
        Console.WriteLine("Створено: {0}", dir.CreationTime);
        Console.WriteLine("Атрибути: {0}", dir.Attributes);
        Console.WriteLine("Корінь: {0}", dir.Root);
        Console.WriteLine("*****\n");
    }
}
```

file:///C:/Users/spuasson/Source/Repos

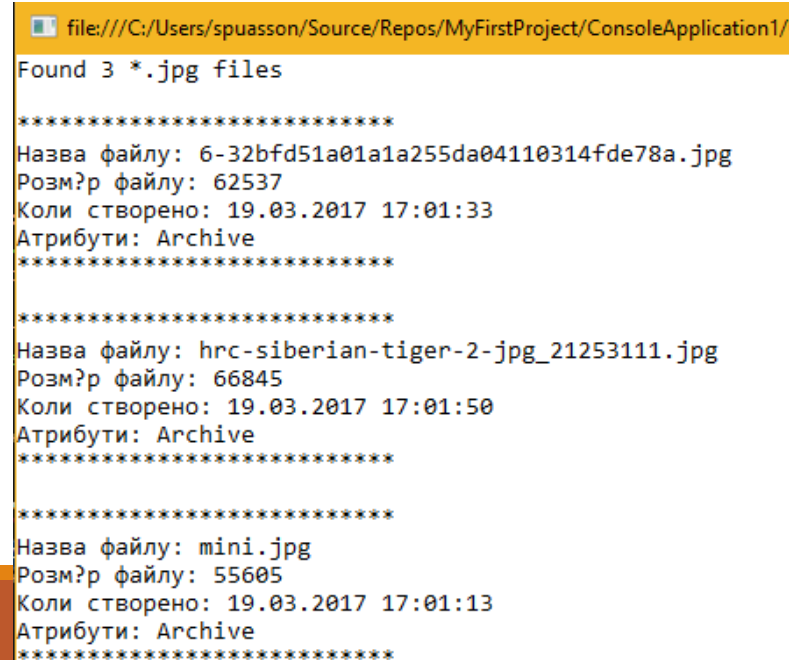
```
***** Directory Info *****
Повна назва: C:\Windows
Назва: Windows
Батьківський каталог:
Створено: 16.07.2016 9:04:24
Атрибути: Directory
Корінь: C:\
*****
```

Перечисление файлов с помощью типа DirectoryInfo

Метод GetFiles() возвращает массив объектов типа FileInfo, каждый из которых отражает детальную информацию о конкретном файле

```
DirectoryInfo dir = new DirectoryInfo(@"C:\Wallpaper");
// Отримати всі файли з розширенням *.jpg.
FileInfo[] imageFiles = dir.GetFiles("*.jpg", SearchOption.AllDirectories);
// Скільки було знайдено?
Console.WriteLine("Found {0} *.jpg files\n", imageFiles.Length);
// Тепер виведемо інформацію про кожний файл.
foreach (FileInfo f in imageFiles)
{
    Console.WriteLine("*****");
    Console.WriteLine("Назва файлу: {0}", f.Name);
    Console.WriteLine("Розмір файлу: {0}", f.Length);
    Console.WriteLine("Коли створено: {0}", f.CreationTime);
    Console.WriteLine("Атрибути: {0}", f.Attributes);
    Console.WriteLine("*****\n");
}
```

Обратите внимание на указание в вызове GetFiles() опции поиска; SearchOption.AllDirectories обеспечивает просмотр всех подкаталогов корня.



```
file:///C:/Users/spuasson/Source/Repos/MyFirstProject/ConsoleApplication1/
Found 3 *.jpg files

*****
Назва файлу: 6-32bfd51a01a1a255da04110314fde78a.jpg
Розм?р файлу: 62537
Коли створено: 19.03.2017 17:01:33
Атрибути: Archive
*****

*****
Назва файлу: hrc-siberian-tiger-2-jpg_21253111.jpg
Розм?р файлу: 66845
Коли створено: 19.03.2017 17:01:50
Атрибути: Archive
*****

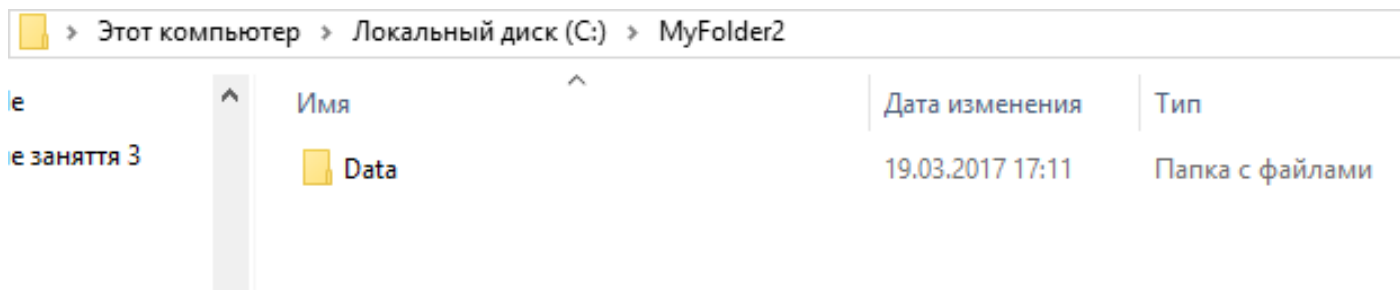
*****
Назва файлу: mini.jpg
Розм?р файлу: 55605
Коли створено: 19.03.2017 17:01:13
Атрибути: Archive
*****
```

Создание подкаталогов с помощью типа DirectoryInfo

Для программного расширения структуры каталогов служит метод `DirectoryInfo.CreateSubdirectory()`.

- С его помощью можно создавать как одиночный подкаталог, так и множество вложенных подкаталогов за один вызов.

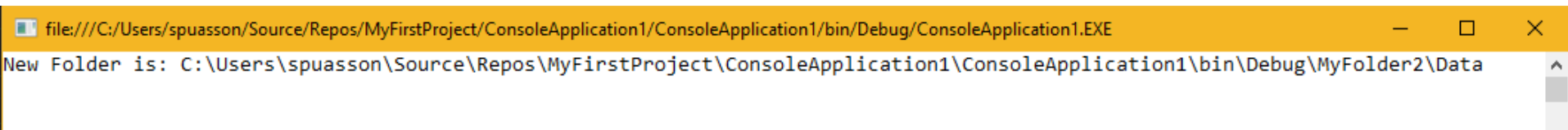
```
static void ModifyAppDirectory()
{
    DirectoryInfo dir = new DirectoryInfo(@"C:\");
    // Створення \MyFolder off application directory.
    dir.CreateSubdirectory("MyFolder");
    // Створення \MyFolder2\Data off application directory.
    dir.CreateSubdirectory(@"MyFolder2\Data");
}
```



Хотя получать возвращаемое значение метода `CreateSubdirectory()` не обязательно, имейте в виду, что в случае его успешного выполнения возвращается объект `DirectoryInfo`, представляющий вновь созданный элемент.

- Ниже показана модификация предыдущего метода.
- Обратите внимание на применение "." в конструкторе `DirectoryInfo`; это обеспечивает доступ к месту установки приложения.

```
static void ModifyAppDirectory() {  
    DirectoryInfo dir = new DirectoryInfo(".");  
    // Створення \MyFolder off initial directory.  
    dir.CreateSubdirectory("MyFolder");  
    // Перехоплення об'єкту DirectoryInfo, що повертається  
    DirectoryInfo myDataFolder = dir.CreateSubdirectory(@"MyFolder2\Data");  
    // Друкує шлях ..\MyFolder2\Data.  
    Console.WriteLine("New Folder is: {0}", myDataFolder);  
}
```

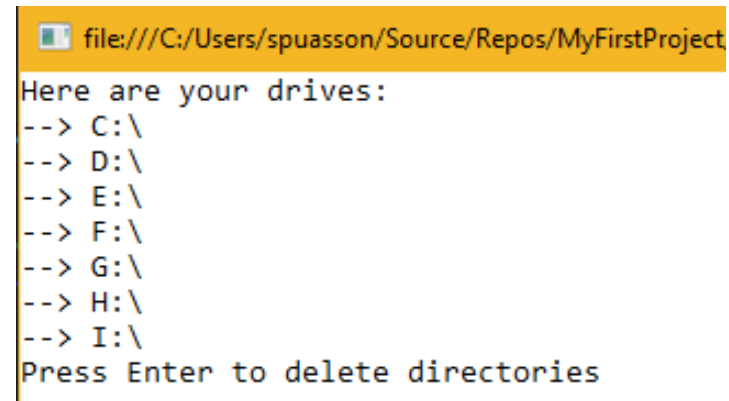


Работа с типом Directory

По большей части статические члены Directory повторяют функциональность, предоставляемую членами уровня экземпляра, которые определены в DirectoryInfo.

- Однако вспомните, что члены Directory обычно возвращают строковые данные вместо строго типизированных объектов FileInfo/DirectoryInfo.

```
static void FunWithDirectoryType() {  
    // Вивести список усіх дискових пристроїв комп'ютера.  
    string[] drives = Directory.GetLogicalDrives();  
    Console.WriteLine("Here are your drives:");  
    foreach (string s in drives)  
        Console.WriteLine("--> {0} ", s);  
    // Видалити те, що було створено.  
    Console.WriteLine("Press Enter to delete directories");  
    Console.ReadLine();  
    try {  
        Directory.Delete(@"C:\MyFolder");  
        // Другий параметр задає,  
        // чи хочете видалити subdirectories.  
        Directory.Delete(@"C:\MyFolder2", true);  
    } catch (IOException e) {  
        Console.WriteLine(e.Message);  
    }  
}
```



```
file:///C:/Users/spuasson/Source/Repos/MyFirstProject  
Here are your drives:  
--> C:\  
--> D:\  
--> E:\  
--> F:\  
--> G:\  
--> H:\  
--> I:\  
Press Enter to delete directories
```


Работа с типом DriveInfo

Подобно `Directory.GetLogicalDrives()`, статический метод `DriveInfo.GetDrives()` позволяет получить имена устройств на машине.

- В отличие от `Directory.GetLogicalDrives()`, класс `DriveInfo` предоставляет множество дополнительных деталей (например, тип устройства, доступное свободное пространство и метка тома).

```
Console.WriteLine("***** Fun with DriveInfo *****\n");
// Get info regarding all drives.
DriveInfo[] myDrives = DriveInfo.GetDrives();
// Now print drive stats.
foreach (DriveInfo d in myDrives) {
    Console.WriteLine("Name: {0}", d.Name);
    Console.WriteLine("Type: {0}", d.DriveType);
    // Check to see whether the drive is mounted.
    if (d.IsReady) {
        Console.WriteLine("Free space: {0}", d.TotalFreeSpace);
        Console.WriteLine("Format: {0}", d.DriveFormat);
        Console.WriteLine("Label: {0}", d.VolumeLabel);
    }
    Console.WriteLine();
}
```

```
file:///C:/Users/spuasson/Source/Repos/MyFirst
***** Fun with DriveInfo *****

Name: C:\
Type: Fixed
Free space: 4381319168
Format: NTFS
Label:

Name: D:\
Type: Fixed
Free space: 4193439744
Format: NTFS
Label: Новый том

Name: E:\
Type: Fixed
Free space: 10515169280
Format: NTFS
Label: Learning

Name: F:\
Type: Fixed
Free space: 965853184
Format: NTFS
Label:

Name: G:\
Type: Fixed
Free space: 3739521024
Format: NTFS
Label: PhD

Name: H:\
Type: Fixed
Free space: 4997697536
Format: NTFS
Label: Библиотека
```

Работа с классом FileInfo

класс `FileInfo` позволяет получать подробные сведения о существующих файлах на жестком диске (например, время создания, размер и атрибуты) и помогает создавать, копировать, перемещать и удалять файлы.

- В дополнение к набору функциональности, унаследованной от `FileSystemInfo`, класс `FileInfo` имеет ряд уникальных членов

Член	Описание
<code>AppendText ()</code>	Создает объект <code>StreamWriter</code> (описанный ниже) и добавляет текст в файл
<code>CopyTo ()</code>	Копирует существующий файл в новый файл
<code>Create ()</code>	Создает новый файл и возвращает объект <code>FileStream</code> (описанный ниже) для взаимодействия с вновь созданным файлом
<code>CreateText ()</code>	Создает объект <code>StreamWriter</code> , записывающий новый текстовый файл

Основные члены FileInfo

Delete()	Удаляет файл, к которому привязан экземпляр FileInfo
Directory	Получает экземпляр родительского каталога
DirectoryName	Получает полный путь к родительскому каталогу
Length	Получает размер текущего файла или каталога
MoveTo()	Перемещает указанный файл в новое местоположение, предоставляя возможность указания нового имени для файла
Name	Получает имя файла
Open()	Открывает файл с различными привилегиями чтения/записи и совместного доступа
OpenRead()	Создает доступный только для чтения объект FileStream
OpenText()	Создает объект StreamReader (описанный ниже) и читает из существующего текстового файла
OpenWrite()	Создает доступный только для записи объект FileStream

Обратите внимание, что большинство методов класса FileInfo возвращают специфический объект ввода-вывода (такой как FileStream и StreamWriter), который позволяет начать чтение и запись данных в ассоциированный файл в разнообразных форматах.

Метод FileInfo.Create()

Один из способов создания дескриптора файла предусматривает применение метода FileInfo.Create().

- Метод FileInfo.Create () возвращает тип FileStream, который предоставляет синхронную и асинхронную операции записи/чтения лежащего в его основе файла.
- Имейте в виду, что объект FileStream, возвращаемый FileInfo.Create (), открывает полный доступ по чтению и записи всем пользователям.

```
// Створює новий файл на диску C.  
FileInfo f = new  
FileInfo(@"C:\Test.dat");  
FileStream fs = f.Create();  
// Використовує FileStream...  
// Закриває file stream.  
fs.Close();
```

Метод FileInfo.Create()

Также обратите внимание, что после окончания работы с текущим объектом FileStream необходимо обеспечить закрытие его дескриптора для освобождения внутренних неуправляемых ресурсов потока.

- Учитывая, что FileStream реализует интерфейс IDisposable, можно применить контекст using и позволить компилятору сгенерировать логику завершения

```
FileInfo f = new FileInfo(@"C:\Test.dat");
using (FileStream fs = f.Create())
{
    // Використовуємо об'єкт FileStream...
}
```

Метод FileInfo.Open()

С помощью метода FileInfo.Open() можно открывать существующие файлы, а также создавать новые файлы с гораздо более высокой точностью, чем FileInfo.Create(), учитывая, что Open() обычно принимает несколько параметров для описания общей структуры файла, с которым будет производиться работа.

- В результате вызова Open() получается возвращенный им объект FileStream.
- Эта версия перегруженного метода Open() требует трех параметров.
- Первый параметр указывает общий тип запроса ввода-вывода (например, создать новый файл, открыть существующий файл и дописать в файл), указываемый в виде перечисления FileMode

```
// Make a new file via FileInfo.Open().
FileInfo f2 = new FileInfo(@"C:\Test2.dat");
using (FileStream fs2 = f2.Open(FileMode.OpenOrCreate,
    FileAccess.ReadWrite, FileShare.None))
{
    // Use the FileStream object...
}
```

```
public enum FileMode
{
    CreateNew,
    Create,
    Open,
    OpenOrCreate,
    Truncate,
    Append
}
```

Члены перечисления FileMode

Член	Описание
CreateNew	Информирует операционную систему о создании нового файла. Если файл уже существует, генерируется исключение <code>IOException</code>
Create	Информирует операционную систему о создании нового файла. Если файл уже существует, он будет перезаписан
Open	Открывает существующий файл. Если файл не существует, генерируется исключение <code>FileNotFoundException</code>
OpenOrCreate	Открывает файл, если он существует; в противном случае создает новый
Truncate	Открывает файл и усекает его до нулевой длины
Append	Открывает файл, переходит в его конец и начинает операции записи (этот флаг может использоваться только с потоками, предназначенными лишь для записи). Если файл не существует, то создается новый

Второй параметр метода `Open ()` — значение перечисления `FileAccess` — используется для определения поведения чтения/записи лежащего в основе потока:

```
public enum FileAccess
{
    Read,
    Write,
    ReadWrite
}
```

И, наконец, третий параметр метода `Open ()` — `FileShare` — указывает, как файл может совместно использоваться другими файловыми дескрипторами:

```
public enum FileShare
{
    Delete,
    Inheritable,
    None,
    Read,
    ReadWrite,
    Write
}
```


Методы FileOpen.OpenRead() и FileInfo.OpenWrite()

Хотя метод FileOpen.Open() позволяет получить дескриптор файла довольно гибким способом, в классе FileInfo также предусмотрены для этого члены OpenRead () и OpenWrite().

- Как и можно было бы ожидать, эти методы возвращают объект FileStream, соответствующим образом сконфигурированный только для чтения или только для записи, без необходимости в указании различных значений перечисления.

```
// Get a FileStream object with read-only permissions.
FileInfo f3 = new FileInfo(@"C:\Test3.dat");
using (FileStream readOnlyStream = f3.OpenRead()) {
    // Use the FileStream object...
}
// Now get a FileStream object with write - only permissions.
FileInfo f4 = new FileInfo(@"C:\Test4.dat");
using (FileStream writeOnlyStream = f4.OpenWrite())
{
    // Use the FileStream object...
}
```

Подобно
FileInfo.Create() и
FileInfo.Open(),
методы OpenRead ()
и OpenWrite ()
возвращают объект
FileStream

Метод FileInfo.OpenText()

Еще один член типа FileInfo, связанный с открытием файлов — OpenText (). В отличие от Create (), Open (), OpenRead () и OpenWrite (), метод OpenText () возвращает экземпляр типа StreamReader, а не FileStream.

- Исходя из того, что на диске C: уже есть файл по имени boot.ini, получить доступ к его содержимому можно следующим образом:

```
// Get a StreamReader object.  
FileInfo f5 = new FileInfo(@"C:\boot.ini");  
using (StreamReader sreader = f5.OpenText())  
{  
    // Use the StreamReader object...  
}
```

Как вскоре будет показано, тип StreamReader предоставляет способ чтения символьных данных из лежащего в основе файла.

Методы FileInfo.CreateText() и FileInfo.AppendText ()

Оба они возвращают объект StreamWriter

```
FileInfo f6 = new FileInfo(@"C:\Test6.txt");
using (StreamWriter swriter = f6.CreateText())
{
    // Use the StreamWriter object...
}
FileInfo f7 = new FileInfo(@"C:\FinalTest.txt");
using (StreamWriter swriterAppend = f7.AppendText())
{
    // Use the StreamWriter object...
}
```

Как и можно было ожидать, тип StreamWriter предлагает способ записи данных в лежащий в основе файл.

Работа с типом File

Тип File предоставляет функциональность, почти идентичную типу FileInfo, с помощью нескольких статических методов.

- Подобно FileInfo, тип File поддерживает методы AppendText(), Create(), CreateText(), Open(), OpenRead(), OpenWrite() и OpenText().
- Фактически во многих случаях типы File и FileInfo могут использоваться взаимозаменяемым образом.

```
// Получить объект FileStream через File.Create().
using(FileStream fs = File.Create("C:\Test.dat")) {}

// Получить объект FileStream через File.Open().
using(FileStream fs2 = File.Open("C:\Test2.dat", FileMode.OpenOrCreate,
FileAccess.ReadWrite, FileShare.None)) {}

// Получить объект FileStream с правами только для чтения.
using(FileStream readOnlyStream = File.OpenRead("Test3.dat11")) {}

// Получить объект FileStream с правами только для записи.
using(FileStream writeOnlyStream = File.OpenWrite("Test4.dat")) {}

// Получить объект StreamReader.
using(StreamReader sreader = File.OpenText("C:\boot.ini11")) {}

// Получить несколько объектов StreamWriter.
using(StreamWriter swriter = File.CreateText("C:\Test6.txt")) {}
using(StreamWriter swriterAppend = File.AppendText("C:\FinalTest.txt")) {}
```

Дополнительные члены File.

Методы

Метод	Описание
<code>ReadAllBytes()</code>	Открывает указанный файл, возвращает двоичные данные в виде массива байт и затем закрывает файл
<code>ReadAllLines()</code>	Открывает указанный файл, возвращает символьные данные в виде массива строк, затем закрывает файл
<code>ReadAllText()</code>	Открывает указанный файл, возвращает символьные данные в виде <code>System.String()</code> , затем закрывает файл
<code>WriteAllBytes()</code>	Открывает указанный файл, записывает в него массив байтов и закрывает файл
<code>WriteAllLines()</code>	Открывает указанный файл, записывает в него массив строк и закрывает файл
<code>WriteAllText()</code>	Открывает указанный файл, записывает в него данные из указанной строки и закрывает файл

Используя эти методы типа `File`, можно осуществлять чтение и запись пакетов данных с помощью всего нескольких строк кода. Еще лучше то, что эти методы автоматически закрывают лежащий в основе файловый дескриптор.

консольная программа сохраняет строковые данные в новом файле на диске C: (и читает их в память) с минимальными усилиями (предполагается, что было импортировано пространство имен

System.IO):

```
Console.WriteLine("***** Simple I/O with the File Type *****\n");
string[] myTasks = {"Fix bathroom sink", "Call Dave",
    "Call Mom and Dad", "Play Xbox One"};
// Записать все данные в файл на диске C: .
File.WriteAllLines(@"C:\tasks.txt", myTasks);
// Прочитать все данные и вывести на консоль.
foreach (string task in File.ReadAllLines(@"C:\tasks.txt"))
{
    Console.WriteLine("TODO: {0}", task);
}
Console.ReadLine();
```

когда необходимо быстро получить файловый дескриптор, тип File позволит сэкономить на объеме кодирования. Однако преимущество предварительного создания объекта FileInfo связано с возможностью исследования файла с использованием членов абстрактного базового класса FileSystemInfo.

Дякую за увагу!
