Двоичные файлы

Все классы определены в пространстве имен System.IO.

Перечисления, связанные с обработкой файлов

FileMode

Перечисление FileMode — *режим открытия* файла:

- CreateNew (1) — создать новый файл (если файл уже существует, то возбуждается исключение IOException);
- Create (2) — создать новый файл; если файл уже существует, то его содержимое очищается;
- Open (3) — открыть существующий файл (если файл не существует, то возбуждается исключение FileNotFoundException);
- OpenOrCreate (4) — открыть существующий файл; если файл не существует, то он создается;
- Truncate (5) — очистить содержимое *существующего* файла, после чего открыть его (если файл не существует, то возбуждается исключение FileNotFoundException);
- Append (6) — открыть существующий файл на запись и переместиться в его конец; если файл не существует, то он создается.

FileAccess

Перечисление FileAccess — *способ доступа* к файлу:

- Read (1) — доступ для чтения;
- Write (2) — доступ для записи;
- ReadWrite (3) — доступ для чтения и записи.

SeekOrigin

Перечисление SeekOrigin — позиция, от которой отсчитывается смещение файлового указателя в методе Seek:

- Begin (0) — смещение определяется относительно начала файла (допускаются только неотрицательные смещения);
- Current (1) — смещение определяется относительно текущей позиции файлового указателя;
- End (2) — смещение определяется относительно конца файла (допускаются только неположительные смещения).

Двоичный файловый поток: класс FileStream

Класс FileStream (*файловый поток*) обеспечивает базовые возможности для работы с файлами (открытие, определение и изменение размера файла, позиционирование файлового указателя, чтение/запись байтов и массивов байтов, закрытие).

Основные свойства

```
string Name { get; }
```

Полное имя файла, связанного с файловым потоком this.

```
long Length { get; }
long Position { get; set; }
```

Свойство Length возвращает размер открытого файла в *байтах*, свойство Position возвращает и позволяет изменить текущую позицию файлового указателя (свойства имеют тип long, поэтому позволяют хранить размер и позицию файлового указателя для файлов размера *9 миллионов терабайт*).

Если присвоить свойству Position значение, большее Length, то автоматического изменения размера файла не произойдет. Для увеличения размера файла необходимо произвести запись новых элементов в установленную позицию (при этом значения байтов, расположенных между старыми и новыми элементами, полагаются равными нулю). Для увеличения размера файла можно также использовать метод SetLength.

Создание: конструктор и методы класса File

```
FileStream(string name, FileMode mode[, FileAccess access]);
```

Создает объект типа FileStream, связывает данный объект с файлом, имеющим имя name, и открывает данный файл в режиме, указанном в параметре mode. Если параметр access указан, то он определяет способ доступа к данному файлу; в противном случае устанавливается доступ для чтения и записи (FileAc-

cess.ReadWrite). Совместный доступ к файлу из нескольких файловых потоков возможен только в случае, если для всех этих потоков установлен доступ *только для чтения*.

Если указано краткое имя файла, то файл ищется в *текущем каталоге*, т. е. в рабочем каталоге приложения (work directory).

Кроме использования конструктора, можно также создать объект типа FileStream с помощью методов класса File.

```
static FileStream Create(string name);
```

Создает файл с именем name (или очищает файл, если он уже существует) и открывает его на чтение и запись.

```
static FileStream OpenRead(string name);
```

Открывает *существующий* файл с именем name на чтение.

```
static FileStream OpenWrite(string name);
```

Открывает файл с именем name на запись; если файл не существует, то он создается.

Главным преимуществом методов Create, OpenRead и OpenWrite класса File является более краткая форма их вызова.

Методы

```
long Seek(long offset, SeekOrigin origin);
```

Изменяет текущую позицию файлового указателя для файлового потока this и возвращает его новую позицию; offset определяет смещение указателя, origin — позицию в файле, относительно которой отсчитывается смещение. Вместо вызова этого метода достаточно изменить свойство Position:

`f.Seek(4, SeekOrigin.Begin)` равносильно `f.Position = 4`

`f.Seek(4, SeekOrigin.Current)` — `f.Position += 4`

`f.Seek(-4, SeekOrigin.End)` — `f.Position = f.Length - 4`

(предполагается, что файл имеет размер не менее 4 байт).

```
void SetLength(long value);
```

Изменяет размер файла (в байтах), полагая его равным значению value. Параметр value должен быть неотрицательным. Можно как уменьшать размер файла (при этом удаляются последние байты), так и увеличивать его размер (при этом в конец файла добавляются новые байты с нулевыми значениями).

```
int ReadByte();
```

Считывает значение байта из текущей позиции файла, перемещает файловый указатель к следующему байту (т. е. увеличивает значение свойства Position на 1) и возвращает значение прочитанного байта, преобразованное к типу int.

Если предпринимается попытка прочесть байт за концом файла, то метод возвращает –1. Если файл открыт и доступен для чтения, то выполнение данного метода никогда не приведет к возбуждению исключения.

```
int Read(byte[] array, int start, int count);
```

Считывает count или менее байтов из файлового потока this (начиная с байта, на который указывает файловый указатель), последовательно записывает их в элементы массива байтов array, начиная с элемента с индексом start, и возвращает количество фактически считанных байтов. Возвращаемое значение будет равно count, если успешно считаны все требуемые байты. В противном случае возвращаемое значение будет меньше параметра count. После выполнения метода файловый указатель перемещается вперед на количество фактически прочитанных байтов.

```
void WriteByte(byte value);
```

Записывает в текущую позицию файлового потока this один байт, равный value, и перемещает файловый указатель к следующему байту.

```
void Write(byte[] array, int start, int count);
```

Записывает count байтов из массива байтов array, начиная с элемента с индексом start, в файловый поток this, начиная с байта, на который указывает файловый указатель. После выполнения метода файловый указатель перемещается вперед на count байтов.

```
void Flush();
```

Записывает в файл данные, содержащиеся в *файловом буфере*, после чего очищает файловый буфер. Метод автоматически вызывается при закрытии файла методом Close.

```
void Close();
```

Закрывает файл, связанный с файловым потоком this, и освобождает неуправляемые ресурсы, выделенные для работы с данным файлом.

Когда объект, связанный с файловым потоком, разрушается (то есть удаляется из памяти), для него автоматически вызывается метод Close. Несмотря на эту возможность, *следует всегда закрывать файловый поток сразу после завершения работы с ним, явным образом вызывая метод* Close.

Повторный вызов метода Close игнорируется. После закрытия файла можно обращаться к свойству Name.

Потоки-оболочки: BinaryReader и BinaryWriter

Рассмотренный в предыдущем пункте класс FileStream позволяет осуществлять ввод-вывод файловых данных только в виде *наборов байтов*. Для возможности чтения или записи более сложных структур данных необходимо использовать «надстройки» над стандартным файловым потоком: класс BinaryReader (*двоичный поток-оболочка для чтения*) или класс BinaryWriter (*двоичный поток-оболочка для записи*).

Конструкторы, общие свойства и методы

```
BinaryReader(Stream stream[, Encoding encoding]);
BinaryWriter(Stream stream[, Encoding encoding]);
```

Каждый из конструкторов создает соответствующий двоичный поток-оболочку, которая связывается с базовым потоком stream. При работе с файлами в качестве параметра stream указывается объект типа FileStream. Поток stream необязательно предварительно сохранять в отдельной переменной; допустимо создавать его «на лету», указывая в качестве первого параметра вызов конструктора класса FileStream или метод класса File. В дальнейшем доступ к базовому потоку можно получить, используя свойство BaseStream потоков-оболочек.

Параметр encoding определяет для класса BinaryReader *формат декодирования* символьных данных при их чтении из файла, а для класса BinaryWriter — *формат кодирования* символьных данных при их записи в файл. Если данный параметр не указан, то используется формат UTF-8.

```
Stream BaseStream { get; }
```

Свойство только для чтения, возвращающее базовый поток для потока-оболочки this. Приводить данное свойство (типа Stream) к типу FileStream следует только в случае, если требуется обратиться к свойству Name или методу SetLength класса FileStream, так как все прочие свойства и методы уже определены в классе Stream, являющемся предком всех классов-потоков. Доступ к базовому потоку с помощью свойства BaseStream возможен только при *открытом* потоке-оболочке.

```
void Close();
```

Закрывает базовый поток BaseStream, связанный с потоком-оболочкой this, и освобождает неуправляемые ресурсы, выделенные для работы с этими потоками. Повторное выполнение метода Close игнорируется, не возбуждая исключения.

Необходимо *обязательно* вызывать метод Close потока-оболочки, так как данный метод (в отличие от одноименного метода класса FileStream) *не вызывается автоматически при разрушении объекта типа* BinaryReader *или* BinaryWriter.

Метод Close потока-оболочки *автоматически* закрывает базовый поток, поэтому явно вызывать метод Close базового потока после закрытия потока-оболочки *не требуется*. Если к одному и тому же файлу подключены два потока-оболочки, нельзя закрывать один из них до завершения работы с другим.

Чтение данных с помощью объекта BinaryReader

В любом из указанных ниже методов считывание данных начинается с текущей позиции файла (то есть с позиции файлового указателя). После выполнения любой операции по считыванию данных файловый указатель перемещается вперед на количество прочитанных байтов.

При работе с символьными данными (типа char и string) следует учитывать, что в файле они хранятся в закодированном виде, поэтому для их правильного считывания необходимо при создании потока-оболочки BinaryReader указать тот же *формат кодирования*, который использовался при записи этих символьных данных в файл.

Для чтения каждого элементарного типа данных в классе BinaryReader предусмотрен особый метод.

```
bool ReadBoolean();
byte ReadByte();
```

```
int ReadInt32();
long ReadInt64();
double ReadDouble();
char ReadChar();
string ReadString();
```

Каждый из методов данной группы считывает из потока один элемент требуемого типа и возвращает его значение (за исключением метода ReadBoolean, который читает из потока один байт и возвращает false, если прочитанный байт равен 0, и true в противном случае).

При попытке прочесть данные за концом файла возбуждается исключение EndOfStreamException.

Метод ReadString вначале читает из потока информацию о длине строки (*в байтах*), а затем считывает указанное количество байтов и преобразует прочитанные байты в символы, учитывая использованный в файле формат кодирования. Информация о длине строки может занимать от 1 до 5 байт. Например, если символы строки занимают не более 127 байт, то длина строки кодируется в *одном* байте, причем значение этого байта равно количеству байтов (не символов!) текста.

Запись данных с помощью объекта BinaryWriter

Для записи данных в классе BinaryWriter предусмотрен единственный метод, который перегружен для различных типов записываемых данных.

В любом из указанных методов запись данных начинается с текущей позиции файла (то есть с позиции файлового указателя). После выполнения любой операции по записи данных файловый указатель перемещается вперед на количество записанных байтов; при этом возможно увеличение размера файла.

```
void Write(bool value);
void Write(числовой_тип value);
void Write(char value);
void Write(string value);
```

Каждый из методов данной группы записывает в поток значение параметра value соответствующего типа. Исключение составляет параметр value типа bool, вместо которого в файл записывается один байт со значением 0 (если параметр равен false) или 1 (если параметр равен true).

При записи строки в файл вначале записывается информация о длине строки (указывается длина *уже закодированной* строки *в байтах*), а затем — сами символы строки (символы кодируются с учетом формата кодирования, определенного для потока BinaryWriter).

Текстовые файлы. Работа с файловой системой

Все классы (кроме Console) определены в пространстве имен System.IO.

Текстовые файловые потоки: классы StreamReader и StreamWriter

Для работы с *текстовыми файлами* предусмотрены два класса: *текстовый поток-оболочка для чтения* StreamReader и *текстовый поток-оболочка для записи* StreamWriter.

Создание и закрытие текстовых потоков

В отличие от двоичных потоков-оболочек (BinaryReader и BinaryWriter) для создания текстовых потоков и связывания их с файлами достаточно указать имя name текстового файла:

```
StreamReader(string name[, Encoding encoding]);
StreamWriter(string name[, bool append[, Encoding encoding]]);
```

В случае потока для чтения StreamReader указанный файл должен существовать. Файл открывается на чтение, и файловый указатель устанавливается на начало файла.

В случае потока для записи StreamWriter файл может отсутствовать; в этом случае он автоматически создается. Файл открывается для записи. Если параметр append не указан или равен false, то содержимое существующего файла очищается; если указан параметр append, равный true, то файл открывается *для дополнения* (его содержимое сохраняется, а файловый указатель устанавливается на маркер конца файла). Если файл является пустым, то значение append может быть любым.

Если указан параметр encoding, то он определяет *формат кодирования* файловых данных; при отсутствии этого параметра используется формат UTF-8. Для установки формата кодирования, который соответствует ANSI-кодировке, используемой системой Windows по умолчанию, в качестве параметра encoding следует указать Encoding.Default.

Совместный доступ к файлу из нескольких текстовых потоков возможен только в случае, если все эти потоки являются потоками типа StreamReader.

```
void Close();
```

Закрывает текстовый поток this и освобождает связанные с ним неуправляемые ресурсы. Повторное выполнение метода Close игнорируется. Для текстовых потоков следует *всегда* вызывать метод Close после завершения работы с ними.

Чтение данных из текстового потока

В любом из описанных ниже методов чтения считывание данных начинается с текущей позиции файла. После выполнения любой операции чтения файловый указатель перемещается вперед на количество прочитанных символов.

```
int Read();
```

Считывает из потока один символ и возвращает его значение, преобразованное к типу int (т. е. возвращается *код* символа в таблице Unicode). Если предпринимается попытка прочесть символ за концом файла, то метод возвращает –1.

```
string ReadLine();
```

Считывает и возвращает очередную строку из текстового потока this. Признаком конца строки считается конец файла или наличие одного из двух вариантов *маркеров конца строки*: символ с кодом 10 ('\n') или пара символов с кодами 13 и 10 ('\r', '\n'); маркер конца строки в возвращаемую строку не включается. Если данный метод вызывается после достижения конца файла, то он возвращает значение null.

```
string ReadToEnd();
```

Считывает все оставшиеся символы из текстового потока this (начиная с текущего символа) и возвращает их в виде одной строки, содержащей как «обычные символы», так и маркеры конца строк (маркер конца файла в возвращаемую строку не включается). Если данный метод вызывается после достижения конца файла, то он возвращает пустую строку "" .

При считывании данных из текстовых файлов оказывается полезным следующее свойство класса StreamReader.

```
bool EndOfStream { get; }
```

Данное свойство возвращает true, если достигнут конец файла, и false в противном случае.

Запись данных в текстовый поток

Для записи данных в классе StreamWriter предусмотрены методы Write и WriteLine, которые перегружены для различных типов записываемых данных. В любом из приведенных методов записи данные добавляются в конец файла.

```
void Write(object value);
void Write(bool value);
void Write(числовой_тип value);
void Write(char value);
void Write(string value);
```

Каждый из методов записывает в поток текстовое представление параметра value (для получения текстового представления вызывается метод ToString указанного параметра). Если параметр равен null, то в файл ничего не записывается.

```
void Write(string fmt, params object[] args);
```

Данный метод обеспечивает *форматный вывод данных*, использующий форматную строку fmt.

Для любого из перечисленных выше методов Write имеется «парный» к нему метод WriteLine с тем же набором параметров. Метод WriteLine записывает в текстовый поток те же данные, что и соответствующий ему метод Write, после чего дописывает в поток *маркер конца строки*. Имеется также вариант метода WriteLine без параметров:

```
void WriteLine();
```

Используемый маркер конца строки берется из свойства NewLine класса StreamWriter:

```
string NewLine { get; set; }
```

По умолчанию это свойство возвращает строку "\r\n".

Стандартный текстовый поток для ввода-вывода: Console

В любых консольных приложениях платформы .NET доступен особый текстовый поток для ввода-вывода: *консольное окно*. Для управления этим окном предназначен класс Console, определенный в пространстве имен System.

В оконных приложениях .NET консольное окно по умолчанию не создается, однако попытки ввода-вывода данных с использованием класса Console не приводят к возбуждению исключений, поскольку в этой ситуации класс Console связывается с «пустыми» текстовыми потоками. Для использования консольного окна в оконных приложениях необходимо указать в качестве типа приложения вариант «Console Application».

```
static TextReader In { get; }
static TextWriter Out { get; }
static TextWriter Error { get; }
```

Каждое из данных свойств обеспечивает доступ к соответствующему текстовому потоку, связанному с консольным окном: In — стандартный поток для ввода, Out — стандартный поток для вывода, Error — поток для вывода сообщений об ошибках..

В стандартных консольных потоках используется формат кодирования, принятый по умолчанию для консольных окон в текущей версии Windows; в частности, в русской версии Windows используется кодовая страница 866 «Cyrillic (DOS)».

```
static void SetIn(TextReader newIn);
static void SetOut(TextWriter newOut);
static void SetError(TextWriter newError);
```

Методы предназначены для перенаправления стандартных консольных потоков ввода-вывода; в качестве их новых значений можно указать, например, текстовые файлы.

```
static int Read();
static string ReadLine();
```

Данные методы предназначены для ввода символов и строк из стандартного потока ввода In. Они работают аналогично соответствующим методам класса StreamReader, однако следует учитывать, что текст, набранный в консольном окне, передается программе (и, следовательно, обрабатывается методами ввода) только после нажатия клавиши [Enter]. Следует также иметь в виду, что нажатие клавиши [Enter] записывает во входной поток два символа с кодами 13 и 10, которые могут считываться методом Read как обычные символы.

```
static void Write(object value);
static void Write(bool value);
static void Write(числовой_тип value);
static void Write(char value);
static void Write(string value);
static void Write(string fmt, params object[] args);
```

Эти методы предназначены для вывода данных различных типов в стандартный поток вывода Out. Кроме того, у класса Console имеются «парные» к Write методы WriteLine с тем же набором параметров и метод WriteLine без параметров.

Вспомогательные классы для работы с файлами, каталогами и дисками

Перечисление DriveType

Данное перечисление определяет *типы логических дисков*:

- Unknown (0) — неизвестный тип логического диска;
- NoRootDirectory (1) — логический диск, не имеющий корневого каталога;
- Removable (2) — устройство для сменных носителей;
- Fixed (3) — жесткий локальный диск;
- Network (4) — сетевой диск;
- CDRom (5) — устройство для чтения компакт-дисков;
- Ram (6) — виртуальный диск, созданный в оперативной памяти.

Класс Path

Класс Path предназначен для манипулирования *именами файлов*. Все его методы являются классовыми. Основная часть методов предназначена для выделения требуемого элемента из имени файла или каталога. Все эти методы имеют один параметр name типа string и возвращают объект типа string:

- GetPathRoot — имя корневого каталога (вида "C:\", "C:" или "\");
- GetDirectoryName — путь к файлу, включающий имя диска, но не содержащий завершающий символ «\» (если name завершается символом «\», то возвращается строка name без завершающего символа «\»);
- GetFileName — имя файла вместе с расширением. Если name оканчивается символом-разделителем для диска или каталога, то возвращается пустая строка;
- GetFileNameWithoutExtension — имя файла *без расширения*;
- GetExtension — расширение файла, включая предшествующую точку;
- GetFullPath — полное имя файла (если name содержит относительное имя, то к нему добавляется имя текущего каталога вместе с именем диска; если name начинается с символа «\», то к нему добавляется имя текущего диска).

```
static bool HasExtension(string name);
```

Возвращает true, если имя файла name содержит непустое расширение, и false в противном случае. Считается, что расширение является пустым, если имя файла оканчивается точкой или вообще не содержит символов «точка».

```
static string ChangeExtension(string name, string ext);
```

Возвращает имя name, расширение которого заменено на расширение ext. Параметр ext может либо содержать, либо не содержать начальный символ «.», в любом случае расширение (даже пустое) будет начинаться с точки. Если параметр ext равен null, то из имени удаляется расширение *вместе с точкой*.

Часть методов класса Path связана с созданием *временных файлов*. Укажем один из них:

```
static string GetRandomFileName();
```

Возвращает случайную строку, которую можно использовать в качестве имени файла или каталога. Строка состоит из цифр и строчных латинских букв и включает собственно имя из 8 символов и расширение из 3 символов.

Классы File и FileInfo

Классы File и FileInfo предназначены для обработки файлов как элементов файловой системы, без доступа к их содержимому. Класс File содержит только классовые методы; при этом первым параметром любого метода является имя name обрабатываемого файла. Класс FileInfo содержит свойства и экземплярные методы, для доступа к которым необходимо создать объект данного класса, указав в его конструкторе имя обрабатываемого файла.

Поскольку все методы класса File имеют свои аналоги (свойства или методы) в классе FileInfo, ниже приводятся описания только методов класса File.

```
static void Copy(string name, string newName[, bool overwrite]);
```

Создает копию файла name с именем newName. Файл name и путь, указанный в имени newName, должны существовать; имя newName не должно быть именем существующего каталога; если параметр overwrite не указан или равен false, то имя newName не может быть именем существующего файла. Если параметр overwrite равен true и файл newName существует, то его содержимое заменяется на содержимое файла name (если файл newName закрыт на запись, то возбуждается исключение).

```
static void Move(string name, string newName);
```

Переименовывает файл name, заменяя его имя на newName. В качестве имени name нельзя указывать имя каталога; файл name и путь, указанный в имени newName, должны существовать; имя newName не должно быть именем существующего каталога или файла. В имени newName можно указывать каталог, находящийся на другом диске. Новое имя файла может совпадать со старым; в этом случае метод не выполняет никаких действий.

```
static void Delete(string name);
```

Удаляет файл с именем name. Если файл не существует, то метод не выполняет никаких действий. Если в имени файла указан несуществующий диск и/или каталог или если файл существует, но удален быть не

может (например, если он в данный момент используется другим приложением), а также если в качестве name указано имя существующего каталога, то возбуждается исключение.

```
static bool Exists(string name);
```

Возвращает true, если файл с именем name существует, и false в противном случае. В частности, метод возвращает false, если параметр name является пустой строкой или именем *каталога* (пусть даже и существующего). Данный метод никогда не возбуждает исключения.

Класс FileInfo содержит также ряд экземплярных свойств (только для чтения), которые не имеют соответствий в классе File. Перечислим эти свойства (все они, кроме Length, могут использоваться и в том случае, когда объект this типа FileInfo связан не с файлом, а с каталогом):

- Directory — объект типа DirectoryInfo, содержащий информацию о каталоге, в котором содержится файл;
- DirectoryName — строка, содержащая полный путь к файлу (данный путь не обязан существовать; завершающий символ «\» указывается только в случае корневого каталога);
- Extension — строка, содержащая расширение файла (непустое расширение дополняется слева точкой);
- Name — строка, содержащая имя файла (с расширением) без предшествующего пути;
- FullName — строка, содержащая полное имя файла;
- Length — число типа long, равное длине файла в байтах.

Классы Directory и DirectoryInfo

Классы Directory и DirectoryInfo предназначены для работы с *каталогами*. Класс Directory, подобно ранее рассмотренному классу File, содержит только классовые методы; при этом первым параметром большинства методов является имя name обрабатываемого каталога. Класс DirectoryInfo, подобно классу FileInfo, содержит свойства и экземплярные методы, для доступа к которым необходимо создать объект данного класса, указав в его конструкторе имя каталога.

Некоторые методы класса Directory не имеют соответствий в классе DirectoryInfo.

```
static string GetCurrentDirectory();
static void SetCurrentDirectory(string newName);
```

Метод GetCurrentDirectory позволяет определить, а метод SetCurrentDirectory — изменить текущий каталог, т. е. *рабочий каталог приложения*.

Все прочие методы класса Directory имеют аналоги в классе DirectoryInfo (здесь эти аналоги не описываются).

```
static string[] GetFiles(string name[, string mask[, SearchOption option]]);
static string[] GetDirectories(string name[, string mask[,SearchOption option]]);
static string[] GetFileSystemEntries(string name[, string mask]);
```

Данные методы возвращают массив строк с полными именами файлов (метод GetFiles), подкаталогов (метод GetDirectories) или одновременно файлов и подкаталогов (метод GetFileSystemEntries) из каталога name. Возвращаемые имена подкаталогов не оканчиваются символом «\».

Если указан параметр mask, то возвращаются имена только тех файлов/каталогов, которые удовлетворяют указанной *маске*; если параметр mask не указан, то его значение считается равным «*», что соответствует *любым* именам файлов/каталогов. Помимо символа «*», обозначающего любое количество любых символов, в маске можно указывать обычные символы, а также символ «?», обозначающий ровно один произвольный символ.

Если указан параметр option перечислимого типа SearchOption, то, в зависимости от его значения, поиск файлов/каталогов может проводиться не только в указанном каталоге (вариант SearchOption.TopDirectoryOnly), но и во всех его подкаталогах любого уровня вложенности (вариант SearchOption.AllDirectories). Если параметр option отсутствует, то поиск проводится только в указанном каталоге.

```
static DirectoryInfo CreateDirectory(string name);
```

Создает последовательность вложенных каталогов, указанных в строке name, и возвращает объект типа DirectoryInfo, связанный с созданным каталогом. Если указанный каталог уже существует, то метод возвращает данный каталог, не выполняя никаких дополнительных действий. Если параметр name является именем *существующего файла*, то возбуждается исключение.

```
static void Move(string name, string newName);
```

Переименовывает файл или каталог name, заменяя его имя на newName. Файл/каталог name, а также путь, указанный в параметре newName, должны существовать; имя newName не должно быть именем существующего файла/каталога. Заметим, что в данном методе (в отличие от метода Move класса File) новое имя не может совпадать со старым, а переименование должно проводиться в пределах одного и того же диска.

```
static void Delete(string name[, bool recursive]);
```

Если параметр recursive не указан или равен false, то метод обеспечивает удаление *пустого* каталога с именем name (если каталог не является пустым, то возбуждается исключение). Если параметр recursive равен true, то удаляемый каталог может содержать файлы и подкаталоги, которые также удаляются. Если каталог с указанным именем не существует или доступен только для чтения, то возбуждается исключение. Исключение возбуждается также в случае, когда указанный каталог является рабочим каталогом какого-либо работающего приложения.

```
static bool Exists(string name);
```

Возвращает true, если каталог с именем name существует, и false в противном случае. В частности, метод возвращает false, если параметр name является пустой строкой или именем *файла* (пусть даже и существующего). Данный метод никогда не возбуждает исключения.

Класс DriveInfo

Класс DriveInfo предназначен для получения информации о *логических дисках* компьютера.

Для получения объекта типа DriveInfo можно вызвать конструктор класса, передав ему в качестве строкового параметра букву требуемого логического диска или любое допустимое имя файла или каталога, содержащее имя диска. Имеется также классовый метод GetDrives (без параметров), который возвращает массив объектов DriveInfo, связанных со *всеми* обнаруженными на компьютере логическими дисками.

Вся информация о логическом диске, связанном с объектом типа DriveInfo, доступна через его свойства. К первой группе относятся свойства (только для чтения), обращение к которым никогда не приводит к возбуждению исключения:

- Name — строка, содержащая имя корневого каталога диска в формате "*<буква>*:\", например, "C:\";
- RootDirectory — объект типа DirectoryInfo, связанный с корневым каталогом диска;
- DriveType — перечисление типа DriveType, определяющее *тип* диска;
- IsReady — свойство логического типа, определяющее, доступен ли указанный диск.

Вторую группу образуют свойства, имеющие смысл только для доступных дисков; если диск недоступен, то обращение к ним приводит к возбуждению исключения. Все эти свойства, кроме свойства VolumeLabel, доступны только для чтения:

- DriveFormat — строка с описанием формата диска, например, "FAT", "FAT32", "NTFS" (жесткие диски), "CDFS" (CD- и DVD-диски) и т. д.;
- TotalSize — размер диска в байтах (целое типа long);
- TotalFreeSize — размер свободного пространства диска в байтах (целое типа long);
- AvailableFreeSize — размер свободного пространства диска в байтах, которое доступно для текущего пользователя (целое типа long);
- VolumeLabel — строка с именем метки диска.

Чтение и запись данных с помощью методов класса File

Класс File включает ряд методов, позволяющих организовать чтение или запись файловых данных, не требующие начальных действий по открытию файла и завершающих действий по его закрытию (указанные действия выполняются автоматически).

```
static byte[] ReadAllBytes(string path);
static string[] ReadAllLines(string path[, Encoding encoding]);
static string ReadAllText(string path[, Encoding encoding]);
```

Эти методы обеспечивают чтение содержимого файла с именем path и его запись в оперативную память. Для чтения двоичных данных предназначен метод ReadAllBytes, записывающий файловое содержимое в массив байтов. Для чтения текстовых данных предназначены методы ReadAllLines и ReadAllText, первый из которых записывает файловое содержимое в массив строк, а второй – в одну строку, содержащую, по-

мимо текста, и маркеры конца строк. При чтении из текстового файла можно дополнительно указать его кодировку, используя параметр encoding.

```
static void WriteAllBytes(string path, byte[] bytes);
static void WriteAllLines(string path, string[] contents[, Encoding encoding]);
static void WriteAllText(string path, string contents[, Encoding encoding]);
static void AppendAllText(string path, string contents[, Encoding encoding]);
```

Эти методы предназначены для записи данных в файл с именем path. Если файл с указанным именем не существует, то он создается. Метод WriteAllBytes записывает в файл набор байтов bytes, прочие методы – строковые данные contents, представленные либо в виде массива строк, либо в виде одной строки, включающей маркеры конца строк. Метод AppendAllText, в отличие от остальных методов, не удаляет прежнее содержимое файла (новые данные дописываются в конец файла). Все методы, связанные с записью текстовых данных, могут содержать параметр encoding, определяющий используемую при записи кодировку.

Применение этих методов требует размещения всех файловых данных в оперативной памяти, что, как правило, оказывается менее эффективным по сравнению с алгоритмами, выполняющими поэлементную файловую обработку.

Чтобы снизить затраты, связанные с выделением оперативной памяти, в версии .NET Framework 4.0 в указанный набор методов были включены следующие варианты методов, обрабатывающих наборы строк:

```
static IEnumerable<string> ReadLines(string path[, Encoding encoding]);
static void WriteAllLines(string path, IEnumerable<string> contents[, Encoding encoding]);
static void AppendAllLines(string path, IEnumerable<string> contents[, Encoding encoding]);
```

В этих методах строки могут размещаться в любых коллекциях, реализующих интерфейс IEnumerable<string> (например, в динамических массивах List<string>), и, главное, при использовании последовательностей, связанных с технологией LINQ, оказывается возможным «поэлементное» выполнение операций файлового чтения/записи. Например, в случае использования метода ReadLines считывание элементов из файла выполняется не в момент выполнения данного метода (как при использовании метода ReadAllLines), а впоследствии, при обработке каждого элемента полученной последовательности в цикле foreach, что позволяет получить более эффективный *построчный* вариант обработки текстовых файлов. Новые варианты методов WriteAllLines и AppendAllLines также оказываются более эффективными, если указываемая в них строковая последовательность contents генерируется в ходе выполнения запросов LINQ, поскольку в этой ситуации строковые элементы последовательности записываются в файл *по мере их формирования*, и в оперативной памяти не требуется выделять место для *одновременного* хранения всех элементов.

Указанные преимущества методов, использующих последовательности LINQ, обеспечиваются благодаря особому свойству запросов LINQ – их *отложенному*, или «ленивому» характеру.