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 (FileAccess.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).