Лабораторная работа. Программирование ввода-вывода

Занятие 1. Сохранение пользовательских настроек

Вас приняли на работу в крупную страховую компанию. Руководитель поручает вам дополнить существующее приложение Windows Forms диалогом для ввода пользователь­ских настроек. При этом возможность сохранения настроек требуется не только адми­нистраторам, но и обычным пользователям. Диалог планируется добавить в приложе­ние для учета потенциальных клиентов, с которым часто будут работать на общедоступ­ных компьютерах. При этом важно не допустить просмотра конфиденциальной инфор­мации посторонними.

Вопросы

Ответьте на следующие вопросы руководства:

1. Кратко опишите, как вы планируете хранить информацию пользователей.
2. Что придется изменить в проекте приложения, если потребуется сжимать индивиду­альные данные пользователей, чтобы уменьшить объема хранимой информации?

Занятие 2. Отслеживание активности старых серверов

Вы — разработчик в крупной ИТ-компании. В следующем месяце предполагается от­ключить множество устаревших серверов. Однако, на эти серверы неизвестные пользо­ватели до сих пор записывают данные, но определить их пока не удается. Вам предло­жено написать небольшое приложение для установки на серверы, которое будет отправ­лять сообщения по электронной почте руководству при создании или сохранении фай­ла на сервере.

Результаты опроса

■ ИТ-менеджер

«Я уверена, что некоторые пользователи до сих пор работают с устаревшими серве­рами, не подозревая об этом. Если нам удастся в течение недели отследить актив­ность старых серверов, то я смогу перевести этих пользователей на новые серверы, а старые можно будет отключить, сэкономив немалые средства на их поддержке».

■ Руководитель департамента разработки

«Один из наших программистов несколько недель назад пытался написать приложе­ние для мониторинга файлов, но не справился с этим. Его идея состояла в том, что­бы вести список файлов и проверять его каждые пять минут, эта задача отнимала слишком много времени».

Вопросы

Ответьте на следующие вопросы руководства:

1. Какое приложение позволит решить проблему ИТ-отдела?
2. Как вы собираетесь отслеживать активность файловых серверов?
3. Как справиться с проблемой быстродействия мониторинга файлов на серверах?

Варианты упражнений

Создание приложения для поиска файлов

* Упражнение 1 Напишите приложение для поиска заданного файла на диске.
* Упражнение 2 Добавьте код, использующий класс FileStream и позволяющий про­сматривать файл в текстовом окне.
* Упражнение 3 В заключение добавьте возможность сжатия найденного файла.

Создание простого хранилища конфигурационных данных

* Упражнение 1 Создайте приложение Windows Forms, позволяющее пользователям сохранять данные в изолированном хранилище.
* Упражнение 2 Для проверки запустите созданное приложение под учетными запи­сями различных пользователей.
* Упражнение 3 Измените приложение так, чтобы оно сохраняло некоторые данные уровня — это поможет различать данные, одинаковые для всех пользователей.

Вопросы на закрепление материала

1. Какие изменения можно отслеживать посредством FileSystemWatcher? (Укажите все  
верные ответы.)

A. Создание файлов.

B. Создание папок.

C. Изменение файлов.

D. Переименование файлов.

E. None.

2. Верно ли, что следующий код изменяет расширение файла?  
' VB

Dim ourPath As String = "cAboot.ini" Path.ChangeExtension(ourPath, "bak")

// C#

string ourPath = @"c:\boot.ini"; Path.ChangeExtension(ourPath, "bak");

А- Да.,. В. Нет.

3. Какие методы класса FileStream изменяют свойство Position? (Укажите все верные от-  
веты.)

A. Read.

B. Lock.

C. Write.

D. Seek.

4. Как приказать объекту Stream Writer переписать модифицированные данные в другой  
поток? (Укажите все верные ответы.)

A. Закрыть the StreamWriter.

B. Вызвать метод Flush класса Stream Writer.

C. Установить значение свойства AutoFlush класса StreamWriter равным true.

D. Закрыть поток.

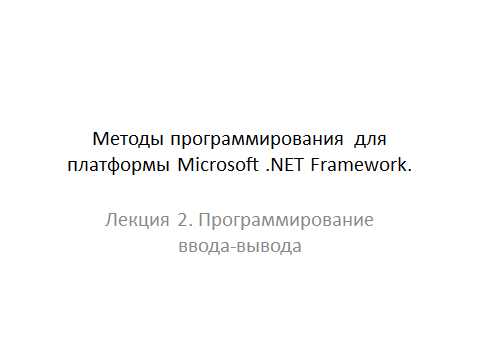
5. Как создать FileStream для записи, открыв существующий файл, либо, в случае его  
отсутствия, создав новый файл? (Укажите все верные ответы.)

A. Создать новый экземпляр класса FileStream, указав член перечислимого  
FileMode.OpenOrCreate.

B. Для создания объекта FileStream вызвать метод File.Create.

C. Вызвать метод File.Open, указав член перечислимого FileMode.OpenOrCreate.

D. Вызвать метод File.Open, указав член перечислимого File Mode.Open.



Методические указания

В общем случае понятие ПОТОК — это абстрактное понятие, которое обозначает динамическую изменяющуюся во времени последовательность чего-либо.

Применительно к обсуждаемым проблемам ввода/вывода в программах на C#, поток – это последовательность байтов, связанная с конкретными устройствами компьютера (дисками, дисплеями, принтерами, клавиатурами) посредством системы ввода/вывода.

Система ввода/вывода обеспечивает для программиста стандартные и не зависящие от физического устройства средства представления информации и управления потоками ввода/вывода. Действия по организации ввода/вывода обеспечиваются стандартными наборами, как правило, одноименных функций ввода/вывода со стандартным интерфейсом.

Функции, обеспечивающие взаимодействие с различными устройствами ввода/вывода, объявляются в различных классах. Вопрос "ОТКУДА ВЫЗВАТЬ функцию" часто становится более важным, чем вопрос "КАК ПОСТРОИТЬ выражение вызова функции".

**Потоки: байтовые, символьные, двоичные**

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

С другой стороны, значительный объем работ, для которых, собственно, и используется вычислительная техника, предполагает работу с символами, а не с байтами (заполнение экранной формы, вывод информации в наглядном и легко читаемом виде, текстовые редакторы).

Символьно-ориентированные потоки, предназначенные для манипулирования с символами, а не с байтами, являются потоками ввода/вывода более высокого уровня. В рамках Framework .NET определены соответствующие классы, которые при реализации операций ввода/вывода обеспечивают автоматическое преобразование данных типа byte в данные типа char и обратно.

В дополнение к байтовым и символьным потокам в C# определены два класса, реализующих механизмы считывания и записи информации непосредственно в двоичном представлении (потоки BynaryReader и BynaryWriter).

**Общая характеристика классов потоков**

Основные особенности и правила работы с устройствами ввода/вывода в современных языках высокого уровня описываются в рамках классов потоков. Для языков платформы .NET местом описания самых общих свойств потоков является класс System.IO.Stream.

Назначение этого класса заключается в объявлении общего стандартного набора операций (стандартного интерфейса), обеспечивающих работу с устройствами ввода/вывода, независимо от их конкретной реализации, от источников и получателей информации.

Процедуры чтения и записи информации определяется следующим (список неполон!) набором свойств и АБСТРАКТНЫХ методов, объявляемых в этом классе.

В рамках Framework .NET, независимо от характеристик того или иного устройства ввода/вывода, программист ВСЕГДА может узнать:

* можно ли читать из потока – bool CanRead (если можно – значение должно быть установлено в true);
* можно ли писать в поток – bool CanWrite (если можно – значение должно быть установлено в true);
* можно ли задать в потоке текущую позицию – bool CanSeek (если последовательность, в которой производится чтение/запись, не является жестко детерминированной и возможно позиционирование в потоке – значение должно быть установлено в true);
* позицию текущего элемента потока – long Position (возможность позиционирования в потоке предполагает возможность программного изменения значения этого свойства);
* общее количество символов потока (длину потока) – long Length.

В соответствии с общими принципами реализации операций ввода/вывода, для потока предусмотрен набор методов, позволяющих реализовать:

* чтение байта из потока с возвращением целочисленного представления СЛЕДУЮЩЕГО ДОСТУПНОГО байта в потоке ввода – int ReadByte();
* чтение определенного (count) количества байтов из потока и размещение их в массиве buff, начиная с элемента buff[index], с возвращением количества успешно прочитанных байтов – int Read(byte[] buff, int index, int count);
* запись в поток одного байта – void WriteByte(byte b);
* запись в поток определенного (count) количества байтов из массива buff, начиная с элемента buff[index], с возвращением количества успешно записанных байтов – int Write(byte[] buff, int index, int count);
* позиционирование в потоке – long Seek (long index, SeekOrigin origin) (позиция текущего байта в потоке задается значением смещения index относительно позиции origin);
* для буферизованных потоков принципиальна операция флэширования (записи содержимого буфера потока на физическое устройство) – void Flush();
* закрытие потока – void Close().

Множество классов потоков ввода/вывода в Framework .NET основывается (наследует свойства и интерфейсы) на абстрактном классе Stream. При этом классы конкретных потоков обеспечивают собственную реализацию интерфейсов этого абстрактного класса.

Наследниками класса Stream являются, в частности, три класса байтовых потоков:

* BufferedStream – обеспечивает буферизацию байтового потока. Как правило, буферизованные потоки являются более производительными по сравнению с небуферизованными;
* FileStream – байтовый поток, обеспечивающий файловые операции ввода/вывода;
* MemoryStream – байтовый поток, использующий в качестве источника и хранилища информации оперативную память.

Манипуляции с потоками предполагают НАПРАВЛЕННОСТЬ производимых действий. Информацию из потока можно ПРОЧИТАТЬ, а можно ее в поток ЗАПИСАТЬ. Как чтение, так и запись предполагают реализацию определенных механизмов байтового обмена с устройствами ввода/вывода.

Свойства и методы, объявляемые в соответствующих классах, определяют специфику потоков, используемых для чтения и записи:

* TextReader,
* TextWriter.

Эти классы являются абстрактными. Это означает, что они не "привязаны" ни к какому конкретному потоку. Они лишь определяют интерфейс (набор методов), который позволяет организовать чтение и запись информации для любого потока.

В частности, в этих классах определены следующие методы, определяющие базовые механизмы символьного ввода/вывода. Для класса TextReader:

* int Peek() – считывает следующий знак, не изменяя состояние средства чтения или источника знака. Возвращает следующий доступный знак, не считывая его в действительности из потока входных данных;
* int Read(...) – несколько одноименных перегруженных функций с одним и тем же именем. Читает значения из входного потока. Вариант int Read() предполагает чтение из потока одного символа с возвращением его целочисленного эквивалента или –1 при достижении конца потока. Вариант int Read (char[] buff, int index, int count) и его полный аналог int ReadBlock(char[] buff, int index, int count) обеспечивает прочтение максимально возможного количества символов из текущего потока и записывает данные в буфер, начиная с некоторого значения индекса;
* string ReadLine() – читает строку символов из текущего потока. Возвращается ссылка на объект типа string;
* string ReadToEnd() – читает все символы, начиная с текущей позиции символьного потока, определяемого объектом класса TextReader, и возвращает результат как ссылка на объект типа string;
* void Close() – закрывает поток ввода.

Для класса TextWriter, в свою очередь, определяется:

* множество перегруженных вариантов функции void Write(...) со значениями параметров, позволяющих записывать символьное представление значений базовых типов (смотреть список базовых типов) и массивов значений (в том числе и массивов строк);
* void Flush() – обеспечивает очистку буферов вывода. Содержимое буферов выводится в выходной поток;
* void Close() – закрывает поток вывода.

Эти классы являются базовыми для классов:

* StreamReader – содержит свойства и методы, обеспечивающие считывание СИМВОЛОВ из байтового потока,
* StreamWriter – содержит свойства и методы, обеспечивающие запись СИМВОЛОВ в байтовый поток.

Вышеуказанные классы включают методы своих "предков" и позволяют осуществлять процессы чтения-записи непосредственно из конкретных байтовых потоков. Работа по организации ввода/вывода с использованием этих классов предполагает определение соответствующих объектов, "ответственных" за реализацию операций ввода/вывода, с явным указанием потоков, которые предполагается при этом использовать.

Еще одна пара классов потоков обеспечивает механизмы символьного ввода– вывода для строк:

* StringReader,
* StringWriter.

В этом случае источником и хранилищем множества символов является символьная строка.

Интересно заметить, что у всех ранее перечисленных классов имеются методы, обеспечивающие закрытие потоков, и не определены методы, обеспечивающие открытие соответствующего потока. Потоки открываются в момент создания объекта-представителя соответствующего класса. Наличие функции, обеспечивающей явное закрытие потока, принципиально. Оно связано с особенностями выполнения управляемых модулей в Framework .NET. Время начала работы сборщика мусора заранее не известно.

**Предопределенные потоки ввода/вывода**

Предопределенные потоки ввода/вывода используются для реализации ввода/вывода в рамках консольных приложений. Это еще одна категория потоков.

В классах предопределенных потоков, в отличие от ранее рассмотренных потоков, явным образом задаются источник (откуда) и место размещения (куда) информации.

Основные свойства и методы этих потоков определены в классе System.Console. Класс Console является достаточно сложной конструкцией.

В рамках этого класса определены свойства, обеспечивающие доступ к предопределенным потокам. В классе System.Console таких потоков три (In, Out, Error).

Ниже описываются свойства этих потоков:

* Standard input stream – является объектом – представителем класса TextReader. По умолчанию поток In ориентирован на получение информации с клавиатуры.
* Standard output stream – является объектом – представителем класса TextWriter. По умолчанию обеспечивает вывод информации на дисплей.
* Standard error stream – также является объектом – представителем класса TextWriter. И также по умолчанию выводит информацию на дисплей.

В классе System.Console также определены следующие СТАТИЧЕСКИЕ (!) функции – члены (вернее, множества СТАТИЧЕСКИХ ПЕРЕГРУЖЕННЫХ функций), обеспечивающие процедуры ввода/вывода:

* Write и WriteLine;
* Read и ReadLine.

При этом функции-члены класса Console ПО УМОЛЧАНИЮ обеспечивают связь с конкретным потоком. Например, при вызове метода

Console.WriteLine(...); // Список параметров опущен

информация направляется в поток, который представляется определенным в классе статическим свойством Console.Out. Это свойство предоставляет доступ к объекту – представителю класса TextWriter. Таким образом, класс Console также обеспечивает вызов соответствующей функции-члена от имени объекта – представителя класса TextWriter. Непосредственное обращение к члену класса Console (то есть к потоку), отвечающего за текстовый вывод, имеет следующую форму вызова:

Console.Out.WriteLine(...);

Вызов определенного в классе Console метода ReadLine

Console.ReadLine(...);

обеспечивает получение информации через поток, доступ к которому предоставляется статическим свойством Console.In. Непосредственный вызов аналогичной функции от имени члена класса TextReader выглядит следующим образом:

Console.In.ReadLine(...);

Функция – представитель класса Console, отвечающая за непосредственное направление информации в поток вывода сообщений об ошибках, не предусмотрена.

Вывод информации об ошибках может быть реализован путем непосредственного обращения к standard error output stream:

Console.Error.WriteLine(...);

Если вспомнить, что свойство Error возвращает ссылку на объект – представитель класса TextWriter и, так же как и свойство Out, обеспечивает вывод в окно консольного приложения, разделение сообщений, связанных с нормальным ходом выполнения приложения и сообщений об ошибках, становится более чем странным.

Информация, выводимая приложением в "штатном" режиме, и сообщения об ошибках одинаково направляются в окно приложения на экран консоли. Однако в классе Console реализована возможность перенаправления соответствующих потоков. Например, стандартный поток вывода можно перенаправить в файл, а поток сообщений об ошибках оставить без изменений — направленным в окно приложения. Результат перенаправления очевиден.

Процедуры перенаправления потоков осуществляются с помощью методов void SetIn(TextReader in), void SetOut(TextWriter out), void SetErr(TextWriter err) и будут рассмотрены ниже.

**Файловый ввод/вывод**

FileMode enumeration:

описывает, каким образом операционная система должна открывать файл.

|  |  |
| --- | --- |
| **Имя члена** | **Описание** |
| Append | Открывается существующий файл и выполняется поиск конца файла или же создается новый файл. FileMode.Append можно использовать только вместе с FileAccess.Write. Любая попытка чтения заканчивается неудачей и создает исключение ArgumentException |
| Create | Описывает, что операционная система должна создавать новый файл. Если файл уже существует, он будет переписан. Для этого требуется FileIOPermissionAccess.Write и FileIOPermissionAccess.Append. System.IO.FileMode.Create эквивалентно следующему запросу: если файл не существует, использовать CreateNew; в противном случае использовать Truncate |
| CreateNew | Описывает, что операционная система должна создать новый файл. Для этого требуется FileIOPermissionAccess.Write. Если файл уже существует, создается исключение IOException |
| Open | Описывает, что операционная система должна открыть существующий файл. Возможность открыть данный файл зависит от значения, задаваемого FileAccess. Если данный файл не существует, создается исключение System.IO.FileNotFoundException |
| OpenOrCreate | Указывает, что операционная система должна открыть файл, если он существует, в противном случае должен быть создан новый файл. Если файл открыт с помощью FileAccess.Read, требуется FileIOPermissionAccess.Read. Если файл имеет доступ FileAccess.ReadWrite и данный файл существует, требуется FileIOPermissionAccess.Write. Если файл имеет доступ FileAccess.ReadWrite и файл не существует, в дополнение к Read и Write требуется FileIOPermissionAccess.Append |
| Truncate | Описывает, что операционная система должна открыть существующий файл. После открытия должно быть выполнено усечение файла таким образом, чтобы его размер стал равен нулю. Для этого требуется FileIOPermissionAccess.Write. Попытка чтения из файла, открытого с помощью Truncate, вызывает исключение |

FileAccess enumerations:

|  |  |
| --- | --- |
| **Члены перечисления** | **Описание** |
| Read | Доступ к файлу для чтения. Данные могут быть прочитаны из файла. Для обеспечения возможностей чтения/записи может комбинироваться со значением Write |
| ReadWrite | Доступ к файлу для чтения/записи |
| Write | Доступ к файлу для записи. Данные могут быть записаны в файл. Для обеспечения возможностей чтения/записи может комбинироваться со значением Read |

using System;

using System.IO;

namespace fstream00

{

class Class1

{

static string[] str = {

"1234567890",

"qwertyuiop",

"asdfghjkl",

"zxcvbnm", };

static void Main(string[] args)

{

int i;

// Полное имя файла.

string filename = @"D:\Users\WORK\Cs\fstream00\test.txt";

string xLine = "";

char[] buff = new char[128];

for (i = 0; i < 128; i++) buff[i] = (char)25;

// Запись в файл.

FileStream fstr = new FileStream(filename,

FileMode.Create,

FileAccess.Write);

BufferedStream buffStream = new BufferedStream(fstr);

StreamWriter streamWr = new StreamWriter(buffStream);

for (i = 0; i < str.Length; i++)

{

streamWr.WriteLine(str[i]);

}

streamWr.Flush();

streamWr.Close();

Console.WriteLine("–––––––––––––––––––––––––––––––––––––––––––");

fstr = new FileStream(filename,

FileMode.Open,

FileAccess.Read);

StreamReader streamRd = new StreamReader(fstr);

for ( ; xLine != null; )

{

xLine = streamRd.ReadLine();

Console.WriteLine(xLine);

}

Console.WriteLine("––––––––––––––––––––––––––––––––––––––––––––");

fstr.Seek(0, SeekOrigin.Begin);

streamRd.Read(buff,0,10);

Console.WriteLine(new string(buff));

Console.WriteLine("1––––––––––––––––––––––––––––––––––––––––––");

streamRd.Read(buff,0,20);

Console.WriteLine(new string(buff));

Console.WriteLine("2––––––––––––––––––––––––––––––––––––––––––");

Console.WriteLine(streamRd.Read(buff,0,15));

i = (int)fstr.Seek(–20, SeekOrigin.Current);

Console.WriteLine(streamRd.ReadLine());

Console.WriteLine("3––––––––––––––––––––––––––––––––––––––––––");

}

}

}

Листинг 11.1.

Пример перенаправления потоков. В файл можно записать информацию, используя привычные классы и методы!

using System;

using System.Collections.Generic;

using System.Text;

using System.IO;

namespace OutFile

{

class Program

{

static void Main(string[] args)

{

string buff;

FileStream outFstr, inFstr; // Ссылки на файловые потоки.

// Ссылка на выходной поток. Свойства и методы,

// которые обеспечивают запись в...

StreamWriter swr;

// Ссылка на входной поток. Свойства и методы,

// которые обеспечивают чтение из...

StreamReader sr;

// Класс Console – средство управления ПРЕДОПРЕДЕЛЕННЫМ потоком.

// Сохранили стандартный выходной поток,

// связанный с окошком консольного приложения.

TextWriter twrConsole = Console.Out;

// Сохранили стандартный входной поток, связанный с буфером клавиатуры.

TextReader trConsole = Console.In;

inFstr = new FileStream

(@"F:\Users\Work\readme.txt", FileMode.Open, FileAccess.Read);

sr = new StreamReader(inFstr); // Входной поток, связанный с файлом.

outFstr = new FileStream

(@"F:\Users\Work\txt.txt", FileMode.Create, FileAccess.Write);

swr = new StreamWriter(outFstr); // Выходной поток, связанный с файлом.

// А вот мы перенастроили предопределенный входной поток.

// Он теперь связан не с буфером клавиатуры, а с файлом, открытым для чтения.

Console.SetIn(sr);

Console.SetOut(swr);

while (true)

{

// Но поинтересоваться в предопределенном потоке отностительно

// конца файла невозможно.

// Такого для предопределенных потоков просто не предусмотрено.

if (sr.EndOfStream) break;

// А вот читать - можно.

buff = Console.ReadLine();

Console.WriteLine(buff);

}

Console.SetOut(twrConsole);

Console.WriteLine("12345");

Console.SetOut(swr);

Console.WriteLine("12345");

Console.SetOut(twrConsole);

Console.WriteLine("67890");

Console.SetOut(swr);

Console.WriteLine("67890");

sr.Close();

inFstr.Close();

swr.Close();

outFstr.Close();

}

}

}

Листинг