**МИНИСТЕРСТВО образования ставропольского края**

**государственное бюджетное Профессиональное образовательное учреждение «курсавский региональный колледж «интеграл»**

|  |  |
| --- | --- |
|  |  |

**Лабораторный практикум по**

**МДК 01.01 Системное программирование**

для специальности:

09.02.03 «Программирование в компьютерных системах»

**с. Курсавка**

**2020 г.**

Введение

В пособии рассмотрены вопросы использования интерфейса программирования приложений (API) Win32 для разработки системных программ управления файловой системой, управления процессами и потоками, асинхронным вводом-выводом. В пособии приводятся примеры программирования процесса включения функции в динамическую библиотеку и вызова этой функции из динамической библиотеки. Кроме того, представлены примеры программирования пользовательского интерфейса с использованием библиотеки классов Microsoft Foundation Classes (MFC). В качестве среды программирования используется Visual Studio C# .NET 2017. Все программы, которые представлены в лабораторных работах, отлажены под управление OC Windows. Пособие рассчитано на студентов старших курсов специальности «Программирование в компьютерных системах», которые хотят освоить тонкости программирования приложений на языке Visual Studio С#.

Содержание

[Практическая работа №1 Функции получения системной информации 5](#_Toc526753475)

[Практическая работа №2 Чтение, запись текстовых и бинарных файлов, текстовый редактор 7](#_Toc526753476)

[Практическая работа №3 Чтение/запись текстового файла в кодировке Windows 1251 11](#_Toc526753477)

[Практическая работа №4 Печать текстового документа 14](#_Toc526753478)

[Практическая работа №5 Чтение/запись бинарных файлов с использованием потока данных 19](#_Toc526753479)

[Практическая работа №6 Управление буфером обмена с данными в текстовом и графическом форматах 23](#_Toc526753480)

[Практическая работа №7 Имитация нажатия комбинации клавиш <Alt>+<PrintScreen> 26](#_Toc526753481)

[Практическая работа №8 Запись в файлы текущих состояний экрана каждые пять секунд 28](#_Toc526753482)

[Практическая работа №9 Экран 30](#_Toc526753483)

[Определение разрешение экрана 30](#_Toc526753484)

[Практическая работа №1 Получение списка установленных принтеров 33](#_Toc526753485)

[Как распечатать документ? 34](#_Toc526753486)

[Как показать окно предварительного просмотра перед печатью 36](#_Toc526753487)

[Практическая работа №11 Работа с мышью и клавиатурой 37](#_Toc526753488)

[Скрыть и показать указатель мыши? 37](#_Toc526753489)

[Меняем кнопки мыши местами 37](#_Toc526753490)

[Работа с клавиатурой 39](#_Toc526753491)

[Переключение раскладки клавиатуры программно. 39](#_Toc526753492)

[Посылка нажатия клавиш программно 40](#_Toc526753493)

[Определение состояния клавиш-индикаторов 43](#_Toc526753494)

[Последнее нажатие на клавишу или на кнопку мыши 43](#_Toc526753495)

[Практическая работа №12 Как узнать число процессоров в системе? 46](#_Toc526753496)

[Получаем список всех процессов, запущенных в системе 46](#_Toc526753497)

[Практическая работа №13 Определение операционной системы пользователя 48](#_Toc526753498)

[Практическая работа №14 49](#_Toc526753499)

[Определяем подключение мышки к системе, узнать число кнопок у мыши, размер вашего монитора и другую информацию? 49](#_Toc526753500)

[Как узнать, что пользователь изменил разрешение экрана? 50](#_Toc526753501)

[Практическая работа №15 Изменение времени 51](#_Toc526753502)

[Практическая работа №16 Диски 52](#_Toc526753503)

[Как узнать тип диска и его свойства? 52](#_Toc526753504)

[Практическая работа №17 54](#_Toc526753505)

[Вызов файла справки СНМ 72](#_Toc526753506)

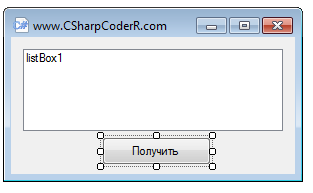
Лабораторный практикум по системному программированию в среде Visual Studio С#

# **Практическая работа №1 Функции получения системной информации**

Для получения информации о температуре центрального процессора и системной платы воспользуемся одной из базовых технологий для централизованного управления и слежения за работой различных частей компьютерной инфраструктуры под управлением платформы Windows - **Windows Management Instrumentation (WMI)**. Так как WMI построен по объектно-ориентированному принципу, то все данные операционной системы представлены в виде объектов и их свойств, и методов. Все классы группируются в пространства имен, которые иерархически упорядочены и логически связаны друг с другом по определенной технологии или области управления. В WMI имеется одно корневое пространство имен Root, которое в свою очередь имеет 4 подпространства: *CIMv2*, *Default*, *Security*и***WMI***. Воспользуемся последним подпространством «WMI» и его классом «**MSAcpi\_ThermalZoneTemperature**». Данный класс содержит данные об текущей температуре**системной платы** и **центрального процессора**.   
     Создайте проект Windows Form в Microsoft Visual Studio и добавьте на главную форму два компонента:

* ListBox
* Button

У вас получится примерно такой вариант:

[](http://3.bp.blogspot.com/-wdz8iSfRBgo/Upt9EKiRyPI/AAAAAAAAC2o/bx12tK15lK4/s1600/1.png)

      Сделайте двойной клик левой клавишей мыши по компоненту «Button» и перейдите в автоматически созданный метод «**button1\_Click**», добавьте в него приведенный ниже листинг.

listBox1.Items.Clear();

try

{

Double CPUtprt = 0;

System.Management.ManagementObjectSearcher MOS = new System.Management.ManagementObjectSearcher("root\\WMI",

"SELECT \* FROM MSAcpi\_ThermalZoneTemperature");

foreach (System.Management.ManagementObject Mo in MOS.Get())

{

CPUtprt = Convert.ToDouble(Convert.ToDouble(Mo.GetPropertyValue("CurrentTemperature".ToString())) - 2732) / 10.0;

listBox1.Items.Add(" CPU: " + CPUtprt.ToString() + " ° C");

}

}

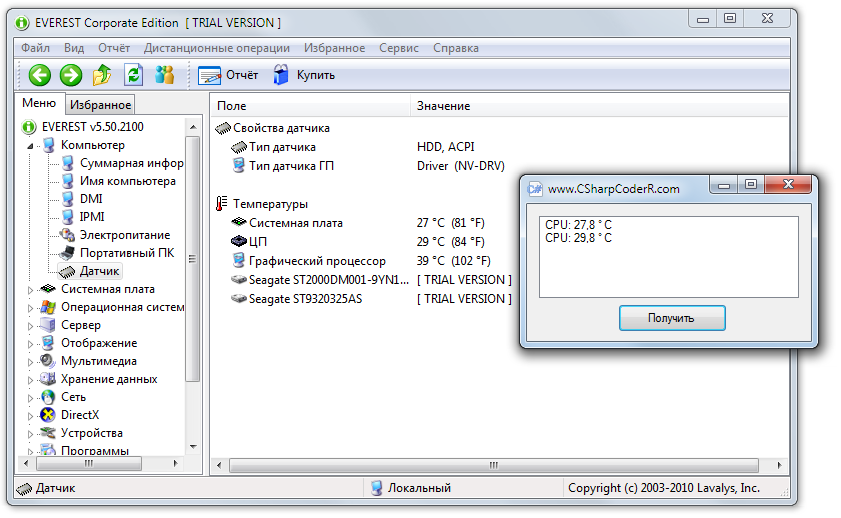
catch (ManagementException ex)

{

MessageBox.Show("Ошибка получения данных " + ex.Message);

}

      Запустите ваш проект и нажмите на единственную кнопку на главной форме. Для проверки работы программы воспользуйтесь любым приложением, предоставляющим информацию с температурных датчиков. Например, вы можете воспользоваться программой **Everest**.

[](http://1.bp.blogspot.com/-iBVJxisqPtM/Upt9hgRR-AI/AAAAAAAAC2w/nT7QMdFILs8/s1600/2.png)

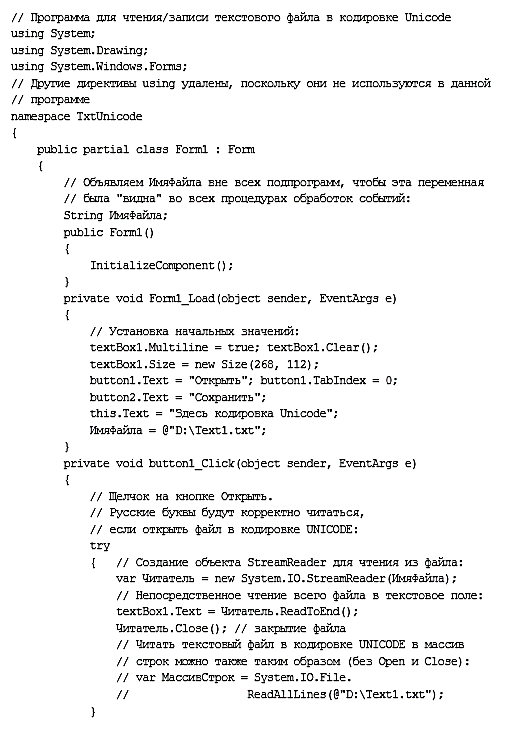
# **Практическая работа №2** **Чтение, запись текстовых и бинарных файлов, текстовый редактор**

Чтение/запись текстового файла в кодировке **Unicode**. Обработка исключений **try...catch**

Очень распространенной задачей является сохранение данных на диске в текстовом (не двоичном) формате. Понятно, что такое деление условно, поскольку и текстовые, и нетекстовые файлы на самом деле являются двоичными файлами. Но если нетекстовые файлы открыть, например, Блокнотом, мы увидим то, что называют "нечитаемым месивом" — отсюда такая классификация. Часто данные сохраняют на диск именно в текстовом формате, поскольку в этом случае сохраненные данные можно читать, редактировать любым текстовым редактором, например, Блокнотом, или TextEdit, или другими. Следует также уметь и читать текстовые данные в своей пользовательской программе. Казалось бы, это очень простая задача. Например, чтение текстового файла сводится буквально к нескольким строчкам:

// Создание экземпляра StreamReader для чтения из файла: var Читатель = new System.IO.StreamReader(@"D:\Text1.txt"); // Непосредственное чтение всего файла в текстовое поле: textBox1.Text = Читатель.ReadToEnd(); Читатель.Close(); // закрытие файла Однако есть некоторые серьезные нюансы. Напишем программу, содержащую на экранной форме текстовое поле и две командные кнопки. При щелчке мышью на первой кнопке происходит чтение текстового файла в текстовое поле в кодировке Unicode. При щелчке на второй кнопке отредактированный пользователем текст в текстовом поле сохраняется в файл на диске. Запустим систему Visual Studio 12 и в окне New Project выберем шаблон Windows Forms Application Visual C#. Новую программу назовем TxtUnicode. Далее в форму из панели Toolbox перенесем текстовое поле и две командные кнопки. Для текстового поля в окне Properties сразу укажем для свойства Multiline значение True, чтобы текстовое поле имело не одну строку, а столько, сколько поместится в растянутом указателем мыши поле. Одна кнопка предназначена для открытия файла, а другая — для сохранения файла на машинном носителе. В листинге приведен текст этой программы. Листинг программы.







Несколько слов о блоках **try**, которые, как можно видеть, присутствуют в данном программном коде. Логика использования **try** следующая: попытаться (**try**) выполнить некоторую задачу, например прочитать файл. Если задача решена некорректно (например, файл не найден), то "перехватить" (**catch**) управление и обработать возникшую (исключительную, **Exception**) ситуацию. Как видно из текста программы, обработка исключительной ситуации свелась к информированию пользователя о недоразумении.

При обработке события "щелчок на кнопке Открыть" организован ввод файла **D:\Text1.txt**. Обычно в такой ситуации для выбора файла применяют элемент управления **OpenFileDialog**. Мы не стали использовать этот элемент управления для того, чтобы не "заговорить" проблему, а также свести к минимуму программный код. Далее создаем объект (поток) Читатель для чтения из файла. Для большей выразительности операций с данным объектом мы написали его название русскими буквами. Затем следует чтение файла **ИмяФайла** методом **ReadToEnd**() в текстовое поле **TextBox1**.**Text** и закрытие файла методом **Close**(). При обработке события "щелчок на кнопке Сохранить" аналогично организована запись файла на диск через объект Писатель. При создании объекта Писатель первым аргументом является **ИмяФайла**, а второй аргумент **false** указывает, что данные следует не добавить (**append**) к содержимому файла (если он уже существует), а перезаписать (**overwrite**). Запись на диск производится с помощью метода **Write**() из свойства **Text** элемента управления **TextBox1**. На рисунке приведен пример работы программы.

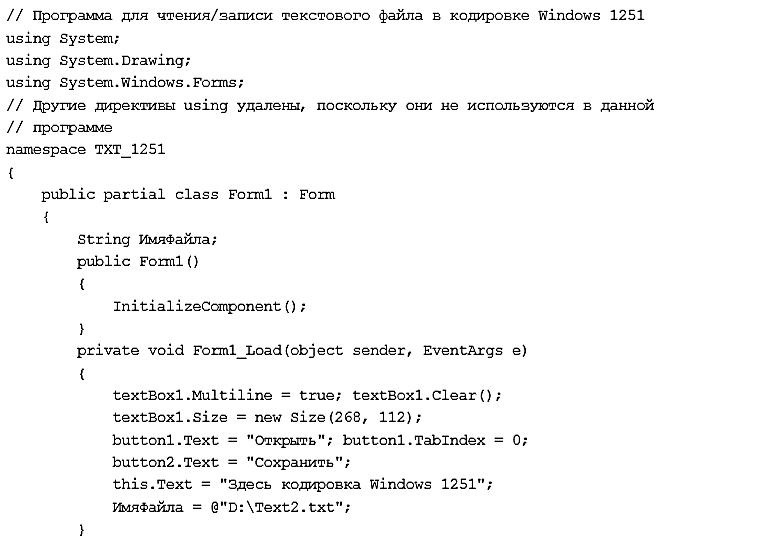
**ВНИМАНИЕ!** Запись текстового файла с помощью данной программы будет осуществляться в формате (кодировке) Unicode, как и чтение из файла. То есть, вы сможете читать эти файлы Блокнотом, редактировать их, но каждый раз при сохранении файлов из Блокнота вам требуется следить, чтобы кодировка была (оставалась) Unicode.

По умолчанию в редакторах обычно используется кодировка **ANSI** (кодировку **ANSI**, включающую русские буквы, называют еще **Windows** **1251**). Некоторые редакторы, например, **RPad** ("русский" Блокнот), вообще не работают с **Unicode**. Таким образом, на сегодняшний день пока что записывать тексты в кодировке **Unicode** — это, как бы, эксклюзив. Возможно, даже наверняка, в ближайшее время многое изменится в пользу **Unicode**, поскольку информационные технологии развиваются очень быстро.

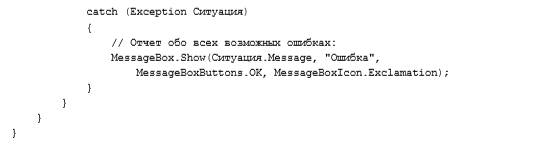
# **Практическая работа №3 Чтение/запись текстового файла в кодировке Windows 1251**

В данном примере также на экранной форме разместим текстовое поле и две командные кнопки (назначение этих элементов управления аналогично предыдущему при- меру). Однако чтение и запись текстового файла здесь происходит в кодировке Windows 1251 (ANSI). Поскольку структура этой программы аналогична предыдущей, сразу обратимся к ее коду в листинге.

Листинг







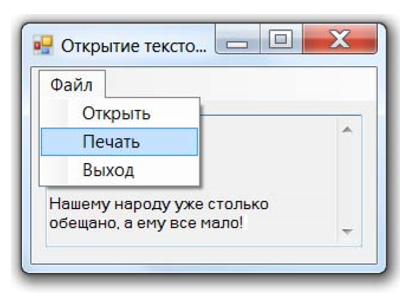
Текст этой программы отличается от предыдущей лишь тем, что здесь введен новый объект — Кодировка. Метод **GetEncoding(1251)** устанавливает кодовую страницу **Windows 1251** для объекта Кодировка. Можно убедиться в этом, если распечатать свойство **Кодировка.HeaderName**. При создании объекта Читатель используются уже два аргумента: имя файла и объект Кодировка, указывающий, в какой кодировке (для какой кодовой страницы) читать данные из текстового файла. А при создании объекта Писатель используются три аргумента: имя файла, установка **false** (для случая, если файл уже существует, нужно будет не добавлять новые данные, а создавать новый файл) и объект Кодировка, указывающий, в какой кодировке писать данные в файл. Заметьте, что при обработке события "щелчок на второй кнопке" мы объявили переменные Кодировка и Писатель просто как **var**, т. е. типы этих переменных выводятся из выражения инициализации.

# **Практическая работа №4 Печать текстового документа**

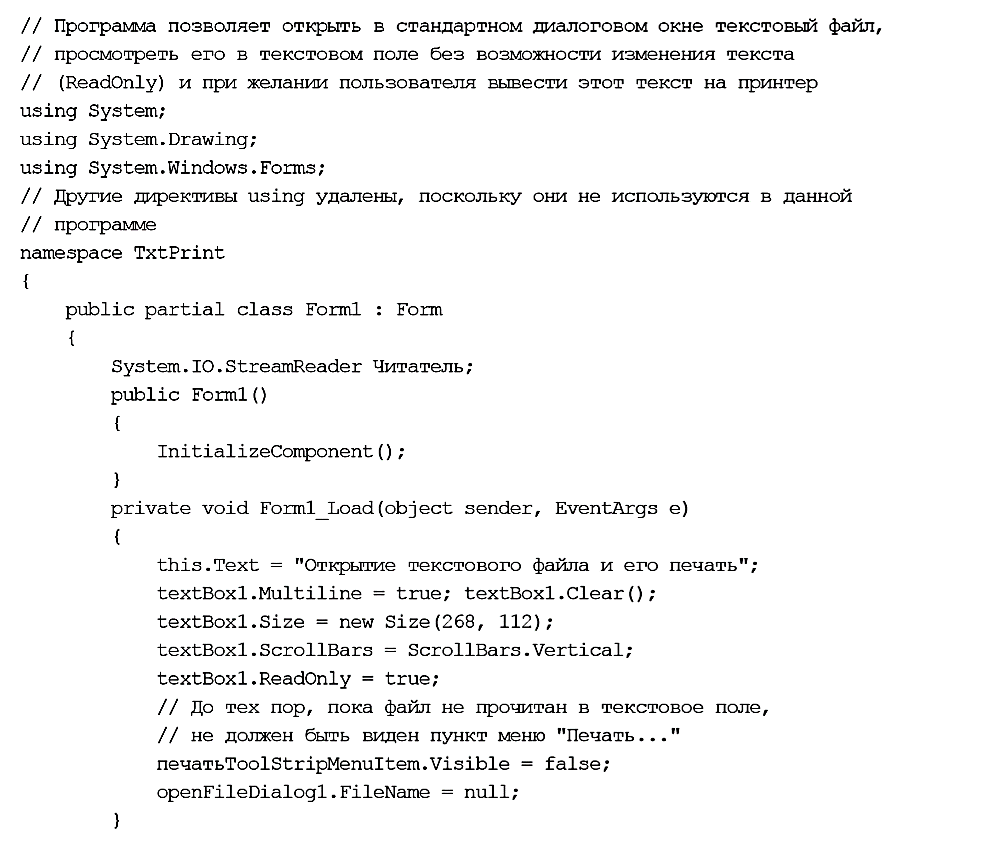
Любой текстовый редактор (и не только текстовый) должен иметь возможность печати на принтере. Мы сознательно не добавляли такую возможность в текстовые редакторы, приведенные в предыдущих разделах, чтобы не запутать читателя. Понятно, что чем больше функциональности имеет программа, тем сложнее ее программный код и тем труднее текст программы для понимания. А наша задача — выразительно и ярко демонстрировать технологии в максимально простой форме. Программа, представленная в этом примере, имеет следующие скромные возможности: открыть в стандартном диалоговом окне Windows текстовый файл, просмотреть его в окне программы (в текстовом поле) без возможности изменения текста (ReadOnly) и при желании пользователя вывести этот текст на принтер. Эту программу назовем TxtPrint.

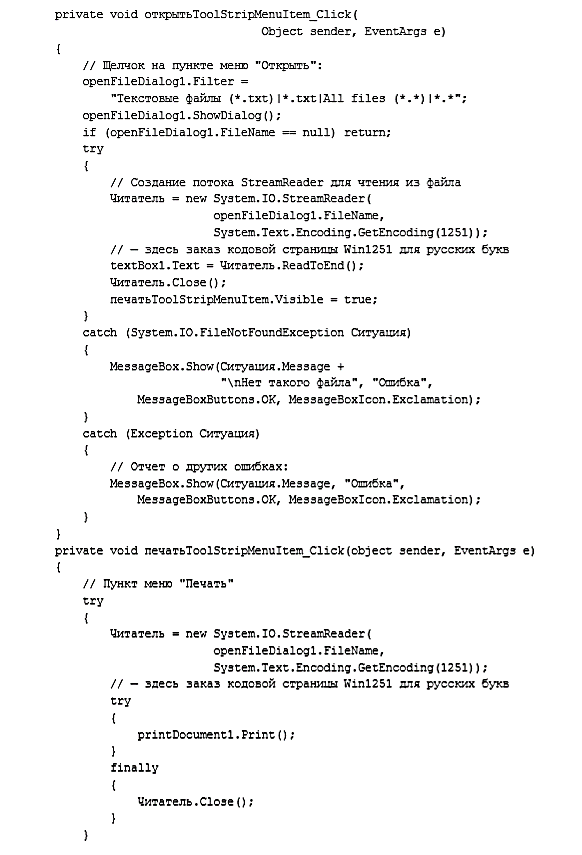
**Чтение, запись текстовых и бинарных файлов, текстовый редактор**

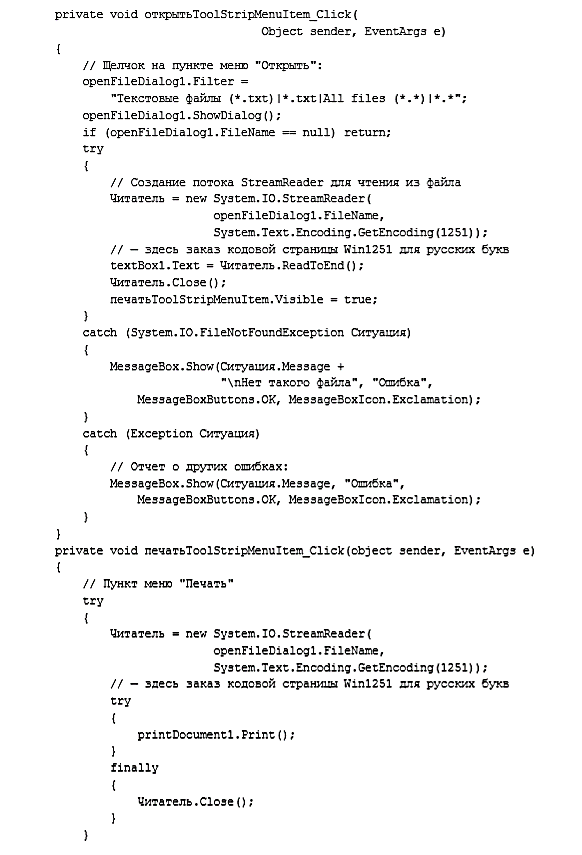
Запустив Visual Studio и выбрав шаблон Windows Forms Application C#, в дизайнере формы из панели элементов перенесем следующие элементы управления: текстовое поле **TextBox** со свойством **MultiLine = true**, меню **MenuStrip** с пунктами меню: Открыть, Печать и Выход (рис.), а также элементы управления **OpenFileDialog** и **PrintDocument**. Текст программы представлен в листинге

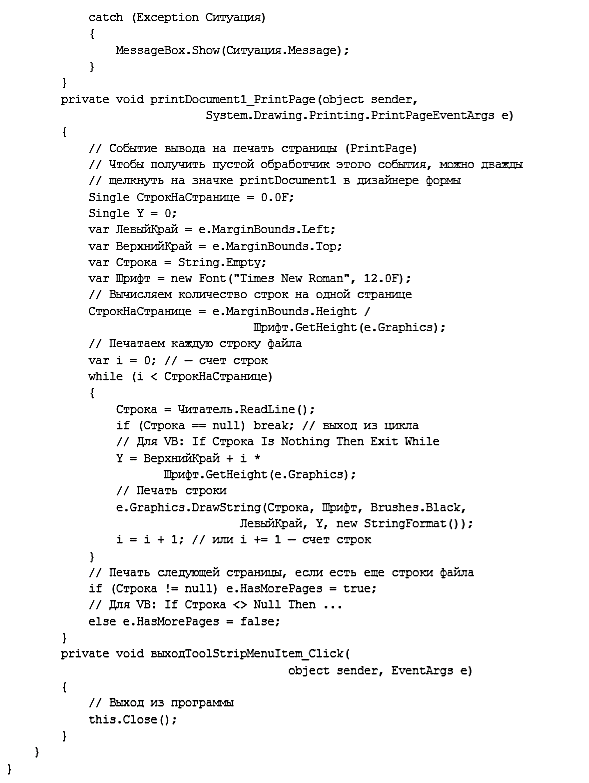


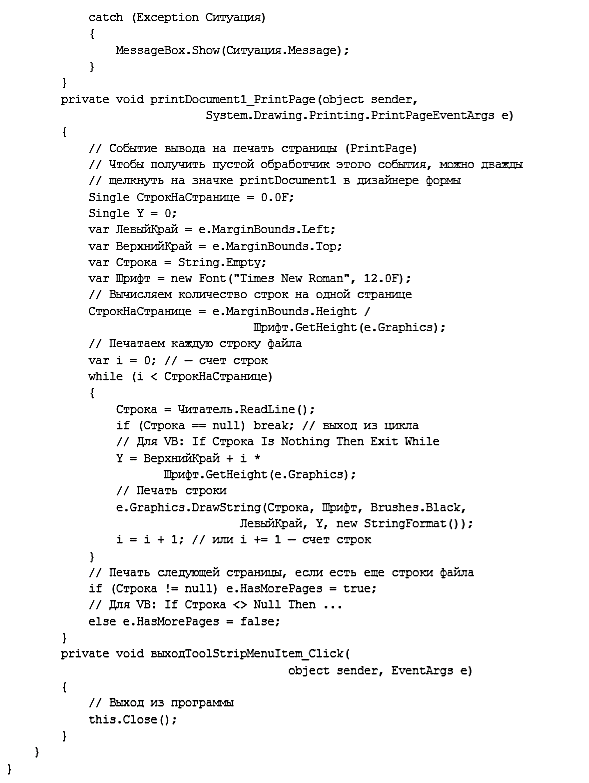
Листинг программы









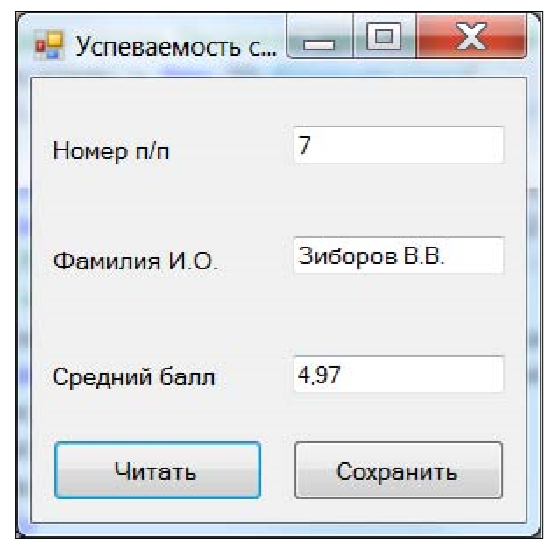


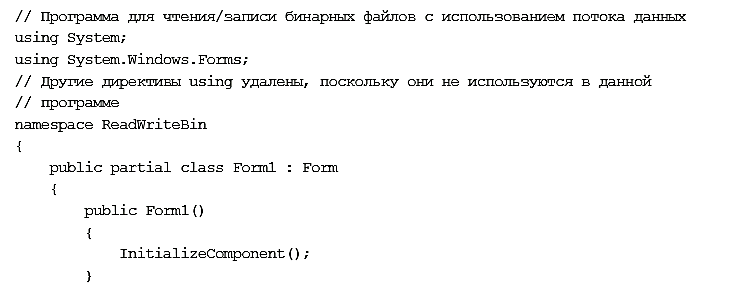
Здесь при обработке события загрузки формы **Form1\_Load** запрещаем пользователю редактировать текстовое поле: **ReadOnly** **= true**. Также назначаем свойству **ПечатьToolStripMenuItem.Visible = false** (пункт меню Печать), т. е. в начале работыпрограммы пункт меню Печать пользователю не виден (поскольку пока распечатывать нечего — необходимо вначале открыть текстовый файл). Остальные присваивания при обработке события **Form1**\_**Load** очевидны. При обработке события "щелчок на пункте меню Открыть" вызываем стандартный диалог **openFileDialog** и организуем чтение файла через создание потока **StreamReader**. Эти процедуры мы уже рассматривали подробно в разделах о текстовых редакторах, поэтому оставим их без комментария. Замечу только, что после чтения файла в текстовое поле назначаем видимость пункту меню Печать: **печатьToolStripMenuItem**.**Visible** = **true**, поскольку уже есть, что печатать на принтере (файл открыт). Представляет интерес обработка события "щелчок на пункте меню Печать". Здесь во вложенных блоках **try...finally...catch** программа еще раз создает поток **StreamReader**, а затем запускает процесс печати документа printDocument1.Print. Если ничего более не программировать, только метод printDocument1.Print, то принтер распечатает лишь пустую страницу. Чтобы принтер распечатал текст, необходимо обработать событие **PrintPage** (см. текст программы), которое создает объект **PrintDocument**. То есть роль метода **Print** — это создать событие **PrintPage**. Обратите внимание на обработку события **printDocument1**\_**PrintPage**. Пример обработки этого события приведен в MSDN. Вначале перечислены объявления переменных, значения некоторых из них получаем из аргументов события e, например, **ЛевыйКрай** — значение отступа от левого края, и т. д. Назначаем шрифт печати — **Times** **New** **Roman**, **12 пунктов**. Далее в цикле **while** программа читает каждую строку Строка из файла — **Читатель**.**ReadLine**(), а затем распечатывает ее командой (методом) **DrawString**. Здесь используется графический объект **Graphics**, который получаем из аргумента события e. В переменной **i** происходит счет строк. Если количество строк оказывается большим, чем число строк на странице, то происходит выход из цикла, поскольку страница распечатана. Если есть еще страницы, а программа выясняет это, анализируя содержимое переменной Строка, — если ее содержимое отличается от значения **null** (**Строка != null**), то аргументной переменной **e.HasMorePages** назначаем **true**, что инициирует опять событие **PrintPage**, и процедура **printDocument1**\_**PrintPage** начинает свою работу вновь. И так, пока не закончатся все страницы **e.HasMorePages** = **false** для печати на принтере.

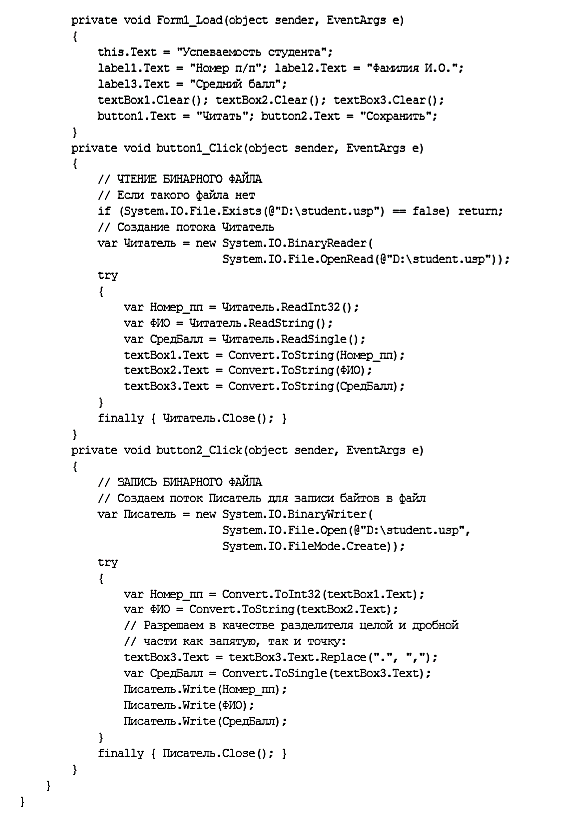
# **Практическая работа №5 Чтение/запись бинарных файлов с использованием потока данных**

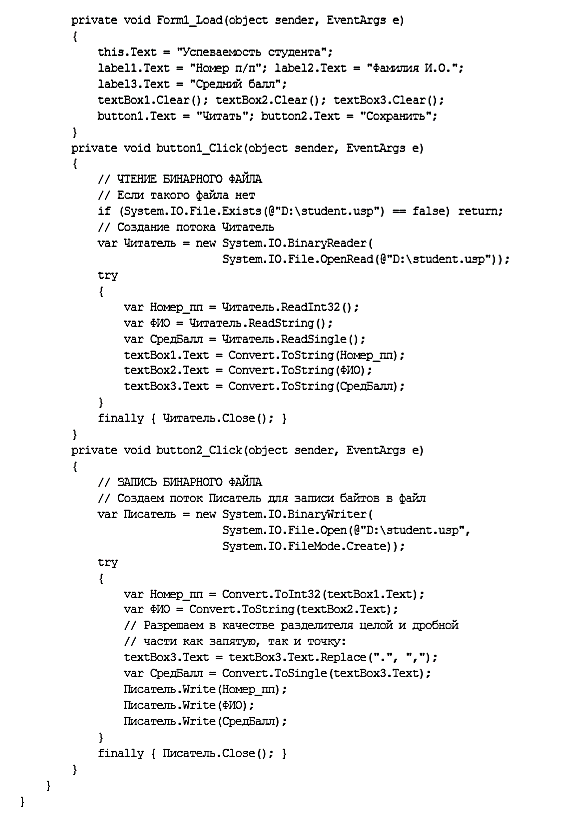
Обычно программа либо что-то читает с диска в оперативную память, либо что-то пишет на диск. Писать/читать можно либо в бинарный (двоичный) файл, либо в текстовый (литерный, строковый) файл. Разница между ними состоит в том, что текстовый файл можно прочитать текстовым редактором, например, Блокнотом, а бинарный — нет. Название "бинарный" (двоичный) — условное, поскольку, по сути, и текстовый, и бинарный файлы являются двоичными файлами.

Операции с двоичным файлом гораздо более скоростные, и такой файл занимает существенно меньшее пространство на диске. Зато текстовый файл можно читать и редактировать любым текстовым редактором. Обычно программист выбирает компромисс между этими двумя преимуществами. Приведем пример самого простейшего случая записи на диск бинарного файла с данными об успеваемости одного студента. Понятно, что эта программа может быть маленькой частью большой программы по обработке успеваемости студентов в вузе. Данная программа принимает от пользователя сведения только об одном студенте в текстовые поля формы. При нажатии кнопки Сохранить программа записывает введенные сведения в двоичный файл, а при нажатии кнопки Читать читает эти сведения из двоичного файла в текстовые поля формы. Итак, в форме мы должны иметь три текстовых поля, куда пользователь может записать соответственно номер студента по порядку, фамилию студента и его средний балл успеваемости. Поэтому в форму из панели **Toolbox** перенесем три текстовых поля **TextBox**, три метки **Label** и две командные кнопки: Читать и Сохранить. Таким образом, получим пользовательский интерфейс, показанный на рис. 4.10. Назовем программу **ReadWriteBin**. Текст ее приведен в листинге.









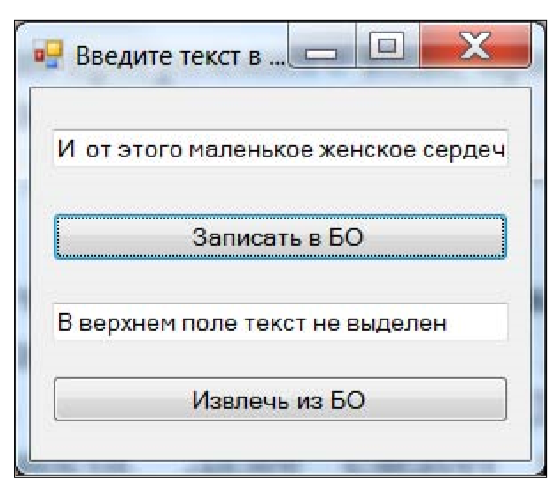
Как можно видеть, в процедуре обработки события загрузки формы организована инициализация (присвоение начальных значений) элементов формы: текстовых полей, меток и кнопок. Запись файла на диск происходит при обработке события **button2**.**Click**, т. е. щелчка мышью на кнопке Сохранить (рис.). С этой целью создаем поток байтов Писатель для открытия файла student.usp. Если такого файла не существует, то он создается (Create), а если файл уже есть, то он перезаписывается. Как видно из текста программы, для ее упрощения мы не использовали элемент управления OpenFileDialog открытия файла в диалоге. Далее преобразуем записанное в текстовых полях в более естественные типы данных. Номер по порядку Номер\_пп — это тип целых чисел, преобразование в целый тип может быть реализовано операцией Convert.ToInt32 (можно использовать другие функции преобразования), для переменной СредБалл (средний балл) больше всего подходит тип с плавающей точкой Single, при этом преобразование осуществляется операцией Convert.ToSingle. Преобразование для строковой переменной ФИО является необязательным и приведено для симметрии записей. Операторы Писатель.Write записывают эти данные в файл. После блока finally происходит обязательное закрытие (Close) файла. Чтение файла выполняется при обработке события "щелчок мышью на кнопке Читать". Как уже упоминалось, для максимального упрощения в данной программе не предусмотрено открытие файла через стандартное диалоговое окно, поэтому в начале процедуры выясняем, существует ли такой файл. Если файла D:\student.usp нет, то программируем выход (return) из обработчика данного события. Заметьте, чтобы программисту было максимально легко отслеживать ветви оператора if, мы написали: "Если файла нет, то return". При этом длинная ветвь логики "если файл есть" не включена непосредственно в оператор if. Поэтому такой фрагмент программного кода читается (воспринимается) программистом легко — это типичный пример хорошо написанного программного кода. Далее создается поток байтов Читатель из файла student.usp, открытого для чтения. Чтение из потока в каждую переменную реализовано с помощью функции ReadInt32 — читать из потока Читатель в переменную целого типа, аналогично функциям ReadString и ReadSingle. Затем осуществлено конвертирование этих переменных в строковый тип Convert.ToString. Как видно из программы, можно было изначально все текстовые поля записывать в файл без конвертирования, но при дальнейшем развитии этой программы значения полей все равно пришлось бы преобразовывать в соответствующий тип. После блока finally происходит закрытие (Close) файла. Блок finally выполнится всегда, даже если перед ним была команда return. Дальнейшее развитие этой программы может быть осуществлено путем добавления в файл сведений о других студентах. В таком случае при чтении файла количество студентов будет неизвестно. Тогда следует обработать ситуацию достижения конца файла:

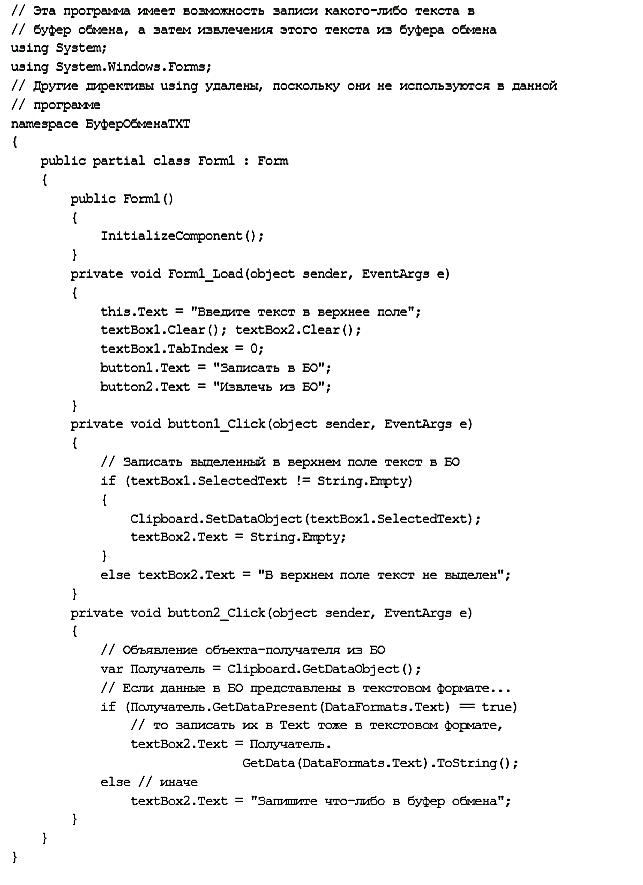
catch (EndOfStreamException e) а затем закрыть файл.

# **Практическая работа №6 Управление буфером обмена с данными в текстовом и графическом форматах**

**Буфер обмена с данными в текстовом формате**

Напишем программу для управления буфером обмена с данными в текстовом формате. Эта программа будет записывать какой-либо текст в буфер обмена (БО), а затем извлекать этот текст из БО. Запустим Visual Studio 12 и в окне New Project выберем шаблон Windows Forms Application Visual C#. Новую программу назовем **БуферОбменаTXT**. В форме создадим два текстовых поля, а также две командные кнопки под этими полями. Одну кнопку назовем Записать в БО, а другую — Извлечь из БО. Чтобы записать в БО какой-либо текст, нужно набрать его в верхнем поле, выделить (с помощью клавиш управления курсором при нажатой клавише <Shift>), а затем нажать кнопку Записать в БО. Нажимая эту кнопку, мы как бы моделируем комбинацию клавиш <Ctrl>+<C>. Теперь записанный в БО текст можно читать в каком-либо текстовом редакторе или вывести в нижнее текстовое поле прямо в нашей форме. Для этого служит кнопка Извлечь из БО (рис. 6.1). Текст данной программы приведен в листинге 6.1.





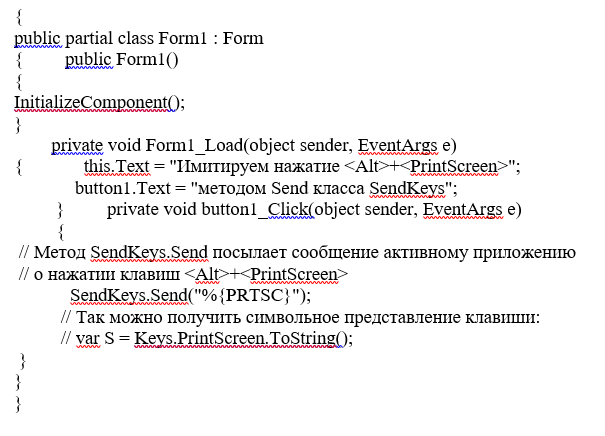
Как видно из программного кода, при обработке события "щелчок на верхней кнопке", если текст в верхнем поле выделен, то записываем его (SelectedText) в буфер обмена (Clipboard) командой (методом) SetDataObject, иначе (else) сообщаем в нижнем поле textBox2 о том, что в верхнем поле текст не выделен. Напомню, что, нажав кнопку Извлечь из БО, пользователь нашей программы в нижнем поле должен увидеть содержимое буфера обмена. Для этого объявляем объектную переменную Получатель. Это объект-получатель из буфера обмена. Эта переменная сознательно названа по-русски для большей выразительности. Далее следует проверка — в текстовом ли формате (DataFormat.Text) данные, представленные в буфере обмена. Если формат текстовый, то в текстовое поле TextBox2 записываем содержимое буфера обмена, используя функцию ToString. Эта функция конвертирует строковую часть объекта в строковую переменную.

# **Практическая работа №7 Имитация нажатия комбинации клавиш <Alt>+<PrintScreen>**

Как известно, при нажатии клавиши <PrintScreen> происходит копирование изображения экрана в буфер обмена, т. е. получаем так называемый screen shot — моментальный снимок экрана (другое название — screen capture). После извлечения из БО графической копии экрана в любом графическом редакторе (например, Paint) эту графическую копию можно редактировать. При нажатии комбинации клавиш <Alt>+<PrintScreen> в буфер обмена копируется не весь экран, а только активное окно. Для имитации нажатия сочетания клавиш <Alt>+<PrintScreen> современная система программирования Visual Studio в пространстве имен System.Windows.Forms имеет класс SendKeys, который предоставляет методы для отправки приложению сообщений о нажатиях клавиш, в том числе и нашей комбинации клавиш <Alt>+<PrintScreen>. Сейчас мы напишем программу, которая будет имитировать нажатие искомых клавиш. Для этого запустим Visual Studio 12 и в окне New Project выберем шаблон Windows Forms Application Visual C#. Новую программу назовем AltPrintScreen. Программа сознательно максимально упрощена, чтобы более выразительно показать ее суть. Здесь в форме имеется только командная кнопка. При обработке события "щелчок на кнопке" будем имитировать нажатие комбинации клавиш <Alt>+<PrintScreen> методом Send класса SendKeys. При этом форму можно как угодно менять — растягивать по ширине и по высоте, но после нажатия кнопки в форме изображение формы запишется в буфер обмена. Следует убедиться в этом — запустить MS Paint и извлечь графическую копию формы из буфера обмена. Текст программы представлен в листинге

// Программная имитация нажатия клавиш <Alt>+<PrintScreen>

// методом Send класса SendKeys using System; using System.Windows.Forms; // Другие директивы using удалены, поскольку они не используются в данной // программе namespace AltPrintScreen



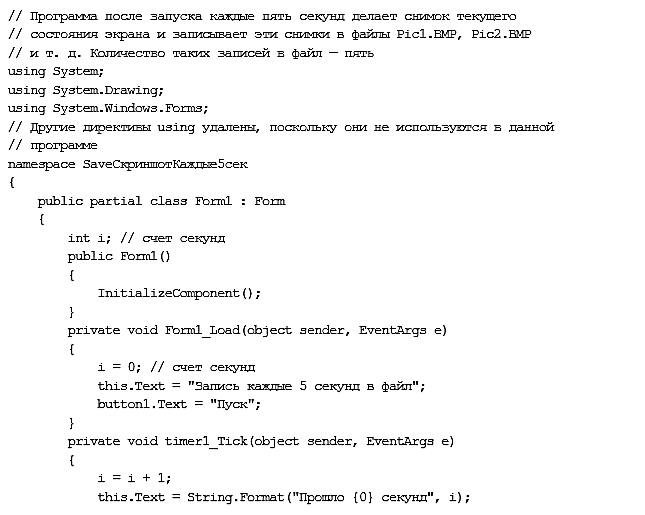
В этой программе при обработке события "щелчок на кнопке" мы решаем обсуждаемую задачу современными средствами. Метод Send класса SendKeys посылает сообщение активному приложению о нажатии комбинации клавиш <Alt>+<PrintScreen>. Нажатие клавиши <PrintScreen> генерирует код "{PRTSC}". Чтобы указать сочетание клавиш <Alt><PrintScreen>, следует в этот код добавить символ процента: "%{PRTSC}". Коды других клавиш можно посмотреть по адресу:

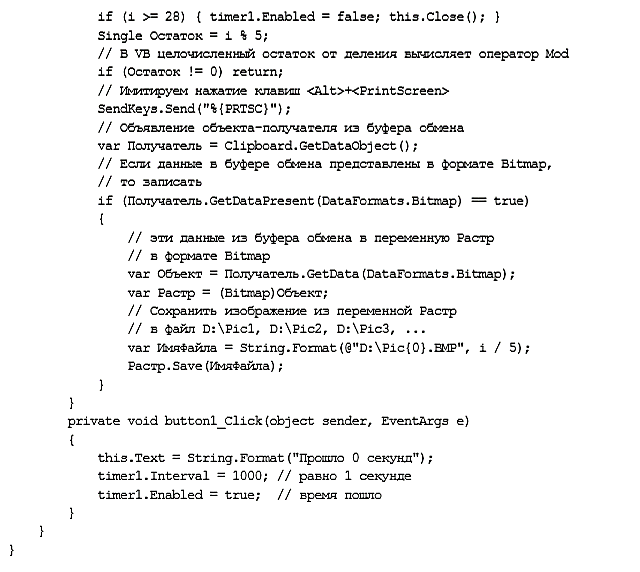
http://msdn.microsoft.com/ru-ru/library/system.windows.forms.sendkeys.aspx

# **Практическая работа №8 Запись в файлы текущих состояний экрана каждые пять секунд**

Как уже говорилось в предыдущем примере, работа с таймером — довольно распространенная задача. Напишем очередную программу в следующей постановке: после запуска программы через каждые пять секунд снимается текущее состояние экрана и записывается в файлы Pic1.BMP, ..., Pic5.BMP. Для решения этой задачи мы имеем уже весь инструментарий, он был рассмотрен в предыдущих примерах. А именно задание интервала времени в две секунды с по- мощью элемента управления Timer, эмуляция нажатия клавиш <Alt>+<PrintScreen> для записи текущего состояния экрана в буфер обмена и, наконец, чтение буфера обмена в формате изображения и запись этого изображения в файл BMP. Для решения задачи запустим Visual Studio 12 и в окне New Project выберем приложение шаблона Windows Forms Application Visual C#. Новую программу назовем SaveСкриншотКаждые5сек. Затем из панели элементов перетаскиваем в проектируемую форму элементы управления Timer и Button. Текст программы приведен в листинге

Листинг





Как видно из программного кода, структура программы включает в себя обработку события "щелчок на кнопке" button1\_Click и обработку события timer1\_Tick. В начале программы задаем переменную i, которая считает прошедшие секунды. При щелчке на кнопке Пуск задаем интервал времени Interval, равный 1000 миллисекунд, т. е. одной секунде. Далее даем команду таймеру начать отсчет времени timer1.Enabled = True, и через каждую секунду наступает событие timer1\_Tick(), т. е. управление переходит этой процедуре. При обработке события timer1\_Tick наращиваем значение переменной i, которая ведет счет секунд после старта таймера. Выражение i % 5 вычисляет целочисленный остаток после деления первого числового выражения на второе. Понятно, что если число i будет кратно пяти, то этот остаток будет равен нулю, и только в этом случае будет происходить имитация нажатия клавиш <Alt>+<PrintScreen> и запись содержимого буфера обмена в файл.

# **Практическая работа №9 Экран**

# **Определение разрешение экрана**

Для определения размеров экрана воспользуйтесь свойством PrimaryScreen.Bounds класса Screen из пространства имен System.windows. FormsScreen, как показано в листинге

private void butScreenRes\_Click(object sender, EventArgs e)

{

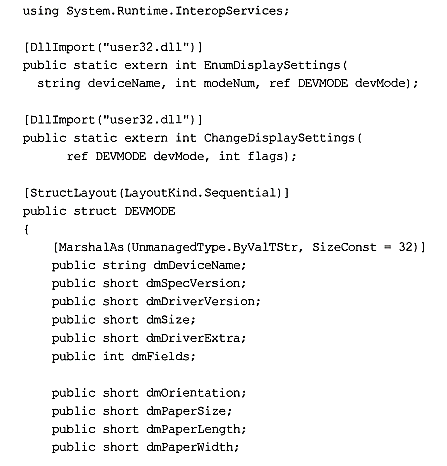
// Определяем размеры экрана

textBoxl.Text = Screen.PrimaryScreen.Bounds.width.ToString

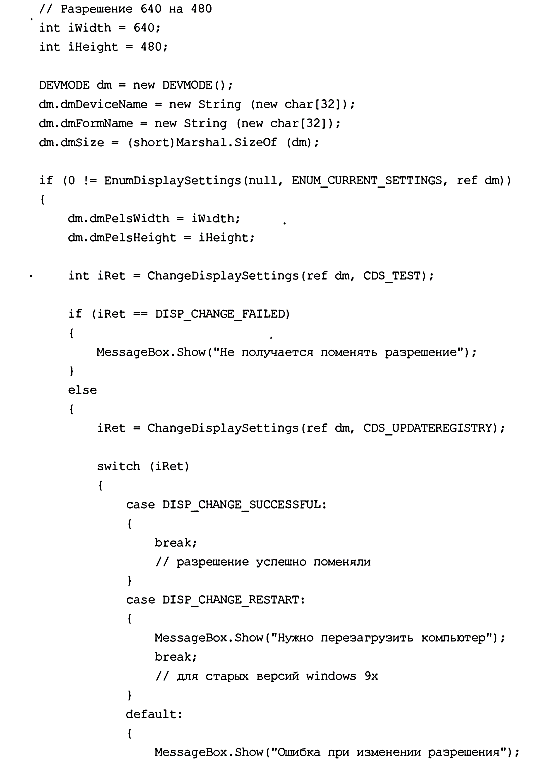
(J + "x" + Screen.PrimaryScreen.Bounds.Height.ToString (); )

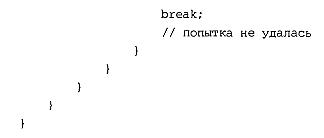
**Изменение разрешения экрана программным путем**

На данный момент .NET Framework не имеет встроенных методов для изме­нения разрешения экрана. Поэтому нам придется использовать две функции **Windows API: EnumDisplaySettings** и **ChangeDlsplaySetcing**. Описание этих функций есть в справочнике по функциям API на прилагаемом к книге ком­пакт-диске, поэтому ограничимся только кодом программы, приведенным в листинге 5.6.



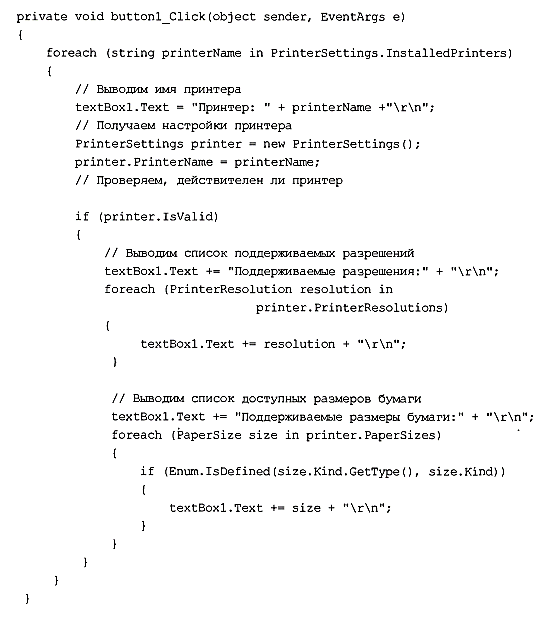






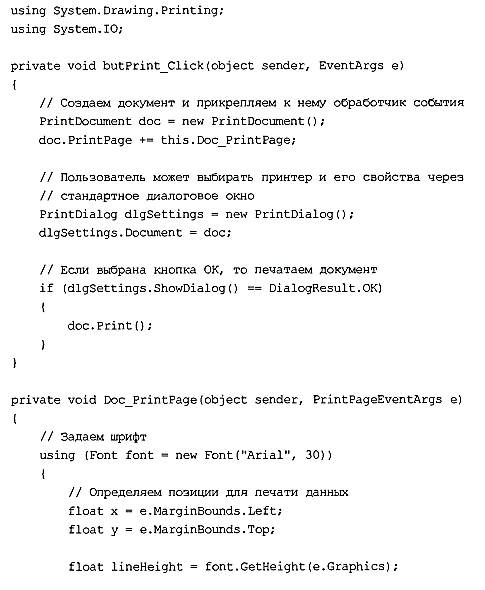
# **Практическая работа №1 Получение списка установленных принтеров**

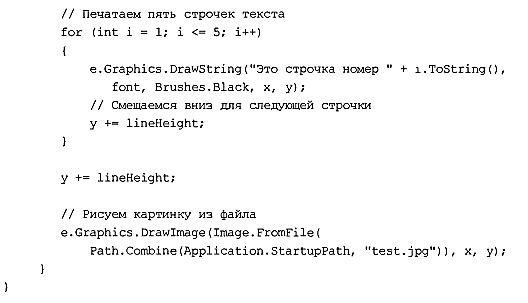
С помощью свойства installedPrintets класса Printersettings можно по­лучить список установленных в системе принтеров и далее можно получить настройки найденных принтеров.

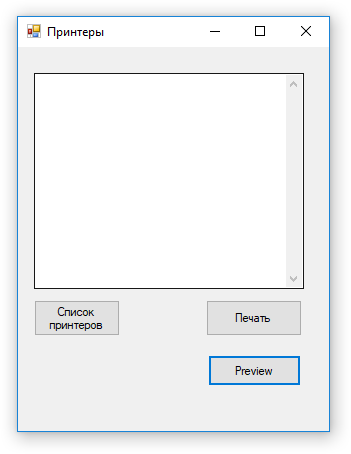


# **Как распечатать документ?**

Вам необходимо распечатать текст или картинку из вашего приложения? В этом случае вам нужно создать объект printDocument и использовать со­бытие PrintDocument.PrintPage, чтобы работать с методами Drawstring и Drawimage для печати данных, а также сконфигурировать свойства печати. После этого вы можете вызывать метод print. Настройка свойств принтера осуществляется через свойства PrintDocument.PrinterSettings и PrintDocument.DefaultPageSettings. Вы можете настраивать свойства в ко­де или через стандартное диалоговое окно печати. В этом окне пользователь может выбрать принтер и количество копий документа. Листинг содер­жит пример печати одной страницы.

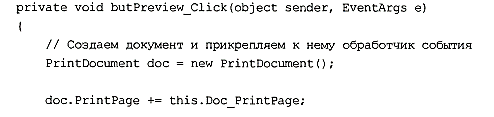


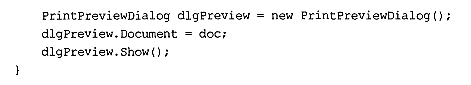




# **Как показать окно предварительного просмотра перед печатью**

Элемент управления PrintPreviewDialog является стандартным диалоговым окном, который используется для отображения компонента PrintDocument в том виде, как он будет напечатан. Данное окно используется в качестве про­стого и быстрого решения вместо диалогового окна, настраиваемого само­стоятельно. В нем имеются кнопки для печати, изменения масштаба, отобра­жения одной или нескольких страниц, а также для закрытия диалогового окна. Ключевым свойством этого элемента управления является свойство Document, задающее документ, который требуется просмотреть. Этот доку­мент должен являться объектом PrintDocument. Чтобы вывести диалоговое окно, необходимо вызвать для него метод show. Использование элемента управления PrintPreviewDialog иллюстрируется в листинге.







# **Практическая работа №11 Работа с мышью и клавиатурой**

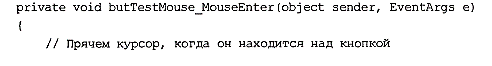
Мышь и клавиатура пока являются основными средствами ввода информа­цию в компьютер со стороны пользователя. Поэтому обойти тему работы с этими устройствами никак нельзя. Здесь вы узнаете несколько полезных со­ветов и трюков, которые, надеюсь, пригодятся в вашей практике.

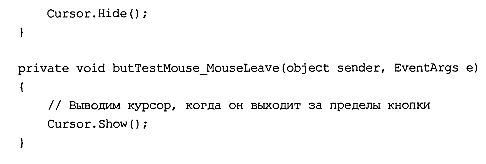
**Мышь**

В современных оконных системах мышь является основным средством вво­да, поэтому уметь с ней работать должен каждый программист.

# **Скрыть и показать указатель мыши?**

Скрыть указатель мыши с экрана очень просто. Для этого достаточно вос­пользоваться методом Cursor.Hide. Чтобы снова показать указатель мыши на экране, надо воспользоваться противоположным методом Cursor.show. Но необходимо помнить, что метод Cursor.Hide обладает накопительным свой­ством. Если вы два раза вызовете этот метод, то вам придется также два раза вызывать и метод Cursor.show, восстанавливающий указатель мыши. Пред­положим, мы хотим скрыть указатель мыши, когда он находится над кноп­кой, и показать его снова, когда он выходит за пределы кнопки. Реализация этого действия показана в листинге 8.1.

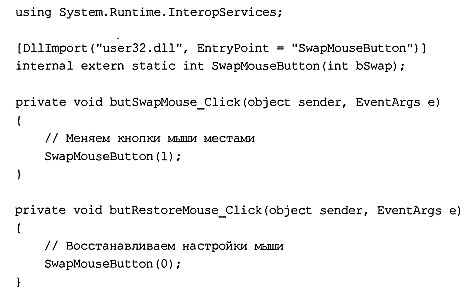




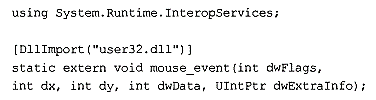
Как вы понимаете, эту возможность можно использовать при создании про­грамм-шуток, чтобы немножко попугать пользователя.

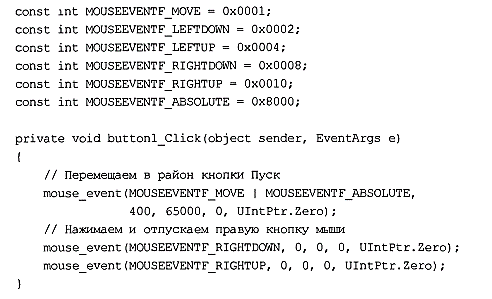
# **Меняем кнопки мыши местами**

У вашей мыши имеются, как минимум, две кнопки. Если вы правша, то ос­новная кнопка у вас левая, а правая кнопка служит для вызова контекстного меню. Но у левшей своя точка зрения на эти кнопки, и основная кнопка у них правая. Чтобы поменять настройки для кнопок мыши, пользователь должен зайти в Панель управления и в апплете **Мышь** установить желаемые на­стройки. Но вы можете программно изменить эти настройки (листинг) через вызов функции Windows API swapMouseButton.



Чтобы убедиться, что кнопки мыши действительно поменялись, я располо­жил на форме текстовое поле, у которого по умолчанию имеется контекстное меню. Запустите проект и нажмите на кнопку с надписью **Поменять места­ми.** Теперь щелкните правой кнопкой на текстовом поле. Ничего не происхо­дит? Так ведь правая кнопка теперь работает у вас как левая. Теперь щелкни­те на текстовом поле левой кнопкой — у вас появится контекстное меню. Замена кнопок местами носит глобальный характер и действует на все приложения. Если хотите вернуть старые настройки, то нажмите на кнопку Вос­становить (догадайтесь, какой кнопкой надо сделать щелчок).





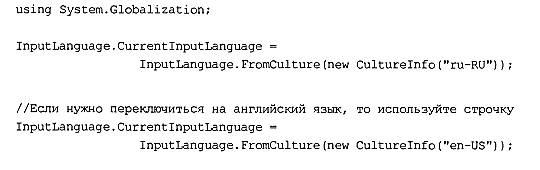
Необходимо помнить, что кнопка **Пуск** у пользователя может находиться не только в нижнем левом углу, но и в других позициях.

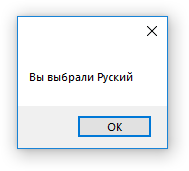
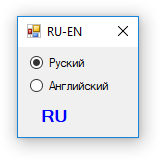
# **Работа с клавиатурой**

Клавиатура является одной из важных составляющих компьютера. Если вы отключите клавиатуру и попробуете включить машину, то появится сообще­ние об ошибке и система откажется загружаться. Клавиатура позволяет вво­дить информацию в компьютер с помощью клавиш. Несмотря на разнообра­зие моделей и количества клавиш на них. у всех клавиатур имеется определенный стандарт клавиш. Прежде всего, это ряд клавиш с буквами и цифрами, клавиши-модификаторы, функциональные клавиши и цифровой блок. Также у клавиатур имеются индикаторы со светодиодами, которые мо­гут загораться и выключаться при нажатии некоторых клавиш.

# **Переключение раскладки клавиатуры программно.**

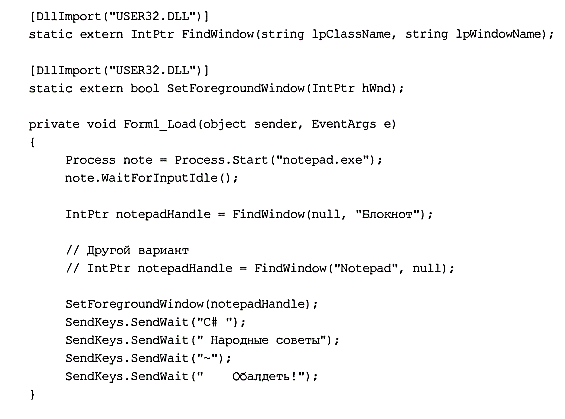
В отличие от американцев, нам приходится использовать две раскладки кла­виатуры, а то и больше. Таким образом, у вас периодически возникает по­требность переключении раскладки. Пользователь может изменить раскладку вручную, нажав комбинацию клавиш <All>+<Shift> (или другую комбина­цию, определенную настройками системы). В последнее время появились программы, которые автоматически переключают раскладку, пытаясь опре­делить нужный язык по вводимым словам. Но если эти программы способны программно переключать раскладки, то почему бы и нам не сделать то же самое. Для этого нужно воспользоваться свойством CurrentlnputLanguage класса System. Windows. Forms. InputLanguage. В листинге приведен код, переключающий язык ввода на русский.





# **Посылка нажатия клавиш программно**

С помощью класса SendKeys, который находится в пространстве имен System.windows.Forms, можно программно эмулировать нажатие определен­ной комбинации клавиш активному приложению, используя его метод SendWait. Так как не существует управляемых методов для активирования других приложений, то для передачи фокуса другому приложению восполь­зуемся функциями Windows API Findwindow и setForegroundWindow. Связка этих функций с классом SendKeys позволит нам, например, запустить про­грамму Блокнот и напечатать несколько слов (листинг).

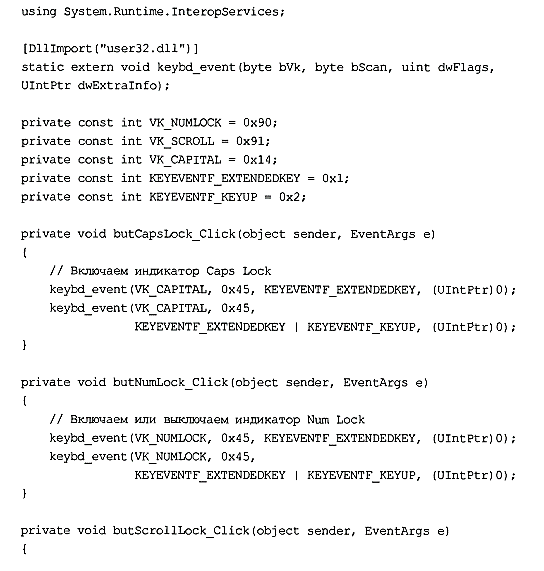


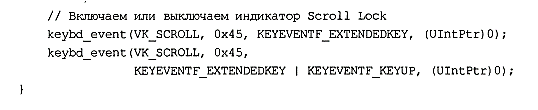
При запуске программы автоматически запустится новая копия программы Блокнот, в которой будут напечатаны нужные слова.

**Включаем и выключаем индикаторы клавиш <Caps Lock>, <Num Lock> и <Scroll Lock>**

На большинстве клавиатур имеются светодиоды-индикаторы, которые по­зволяют пользователю отслеживать состояние клавиш <Caps Lock>,

<Num Lock> и <Scroll Lock>. Например, если горит индикатор Caps Lock, значит, клавиша <Caps Lock> активирована, и все символы, вводимые в тек­стовом поле, будут выводиться в верхнем регистре (если при этом удержи­вать клавишу <Shift>, то символы будут выводиться в нижнем регистре). Мы можем программно изменить состояние этих клавиш и устроить небольшую светомузыку, включая и выключая индикаторы в такт какой-нибудь мелодии. Для реализации этой идеи нам понадобится вызов функций Windows API keybd\_event, как показано в листинге.

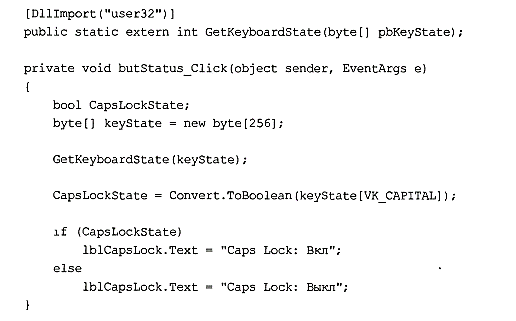




Как видите, для включения какой-нибудь клавиши с индикатором нужно дважды вызвать функцию keybd\_event. Повторный двойной вызов этой функции выключает индикатор, поэтому нет необходимости писать какой-либо дополнительный код для этого.

# **Определение состояния клавиш-индикаторов**

У вас может возникнуть вопрос, а как узнать текущее состояние этих кла­виш-индикаторов перед их включением или выключением. В .NET Frame­work 1.1 приходилось использовать неуправляемый код (листинг)

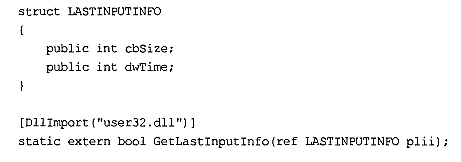


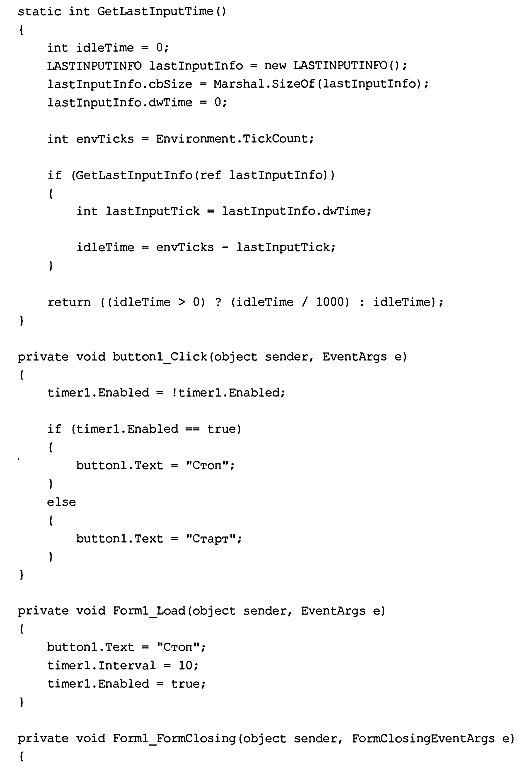
С помощью данного кода можно было узнать состояние клавиши <Caps Lock>. Но с появлением .NET Framework 2.0 задача упростилась. В классе Control появился новый метод isKeyLocked, который определяет состояние трех кла­виш <Caps Lock>, <Num Lock> и <Scroll Lock>. Работа с ним проиллюстри­рована в листинге.

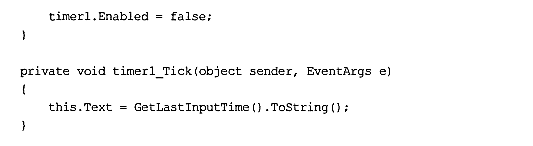


# **Последнее нажатие на клавишу или на кнопку мыши**

Наверное, вы замечали, что ICQ и ей подобные программы могут определять активность пользователя. Если пользователь в течение определенного време­ни не прикасался к клавишам клавиатуры или не трогал мышь, то значок программы меняется на состояние Отошел. И его собеседник, видя это со­стояние, уже не будет понапрасну посылать сообщения. Для реализации этой задачи очень удобно использовать функцию Windows API GetLastinputinfo, которая возвращает время, прошедшее с последнего нажатия на клавиатуру или мышь. Разместим на форме кнопку и таймер. Все свойства элементов будут определены программно, поэтому оставляем все значения свойств по умолчанию (листинг).



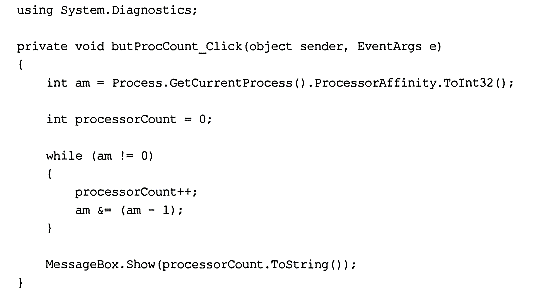




Запустите программу и не предпринимайте никаких действий. Вы увидите в заголовке формы счетчик времени, показывающий бездействие пользователя. Стоит вам пошевелить мышкой или нажать на любую клавишу, как счетчик сбросит свои показания и начнет новый отсчет.

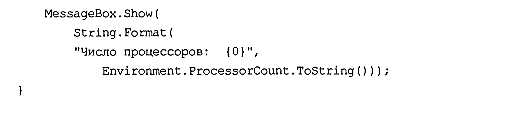
# **Практическая работа №12 Как узнать число процессоров в системе?**

Вы можете узнать число процессоров в системе при помощи свойства ProcessorACf inity, как показано в листинге.



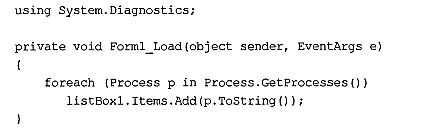
Впрочем, этот код устарел для .NET Framework 2.0, так как число процессо­ров можно узнать с помощью одной строчки кода (листинг) с использо­ванием свойства Environment. ProcessorCount.

private void butProcCount2\_Click(object sender, EventArgs e)



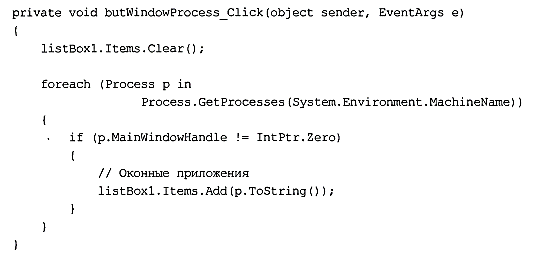
# **Получаем список всех процессов, запущенных в системе**

Класс Process ИЗ пространства имен System. Diagnostics позволяет узнать множество информации о процессах. Например, с помощью метода process.GetProcesses не составит труда получить список всех запушенных процессов на локальной машине, как показано в листинге.



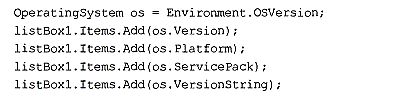
Как получить список только оконных процессов на ПК.

Можно получить список только оконных процессов, если проверять свойство MainwindowHandle (листинг). Мы также передаем методу GetProcesses имя компьютера, в этом случае равное значению свойства System. Environment.MachineName. Результат показан на рис. 9.1.



# **Практическая работа №13 Определение операционной системы пользователя**

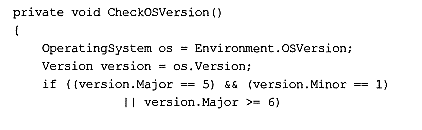
Для определения операционной системы нужно использовать свойство OSVersion класса Environment из пространства имен System. Свойство Version имеет ПОЛЯ Major и Minor, которые содержат дополнительную ин­формацию. Операционная система Windows XP имеет версию 5.1 и сборку 2600, а новая Windows Vista имеет версию 6.0 и сборку 6600. В статье "How То Determine the Operating System Service Pack Level in Visual *CH* .NET" (<http://support.microsoft.com/kb/304721/EN-US/>) приводится пример полу­чения номера сервис-пака с помощью функции Windows API GetVersionEx. Данный пример был предназначен для использования в .NET Framework I.I, но с появлением .NET Framework 2.0 надобность в использовании неуправ­ляемого кода отпала. В составе новой платформы .NET Framework 2.0 появи­лись такие новые свойства, как servicePack и versionstring класса operatingSystem, которые позволяют быстро получить необходимые данные (листинг).

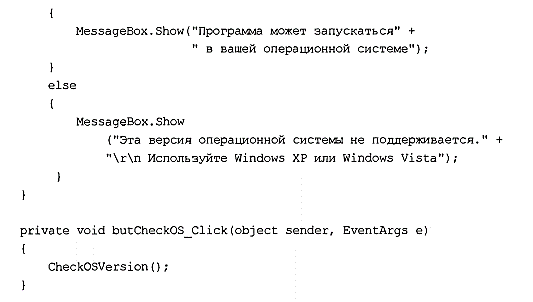


Особое внимание нужно обратить тем программистам, которые писали про­граммы специально для Windows XP. После выхода этой операционной сис­темы многие разработчики в своих проектах делали проверку на наличие этой системы у пользователя, чтобы задействовать предоставляемые ей но­вые возможности. Проверка могла осуществляться такой строчкой:

if ((os.Version-Major = 5) ££ {os.Version.Minor =-1)) //и так далее

В данном примере проверяется наличие операционной системы Windows XP. Если вы запустите проект с подобной проверкой под Windows Vista, то про­грамма будет считать, что Windows Vista не входит в число поддерживаемых операционных систем. Поэтому вам нужно переделать эту проверку, как по­казано в листинге.



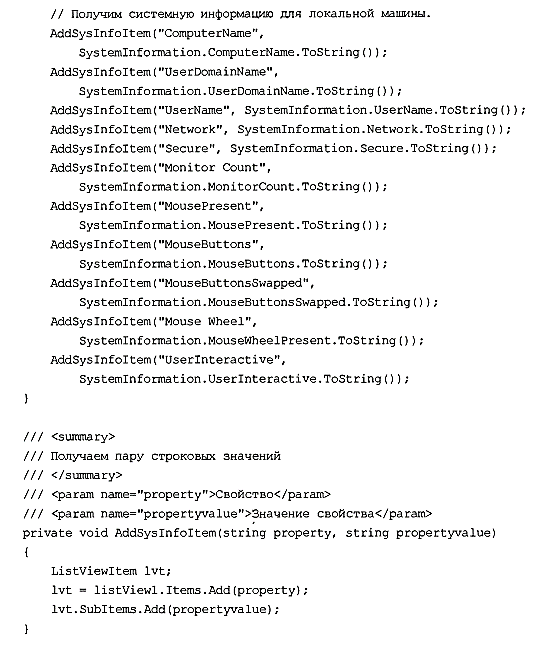


# **Практическая работа №14**

# **Определяем подключение мышки к системе, узнать число кнопок у мыши, размер вашего монитора и другую информацию?**

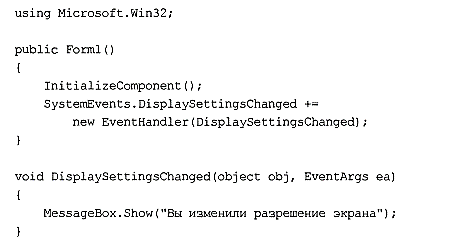
На самом деле класс System.Windows.Forms.Systemlnformation, который мы использовали в предыдущем примере, имеет множество других полезных методов и свойств. Например, мы можем узнать, используется ли в системе мышь, сколько у нее кнопок, есть ли колесико, менялись ли местами кнопки у мыши и многое другое. В листинге приведен небольшой кусочек кода, который извлекает полезную системную информацию.

В автозагрузку формы добавьте следующий код:

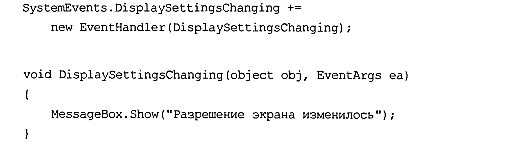


# **Как узнать, что пользователь изменил разрешение экрана?**

Класс SystemEvencs из пространства имен Win32 имеет в своем составе собы­тие DisplaySettingsChanged, с помощью которого можно узнать, что поль­зователь изменил разрешение экрана (листинг).

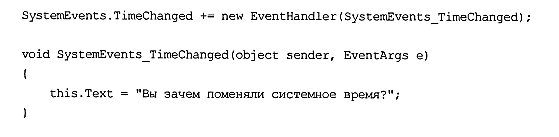


В составе .NET Framework 2.0 появилось новое дополнительное событие DisplaySettingsChanging, которое возникает перед изменением разрешения экрана. Его использование показано в листинге 9.33.



# **Практическая работа №15 Изменение времени**

Рассмотрим еще одно событие TimeChanged (листинг), которое возника­ет при смене системного времени через приложение Панели управления **Дата и Время.**



Когда вы поменяете время в Панели управления и нажмете на кнопку При­менить, то программа получит сообщение через событие TimeChanged.

Остальные системные события изучите самостоятельно.

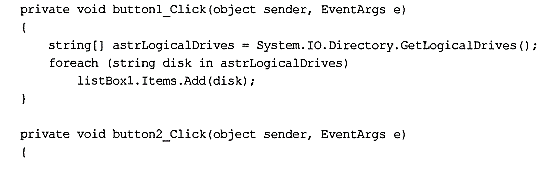
Также обратите внимание, что к событиям SessionEnded и SessionEnding в .NET Framework 2.0 добавилось новое событие sessionswitch, которое про­исходит при переключении на другую учетную запись системы.

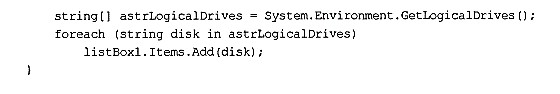
# **Практическая работа №16 Диски**

Диски являются вершиной в иерархии файловой системы, и именно в них содержатся файлы и панки. Рассмотрим наиболее востребованные програм­мистами приемы извлечения нужной информации.

Как получить список логических дисков?

В .NET Framework имеются два метода GetLogicalDrives, которые возвра­щают массив имен логических дисков. Один метод относится к классу Directory, а второй— к классу Environment. В листинге 10.1 рассматрива­ются оба варианта.

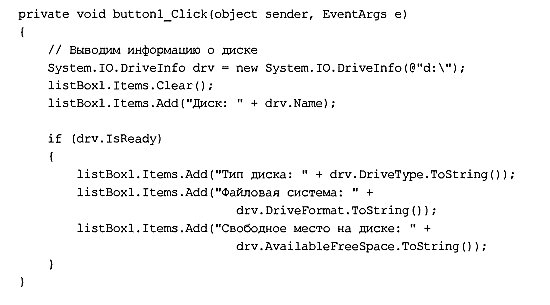




Разница между этими одноименными методами в том, что они вызывают разные типы исключений, связанные с правами доступа и разрешениями.

# **Как узнать тип диска и его свойства?**

Несмотря на растущие объемы жестких дисков, наступает момент, когда пользователь начинает поглядывать на размер свободного места на своих дисках. Фотографии, фильмы, музыка, "навороченные" программы быстро заполняют диски. Чтобы узнать количество свободного места на диске, мож­но воспользоваться СВОЙСТВОМ Drivelnfo.AvailableFreeSpace. Обратите внимание, что класс Drivernfo является новым классом в .NET Framework 2.0. Поэтому вам, возможно, придется пересмотреть старые про­екты, в которых вы получали информацию о дисках при помощи неуправ­ляемого кода. Для создания нового объекта Driveinfo нужно указать букву диска или более привычную запись с двоеточием и обратной косой чертой, например, "С" или "С:\" для диска С:. В листинге приведен пример, по­казывающий, как извлечь информацию о нужном диске.



Надо отметить, что у класса Drivelnfo есть еще свойство TotalFreeSpace, изучить которое необходимо самостоятельно. Обратите внимание, что свой­ство AvailableFreeSpace заменило функцию Windows API Get-FreeDiskSpaceEx.

**Практическая работа №17 Локальная сеть и Интернет**

Интернет-технологии по-прежнему являются очень динамично развиваю­щимся направлением в компьютерной индустрии. Появляются новые прото­колы, развиваются браузеры, появляются новые способы взаимодействия пользователей. Также развиваются местные локальные сети внутри предпри­ятий.

Поэтому сетевое программирование является очень важной частью в работе программиста. Знание некоторых приемов программирования сетей всегда пригодится любому разработчику.

**Информация о сети**

Сначала рассмотрим некоторые вопросы, связанные с определением конфи­гурации сети, адресов компьютера и т. п.

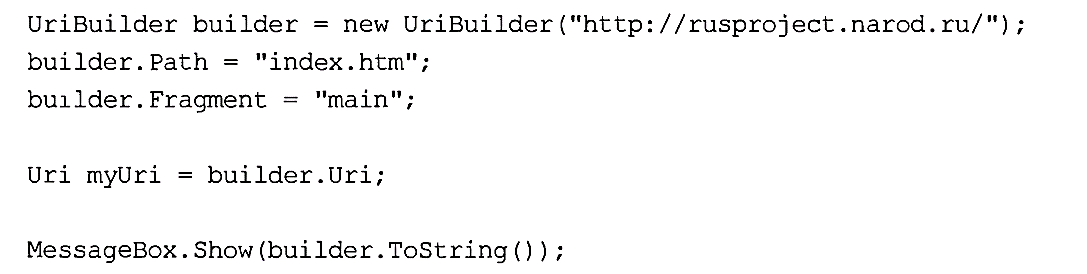
**Получаем хост, порт, протокол из веб-адреса.**

С помощью класса **System.uriBuilder** можно легко получить необходимую информацию о хосте, порте, используемом протоколе и другие данные. При­мер показан в листинге





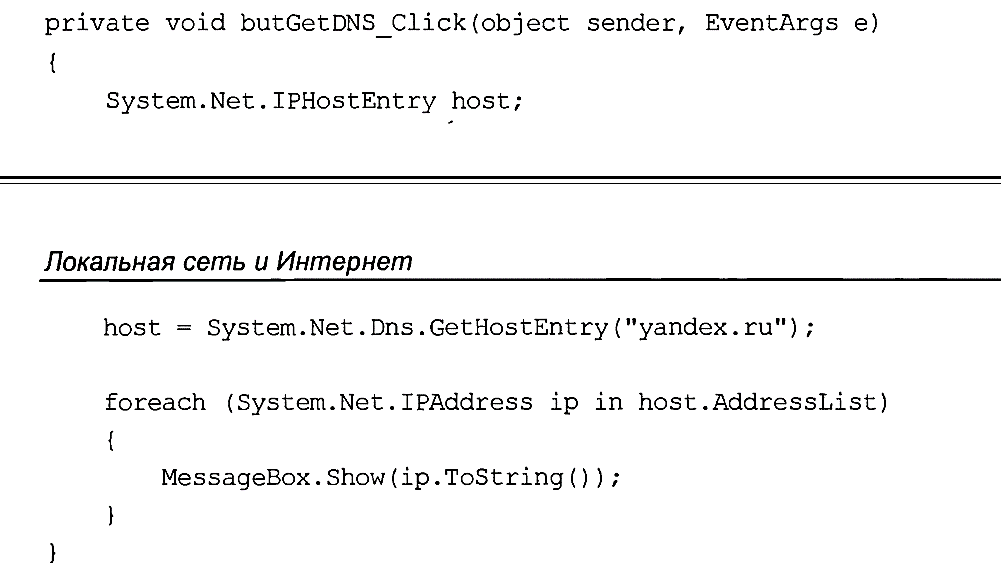
С помощью данного класса также можно создавать URI (универсальный идентификатор ресурсов) из составляющих элементов. Например, имеется адрес сайта, имя файла и фрагмент. Свойство Fragment содержит любой текст, который следует за маркером фрагмента (знаком #) в URI, включая сам маркер. При установлении свойства Fragment данный маркер также добавляется к его значению. В листинге показано создание URI [**http://rusproject.narod.ru/index**](http://rusproject.narod.ru/index)**. htm#main.**

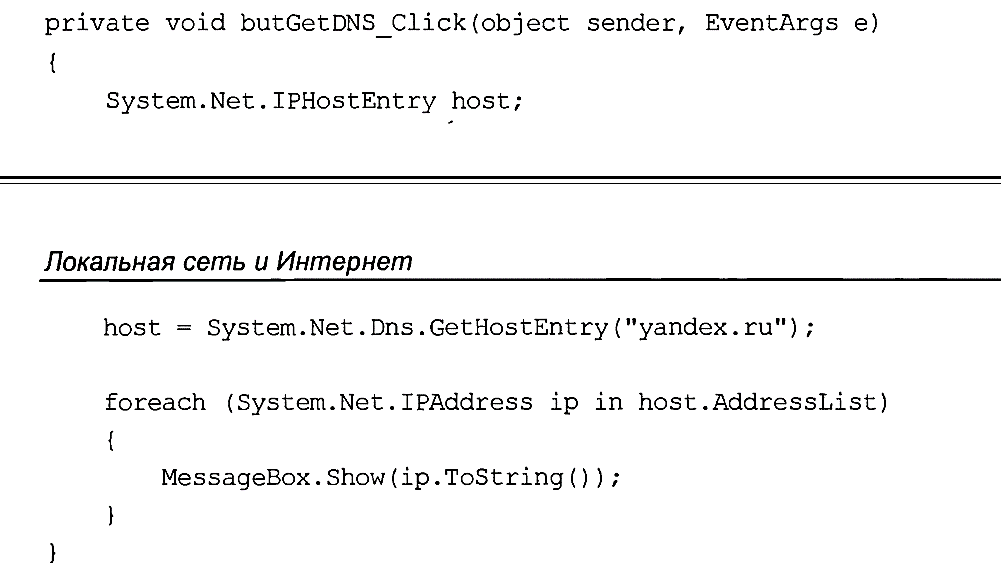


|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  | | |

**Получаем IP-адрес компьютера, используя DNS**

С помощью метода GetHostEntry класса Net.Dns можно получить IP-адрес компьютера, используя его имя, которые мы обычно вводим в браузер. При­мер показан в листинге





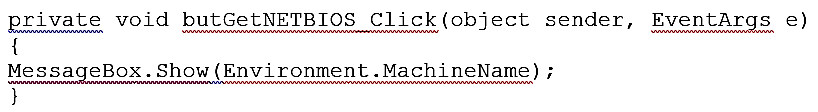
В результате выполнения кода мы получим список IP-адресов указанного сайта. Теперь попробуйте ввести эти адреса непосредственно в браузер, что­бы убедиться в работоспособности кода. Также с помощью этого метода можно выполнять и обратное преобразование. Обратите внимание, что этот метод введен в .NET Framework 2.0, где он дополняет и заменяет устаревший метод GetHostByName из библиотеки классов .NET Framework 1.1.

**Получение NETBIOS-имя машины**

Для получения имени NETBIOS локальной машины используется свойство

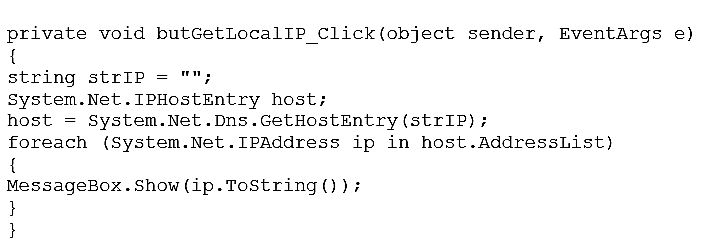
Envinronment.MachineName

Получение NETBIOS-имени компьютера



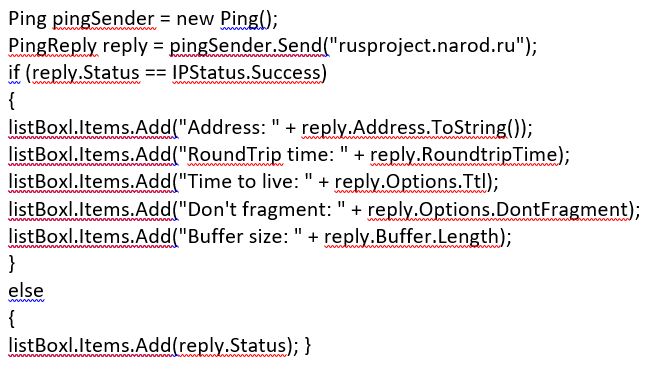
**Получаем IP-адрес локальной машины**

Для получения IP-адреса локальной машины достаточно указать пустую строку при использовании метода GetHostEntry (листинг).



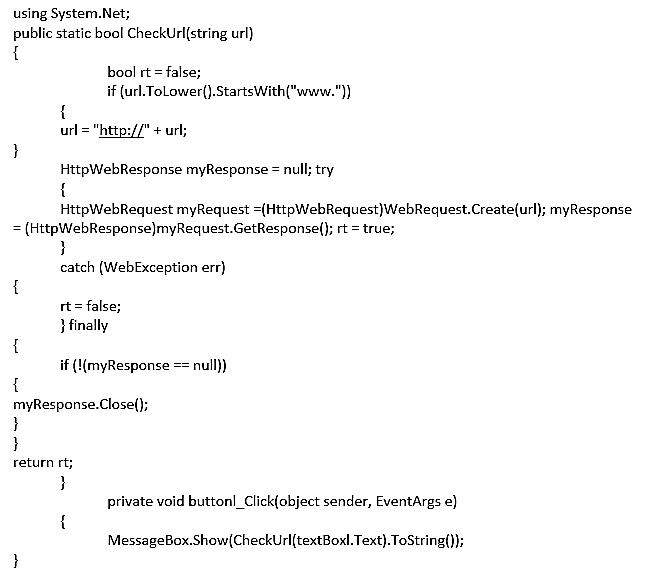
**Ping**

В состав операционной системы Windows входит утилита командной строки ping.exe, позволяющая определить доступность компьютера в сети. Для реа­лизации подобной функциональности в .NET Framework 1.1 требовалось пи­сать сложный код с использованием вызовов функций Windows API. В .NET Framework 2.0 появился новый класс Ping из пространства имен System.Net .Networkinformation, с помощью которого реализация возмож­ностей, предоставляемых утилитой ping.exe, стала очень простым занятием. В справочном материале, поставляемом с Visual Studio, приводится много примеров использования этого класса. В листинге приведен про­стой пример, который даст вам представление, как вы можете использовать данный класс в своем приложении.



**Проверка доступности веб-адреса**

Если нужно определить, доступна ли указанная ссылка в Интернете, можно написать небольшую вспомогательную функцию checkURL, приведенную в листинге

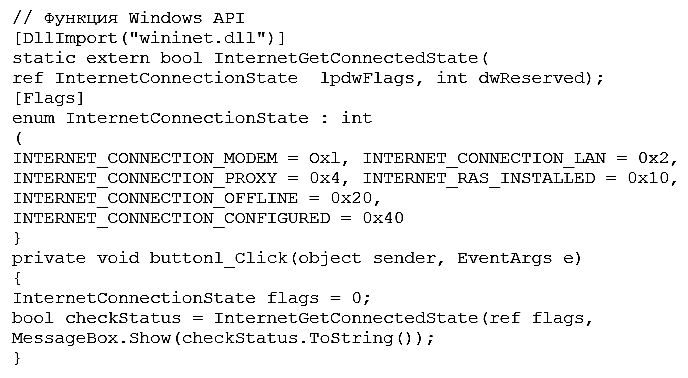
****

Теперь вам достаточно ввести в текстовое поле какой-нибудь адрес и убе­диться, что данный адрес доступен в сети. Можно использовать как название сайта (например, [www.stream.ru](http://www.stream.ru)), так и указывать конкретный файл на сайте (например, [www.whatis.ru/index.shtml](http://www.whatis.ru/index.shtml)). В данном примере используется префикс www для проверки, что адрес является веб-сайтом. Можно сделать функцию более универсальной, чтобы она позволяла бы проверять доступ­ность ссылки, не имеющей подобного префикса.

****

**Проверка подключения компьютера к Интернету**

Чтобы проверить наличие подключения компьютера к Интернету, можно воспользоваться функцией **Windows API internetGetConnectedState**, при­менение которой проиллюстрировано в листинге



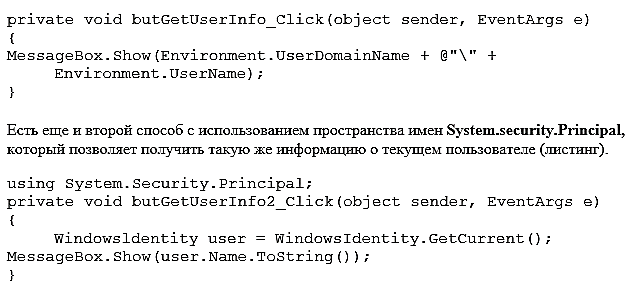
**Работа с локальной сетью**

Не менее важным аспектом, чем программирование сети Интернет, является программирование локальной сети. Мы рассмотрим несколько приемов.

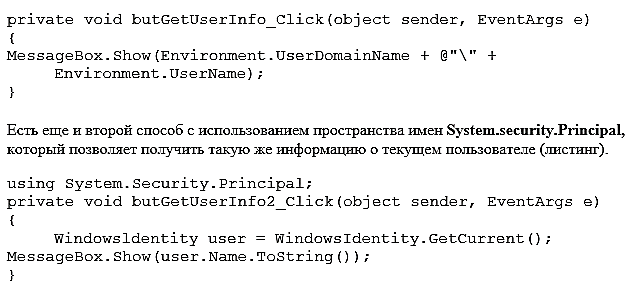
**Получаем имя текущего пользователя**

Класс System.Environment содержит свойства UserDomainName и UserName, С

помощью которых можно получить информацию о текущем пользователе системы. Свойство UseDomainName возвращает имя сетевого домена, к кото­рому подключен текущий пользователь, а свойство UserName — имя находя­щегося в системе пользователя, который запустил текущий поток команд. Комбинация из этих двух свойств позволят нам идентифицировать пользова­теля (листинг).

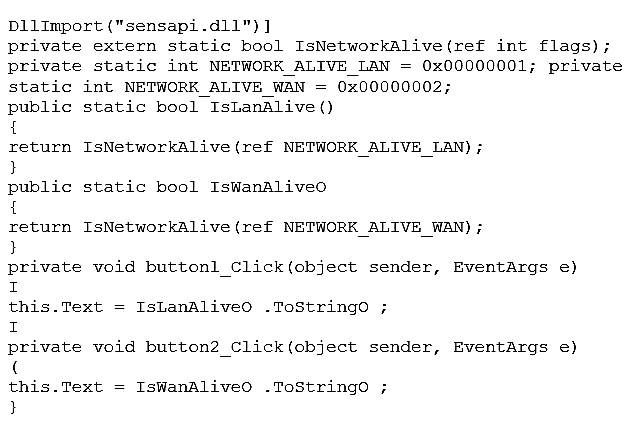


Есть еще и второй способ с использованием пространства имен **System.security.Principal,** который позволяет получить такую же инфор­мацию о текущем пользователе (листинг).



**Выясняем, подключена ли локальная система к сети, и узнать используемый тип соединения**

Сетевая функция Windows API isNetworkAiive определяет, подключена ли локальная система к сети, а также возвращает информацию о типе соедине­ния (LAN, WAN). Пример ее использования показан в листинге.



**Практическая работа 18.  
Функции Windows API**

Программный код, который выполняется под управлением CLR (Common Language Runtime, то есть общая среда выполнения языков), называется управляемым (managed) кодом. Программный код, выполняющийся вне сре­ды выполнения CLR, называется неуправляемым (unmanaged) кодом. Приме­ром неуправляемого программного кода служат функции Win32 API, компо­ненты СОМ, интерфейсы ActiveX. Несмотря на большое количество классов .NET Framework, содержащих множество методов, программисту все равно приходится иногда прибегать к неуправляемому коду. Надо сказать, что чис­ло вызовов неуправляемого кода уменьшается с выходом каждой повой вер­сии .NET Framework. Microsoft надеется, что наступит такое время, когда весь код можно будет сделать управляемым и безопасным. Но пока реаль­ность такова, что без вызовов функций Windows API не обойтись, что вы могли наблюдать на примерах из предыдущих глав. Здесь мы приведем еще несколько полезных приемов с использованием функций Windows API, кото­рые не попали в предыдущие главы. Но сначала немного теории.

Управляемый код .NET Framework может вызывать неуправляемую функцию из DLL (функцию Windows API) при помощи специального механизма Plat­form Invoke (сокр. P/lnvoke). Для того чтобы обратиться к какой-нибудь не­управляемой библиотеке DLL, вы должны преобразовать .NET-объекты в наборы struct, char\* и указателей на функции, как того требует язык С. Как сказали бы программисты на своем жаргоне— вам нужно «маршалировать» параметры. Более подробно о маршалинге (Marshalling) вам следует почитать в документации. Чтобы вызвать DLL-функцию из С#, сначала ее необходимо объявить (программисты, имеющие опыт работы с Visual Basic 6.0, уже зна­комы с этим способом). Для этого используется атрибут Dllimport, напри­мер, как показано в листинге

using System.Runtime.InteropServices;

public class Win32

{

[DllImport(“User32.Dll”)]

public static extern void SetWindowText(IntPtr hwnd, String IpString);

}

Атрибут Dllimport сообщает компилятору, где находится точка входа, что позволяет далее вызывать функцию из нужного места. Вы должны всегда использовать тип IntPtr для hwnd, hmenu и любых других описателей. Для lpctstr используйте string, а сервисы взаимодействия (interop services) вы­полнят автоматический маршалинг System.string в lpctstr до передачи в Windows. Компилятор ищет указанную выше функцию SetwindowText в фай­ле User32.dll и перед ее вызовом автоматически преобразует вашу строку в lptstr (tchar\*). Почему это происходит? Для каждого типа в С# определен свой тип, используемый при маршалинге по умолчанию (default marshaling type). Для строк это lptstr.

Часть примеров, приводимых ниже, были взяты из разных номеров журнала "MSDN Magazine".

**Является ли текущий пользователь администратором?**

Если необходимо удостовериться, что текущий пользователь имеет права ад­министратора, то можно вызвать функцию isUserAnAdmin, как показано в листинге.

[DllImport("shell32.dll")]

public static extern bool IsUserAnAdmin();

private void butIsAdmin\_Click(object sender, EventArgs e)

{

MessageBox. Show (IsUserAnAdmin () .ToStringO );

}

**Форматирование дисков**

Чтобы вызвать стандартное диалоговое окно форматирования дисков, нужно воспользоваться функцией SHFormatDnve. Пример приведен в листин­ге.

[DllImport(,,shell32.dll")]

static extern uint SHFormatDrivedntPtr hwnd,

uint drive, uint fmtID, uint options);

private void butFormatDiskDial\_Click(object sender, EventArgs e) I

SHFormatDrivetthis.Handle, 3, 0, 0);

}

**Открытие и закрытие лотка привода компакт-дисков**

Наверное, при работе с утилитами, прожигающими компакт-диски CD-R и CD-RW. вы замечали, что у них имеется возможность извлекать компакт-диск из привода программным путем. Неплохо бы научиться делать то же самое при помощи С#. Для этого используем функцию mciSendstring в связ­ке со специальными командами, которые и позволят нам открывать и закры­вать лоток привода компакт-дисков (листинг).

// Функция для открытия и закрытия лотка привода CD

[Dlllmport("wimnm.dir', EntryPoint = "mciSendStringA",

CharSet = CharSet.Ansi)]

public static extern int mciSendString(string strCommand,

StringBuilder strReturnString,

int cchReturn, IntPtr hwndCallback);

private void butOpenCD\_Click(object sender, EventArgs e)

*I*

*II* Открываем лоток

mciSendString("set CDAudio door open", null, 0, IntPtr.Zero); I

private void butCloseCD\_Click(object sender, EventArgs e) (

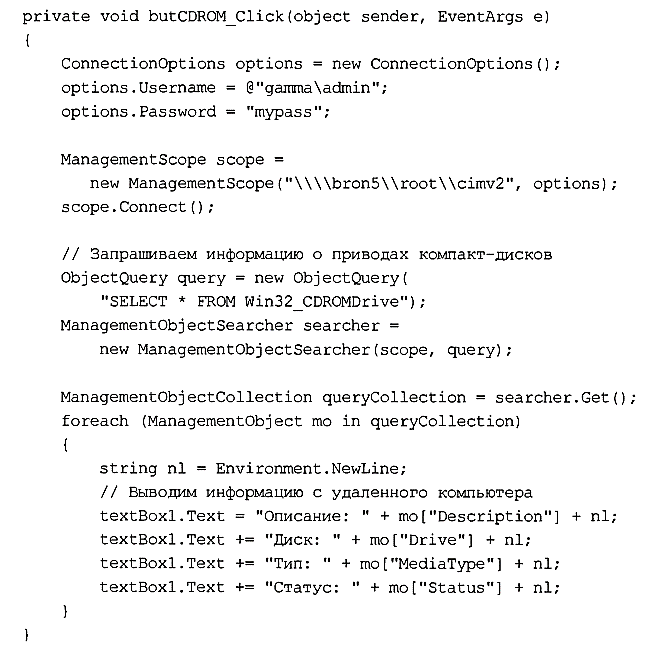
// Закрываем лоток

mciSendString("set CDAudio door closed", null, 0, IntPtr.Zero);

}

**Получение свойств приводов компакт-дисков**

Теперь мы хотим узнать немного о приводах компакт-дисков. Для этого су­ществует класс win32\_CDROMDrive. Его применение показано в листинге.



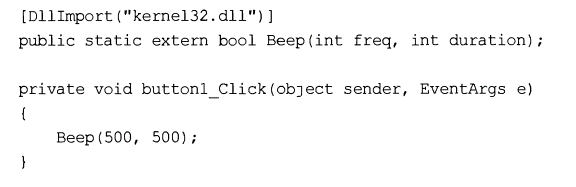
**Мультимедиа**

**Звуковые сигналы**

Звуковые сигналы— один из способов привлечь внимание пользователя к проблеме. Но надо помнить, что у пользователя могут быть отключены ди­намики. Однако есть еще системный динамик, который издает простейшие гудки, и который обычно подключен. Давайте рассмотрим все способы.

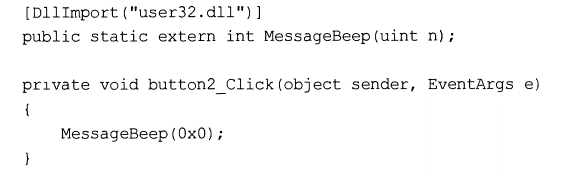
**Функция Windows API *Beep***

Функция Windows API Beep воспроизводит тоновый звук через внутренний динамик с различной частотой. Вы можете задавать частоту (в герцах) и про­должительность звука (в миллисекундах) для получения различных звуков, как показано в листинге 13.1.



**Функция Windows API MessageBeep**

Существует еще одна функция Windows API MessageBeep, которая воспроиз­водит один из системных звуков (листинг). Системные звуки можно определить при помощи апплета Звук в Панели управления. Функция также может воспроизводить звук встроенного динамика компьютера.



**Диски, папки и файлы**

**Диски**

Диски являются вершиной в иерархии файловой системы, и именно в них содержатся файлы и папки. Рассмотрим наиболее востребованные програм­мистами приемы извлечения нужной информации.

**Как получить список логических дисков?**

В .NET Framework имеются два метода GetLogicalDrives, которые возвра­щают массив имен логических дисков. Один метод относится к классу Directory, а второй— к классу Environment. В листинге рассматрива­ются оба варианта.

//Вариант 1

private void buttonl\_Click(object sender, EventArgs e)

{

string[] astrLogicalDrives = System.I0.Directory.GetLogicalDrives ();

foreach (string disk in astrLogicalDrives)

listBoxl.Items.Add(disk);

}

//Вариант2

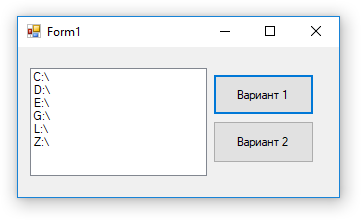
private void button2\_Click(object sender, EventArgs e)

{

string[] astrLogicalDrives = System.Environment.GetLogicalDrives();

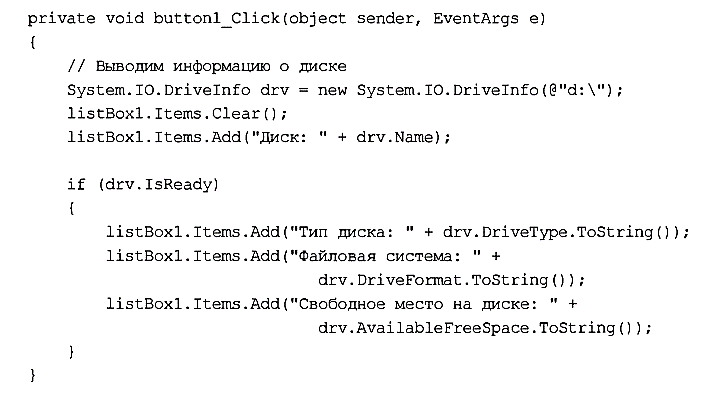
foreach (string disk in astrLogicalDrives) listBoxl.Items.Add(disk);

}

****

**Получаем информацию о диске и его свойствах.**

Несмотря на растущие объемы жестких дисков, наступает момент, когда пользователь начинает поглядывать на размер свободного места на своих дисках. Фотографии, фильмы, музыка, "навороченные" программы быстро заполняют диски. Чтобы узнать количество свободного места на диске, мож­но воспользоваться свойством Drivelnfo.AvailableFreeSpace. Обратите внимание, что класс Driveinfo является новым классом в .NET Framework 2.0. Поэтому вам, возможно, придется пересмотреть старые про­екты, в которых вы получали информацию о дисках при помощи неуправ­ляемого кода. Для создания нового объекта Driveinfo нужно указать букву диска или более привычную запись с двоеточием и обратной косой чертой, например, "С" или "С: V\* для диска С:. В листинге 10.2 приведен пример, по­казывающий, как извлечь информацию о нужном диске.

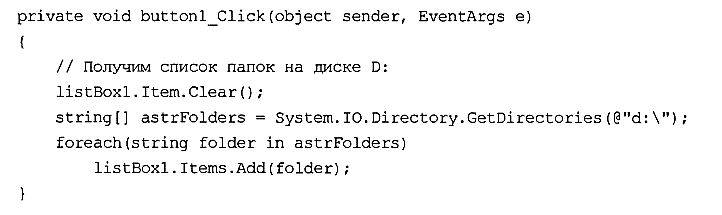


Надо отметить, что у класса Driveinfo есть еще свойство TotalFreeSpace, изучить которое я предлагаю самостоятельно. Обратите внимание, что свой­ство AvailableFreeSpace заменило функцию Windows API Get-FreeDiskSpaceEx.

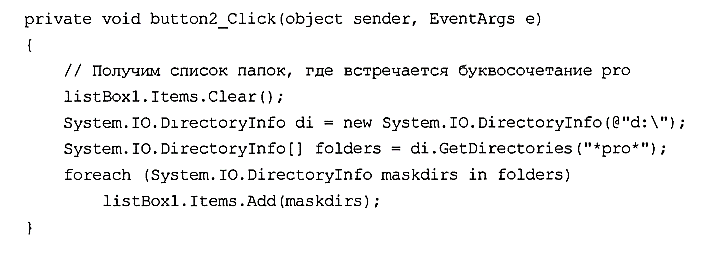
**Получаем список папок.**

Получить список папок можно также несколькими способами. Например, можно воспользоваться методом GetDirectories класса

System. 10. Directory. Как это сделать, показано в листинге



Также метод GetDirectories имеется в классе Directorylnfo. Рассмотрим и этот вариант, и при этом выведем список папок по маске (листинг). На­пример, мы хотим получить список только тех папок, в именах которых встречается буквосочетание "pro".



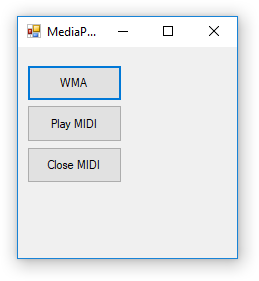
**Практическая работа № 19 Обработка звука**

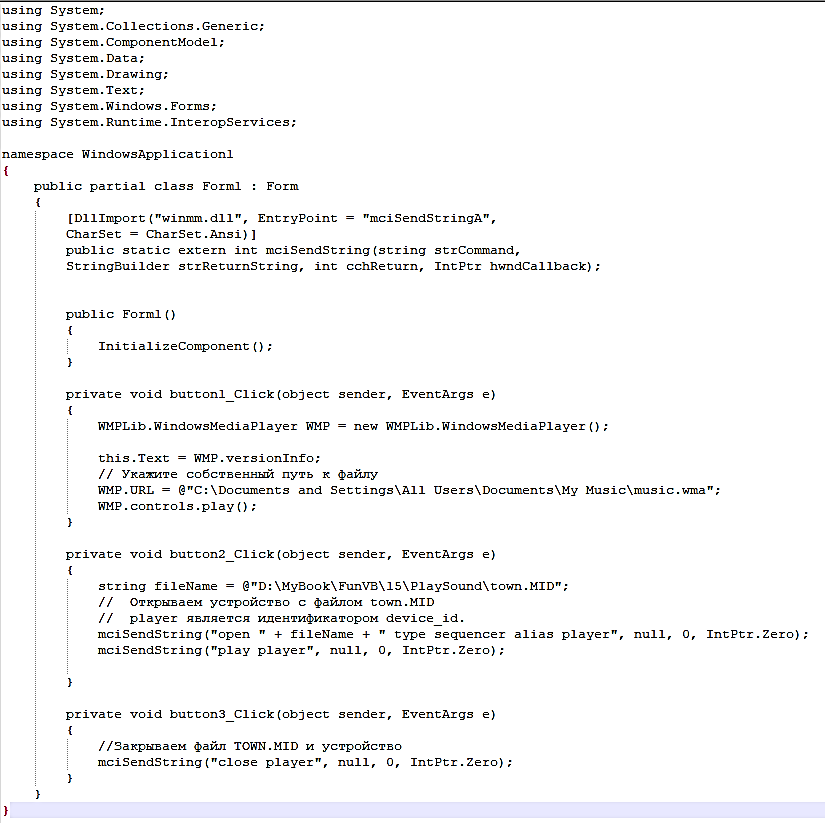
Мы рассмотрим способ воспроизведения звукового файла формата WAV при помощи класса SoundPlayer. Но существует масса других звуковых форматов, которые используются для прослушивания музыкальных треков. Среди распространенных музыкальных форматов лидерами являются форма­ты МРЗ и WMA. Файлы этих форматов с помощью встроенных методов уже не проиграешь. Тем не менее, выход есть. Во-первых, можно воспользовать­ся помощью универсального проигрывателя Windows Media.

Если подклю­чить библиотеки данного проигрывателя, то можно воспроизводить файлы любого формата, который поддерживается этим проигрывателем.

**Чтобы подключить библиотеку Windows Media**, выберите меню **Project | Add Ref­erence... и** в открывшемся окне на вкладке Browse найдите файл **wmp.dll**, который находится в папке Windows\System32. После подключения этой библиотеки вам станут доступны многие свойства этого проигрывателя. Для примера в листинге показан простой код для воспроизведения файла в формате WMA.

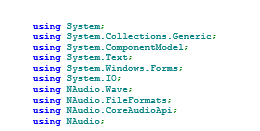
Нельзя забывать, что пользователь может отказаться от установки стандарт­ного проигрывателя Windows Media и установить какой-нибудь альтернатив­ный музыкальный проигрыватель. И нет никакой гарантии, что необходимая библиотека wmp.dll окажется на его компьютере. В этом случае нам опять поможет вызов неуправляемой функции Windows API mcisendstring.

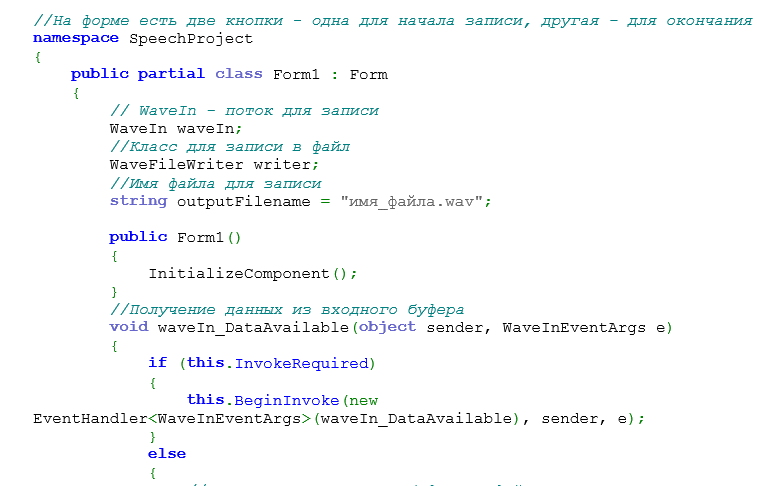
****

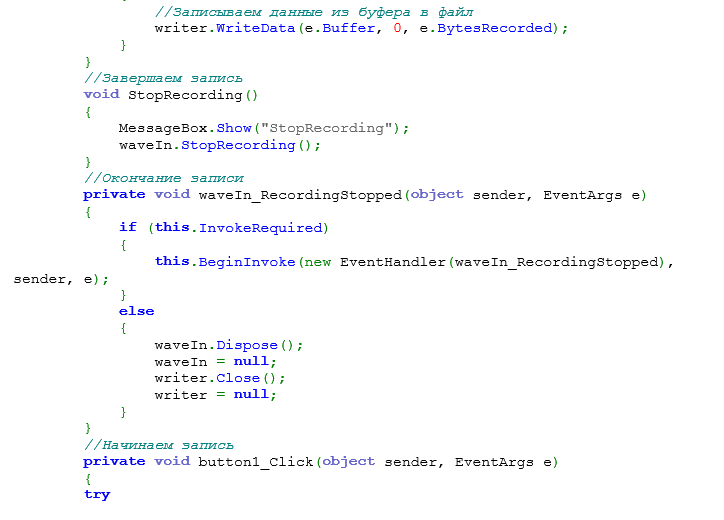
****

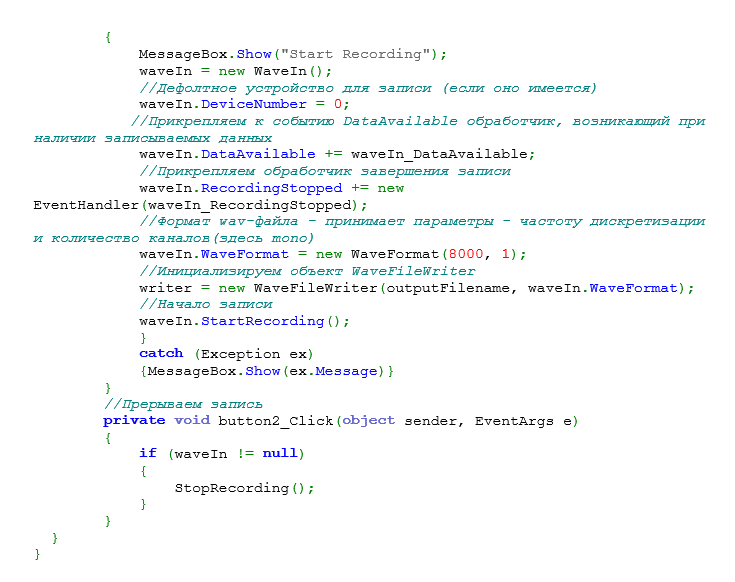
**Запись файла с микрофона**

Для записи звука воспользуемся библиотекой [NAudio](https://naudio.codeplex.com/). После загрузки библиотеки добавим на нее ссылку в проект, чтобы видны были пространства имен в **NAudio**.









# **Вызов файла справки СНМ**

**В пространстве имен Syscem.Windows. Forms** имеется класс **Help,** которыйпозволяем выводить справку, подготовленную при помощи утилиты **IITMLHelp Workshop (файл СНМ).**

Как это сделать, показано в листинге.

