Виктор Зиборов

Visual C# 2010
НА ПРИМЕРАХ

Санкт-Петербург
«БХВ-Петербург»
2011

УДК 681.3.068+800.92C#
ББК 32.973.26-018.1
3-59

Зиборов В. В.

3-59 Visual C# 2010 на примерах. — СПб.: БХВ-Петербург, 2011. — 432 с.: ил. + CD-ROM

ISBN 978-5-9775-0698-4

Рассмотрено более 120 типичных примеров, встречающихся в практике реального программирования для платформы .NET Framework в среде Microsoft Visual C# 2010: обработка событий мыши и клавиатуры, чтение/запись файлов, редактирование графических данных, управление буфером обмена, ввод/вывод данных, использование функций MS Word, MS Excel, AutoCAD и MATLAB, использование технологий LINQ и ADO.NET при работе с базами данных, разработка интерактивных Web-приложений, создание Web-служб, разработка WPF-приложений и многое другое. Материал располагается по принципу от простого к сложному, что позволяет использовать книгу одновременно как справочник для опытных и как пособие для начинающих программистов. Компакт-диск содержит исходные коды примеров из книги.

Для программистов

УДК 681.3.068+800.92C#
ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Анна Кузьмина*
Компьютерная верстка	*Натальи Смирновой*
Корректор	*Наталия Першакова*
Дизайн серии	*Игоря Цырульникова*
Оформление обложки	*Елены Беляевой*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 31.01.11.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 34,83.
Тираж 1500 экз. Заказ №
"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

Санитарно-эпидемиологическое заключение на продукцию № 77.99.60.953.Д.005770.05.09
от 26.05.2009 г. выдано Федеральной службой по надзору
в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 978-5-9775-0698-4

Оглавление

Введение

Система разработки программного обеспечения Microsoft Visual C# 2010 является хорошим средством *быстрой разработки программ* для ускоренного создания приложений для Microsoft Windows и Интернета. Цель книги — популяризация программирования. Автор стремился показать, как *малой кровью* можно написать, почти сконструировать, *как в детском конструкторе*, довольно-таки функционально сложные приложения. Для реализации этой цели автор выбрал форму демонстрации *на примерах* решения задач от самых простых, элементарных, до более сложных.

Так, рассмотрены примеры программ с экранной формой и элементами управления в форме, такими как текстовое поле, метка, кнопка и др. Написанные программы *управляются событиями*, в частности событиями мыши и клавиатуры. Поскольку большинство существующих программ взаимодействует с дисковой памятью, в книге приведены примеры чтения и записи файлов в долговременную память. Описаны решения *самых типичных задач*, которые встречаются в практике программирования, в частности работа *с графикой* и *буфером обмена*. Рассмотрены манипуляции табличными данными, использование обозревателя Web-страниц для отображения различных данных. Приведены примеры программирования с применением функций (методов) объектных библиотек систем Microsoft Office, MATLAB и AutoCAD. Разобраны вопросы обработки баз данных SQL Server и MS Access с помощью технологии ADO.NET. Рассмотрены методы обработки различных источников данных с использованием технологии LINQ. Приведены примеры программирования *Web-ориентированных приложений*, а также использования и разработки *Web-сервисов*. Новейшая технология WPF представлена несколькими выразительными примерами.

Несколько слов об особенностях книги. Спросите у любого программиста, *как он работает* (творит...) над очередной поставленной ему задачей. Он вам скажет, что всю задачу он мысленно *разбивает на фрагменты* и вспоминает, в каких уже решенных им задачах он *уже сталкивался с подобной ситуацией*. Далее он просто копирует фрагменты отлаженного программного кода и *вставляет их в новую задачу*. Сборник таких фрагментов (более 120 примеров) содержит данная книга. Автор пытался выделить *наиболее типичные*, актуальные задачи и решить их, с одной стороны, максимально эффективно, а с другой стороны, *кратко* и *выразительно*. Вместе с книгой читателю предлагается компакт-диск с рассмотренными в книге примерами.

Самая серьезная проблема в программировании больших, сложных программ — это *сложность, запутанность текстов*. Из-за запутанности программ имеются ошибки, нестыковки и проч. Как следствие — страдает производитель-

ность процесса создания программ *и их сопровождение*. Решение этой проблемы состоит в *структуризации* программ. Появление объектно-ориентированного программирования связано в большой степени со структуризацией программирования. Мероприятия для обеспечения большей структуризации — это проектирование программы как *иерархической структуры*, отдельные процедуры, входящие в программу, не должны быть *слишком длинными*, неиспользование операторов перехода `goto` и проч. Кроме того, современные системы программирования разрешают в названиях переменных, методов, свойств, событий, классов, объектов *использовать русские буквы*. Между тем современные программисты, как правило, *не используют* данную возможность, хотя когда вдруг в среде англоязычного текста появляются русские слова, это *вносит большую выразительность* в текст программы, и тем самым большую структуризацию. Программный код начинает от этого лучше читаться, *восприниматься человеком* (транслятору, компилятору — все равно).

Данная книга предназначена для начинающих программистов, программистов среднего уровня, а также для программистов, имеющих навыки разработки *на других языках* и желающих ускоренными темпами освоить новый для себя язык *Visual C# 2010*. Как пользоваться книгой? Эффективно пользоваться книгой можно, последовательно решая примеры в порядке их представления в книге, поскольку примеры расположены *от простого к более сложному*. И тем самым постепенно совершенствуя свои навыки программирования на *Visual C#*. А для программистов среднего уровня можно посоветовать искать *выборочно* именно те задачи, которые возникли у них при программировании их текущих задач.

Надеюсь, что читатель получит одновременно *удовольствие* и *пользу* от использования данной книги в своей работе и творчестве. Свои впечатления о данной книге присылайте по адресу **ziborov@ukr.net**, я с удовольствием их почитаю.

Глава 1

Простейшие программы с экранной формой и элементами управления

Пример 1. Форма, кнопка, метка и диалоговое окно

После инсталляции системы программирования Visual Studio 2010, включающей в себя Visual C# 2010, загрузочный модуль системы devenv.exe будет, скорее всего, расположен в папке: C:\Program Files\Microsoft Visual Studio 10.0\Common7\IDE.

Целесообразно создать ярлык на рабочем столе для запуска Visual C#. После запуска увидим начальный пользовательский интерфейс, показанный на рис. 1.1.

Рис. 1.1. Фрагмент стартовой страницы системы Visual Studio 2010

Чтобы запрограммировать какую-либо задачу, необходимо в пункте меню **File** выполнить команду **New Project**. В появившемся окне **New Project** в левой колонке находится список инсталлированных шаблонов (**Installed Templates**). Среди

них — шаблоны языков программирования, встроенных в Visual Studio, в том числе Visual Basic, Visual C#, Visual C++, Visual F# и др. Нам нужен язык Visual C#. В средней колонке выберем шаблон (Templates) **Windows Forms Application C#** и щелкнем на кнопке **OK**. В результате увидим окно, представленное на рис. 1.2.

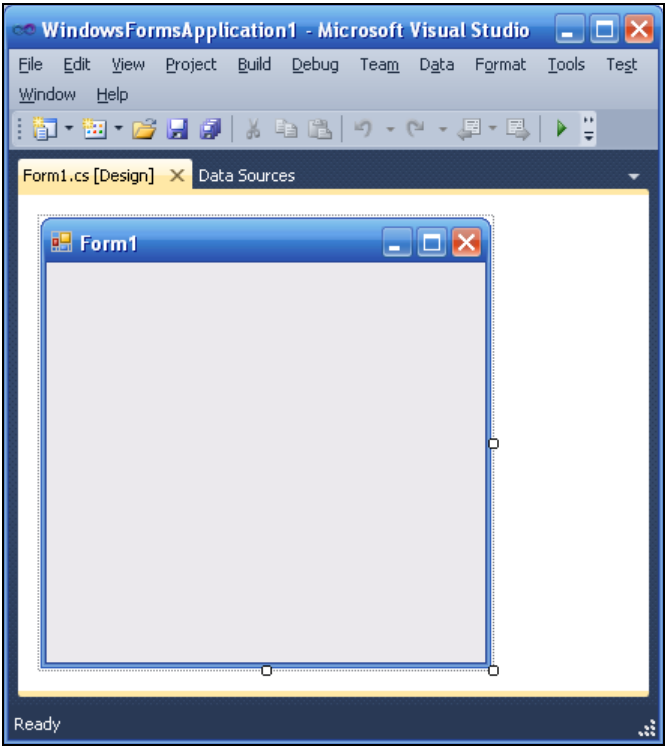


Рис. 1.2. Окно для проектирования пользовательского интерфейса

В этом окне изображена *экранная форма* — **Form1**, в которой программисты располагают различные компоненты графического интерфейса пользователя или, как их иначе называют, элементы управления. Это поля для ввода текста **TextBox**, командные кнопки **Button**, строчки текста в форме — метки **Label**, которые не могут быть отредактированы пользователем, и прочие элементы управления. Причем здесь используется самое современное так называемое *визуальное программирование*, предполагающее простое перетаскивание мышью из панели элементов **Toolbox**, где расположены всевозможные элементы управления, в форму. Таким образом, стараются свести к минимуму непосредственное написание программного кода.

Ваша первая программа будет отображать такую экранную форму, в которой будет что-либо написано, например "Microsoft Visual C# 2010", также в форме будет расположена командная кнопка с надписью "Нажми меня". При нажатии кнопки появится диалоговое окно с сообщением "Всем привет!"

Написать такую программку — вопрос 2—3 минут. Но вначале я хотел бы буквально двумя словами объяснить современный объектно-ориентированный подход к программированию. Подход заключается в том, что в программе все, что может быть названо именем существительным, называют *объектом*. Так, в нашей программе мы имеем четыре объекта: форму **Form**, надпись на форме **Label**, кнопку **Button** и диалоговое окно `MessageBox` с текстом "Всем привет!" (окно с приветом). Итак, добавьте метку и кнопку на форму примерно так, как показано на рис. 1.3.

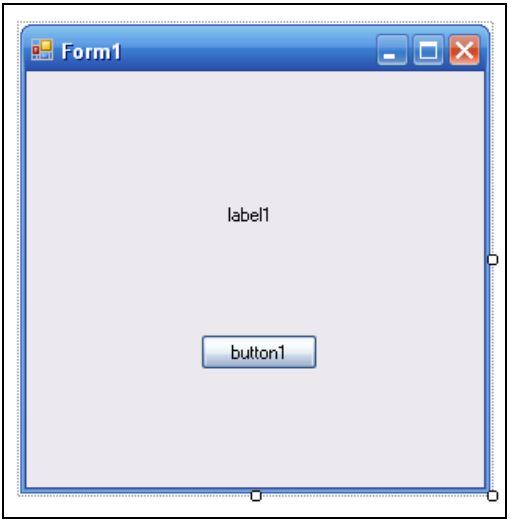


Рис. 1.3. Форма первого проекта

Любой такой объект можно создавать самому, а можно пользоваться готовыми объектами. В данной задаче мы пользуемся готовыми визуальными объектами, которые можно перетаскивать мышью из панели элементов управления **Toolbox**. В этой задаче нам нужно знать, что каждый объект имеет свойства (properties). Например, свойствами кнопки являются (рис. 1.4): имя кнопки (`Name`) — `button1`, надпись на кнопке (`Text`), расположение кнопки (`Location`) в системе координат формы `X`, `Y`, размер кнопки `Size` и т. д. Свойств много, их можно увидеть, если щелкнуть правой кнопкой мыши в пределах формы и выбрать в контекстном меню команду **Properties**; при этом появится панель свойств **Properties** (см. рис. 1.4).

Указывая мышью на все другие элементы управления в форме, можно просмотреть их свойства: формы `Form1` и надписи в форме — метки `label1`.

Вернемся к нашей задаче. Для объекта `label1` выберем свойство `Text` и напишем напротив этого поля "Microsoft Visual C# 2010" (вместо текста `label1`). Для объекта `button1` также в свойстве `Text` напишем "Нажми меня".

Кроме того, что объекты имеют свойства, следует знать, что объекты обрабатываются событиями. Событием, например, является щелчок на кнопке, щелчок в пределах формы, загрузка (`Load`) формы в оперативную память при старте программы и проч.

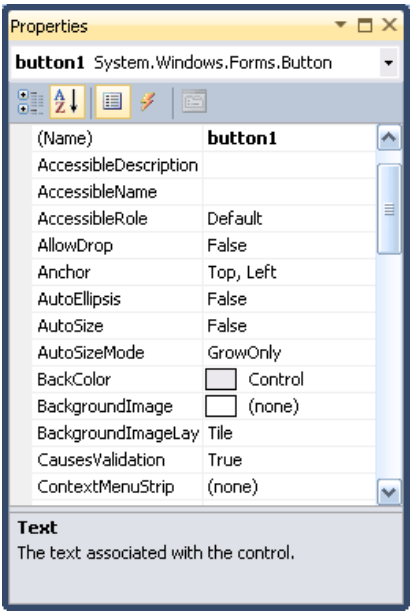


Рис. 1.4. Свойства кнопки `button1`

Управляют тем или иным событием посредством написания процедуры обработки события в программном коде. Для этого вначале нужно получить *"пустой" обработчик события*. В нашей задаче единственным событием, которым мы управляем, является щелчок на командной кнопке. Для получения пустого обработчика этого события следует в свойствах кнопки `button1` (см. рис. 1.4) щелкнуть на значке молнии **Events** (События) и в списке всех возможных событий кнопки `button1` выбрать двойным щелчком событие **Click**. После этого мы попадем на вкладку программного кода **Form1.cs** (рис. 1.5).

При этом управляющая среда Visual C# 2010 сгенерировала тот самый пустой обработчик события `button1_Click`:

```
private void button1_Click(object sender, EventArgs e) {   }
```

в фигурных скобках которого мы можем написать команды, подлежащие выполнению после щелчка на кнопке. Вы видите, что у нас теперь две вкладки: **Form1.cs** и **Form1.cs [Design]**, т. е. вкладка программного кода и вкладка визуального проекта программы (другое название этой вкладки — *дизайнер формы*). Переключаться между ними можно мышью или нажатием комбинации клавиш <Ctrl>+<Tab>, как это принято обычно между вкладками в Windows, а также функциональной клавишей <F7>.

Напомню, что после щелчка на кнопке должно появиться диалоговое окно, в котором написано: "Всем привет!" Поэтому в фигурных скобках обработчика события напишем:

```
MessageBox.Show("Всем привет!");
```

Здесь вызывается метод (программа) `Show` объекта `MessageBox` с текстом "Всем привет!" Таким образом, я здесь "нечаянно" проговорился о том, что объекты кроме свойств имеют также и *методы*, т. е. программы, которые обрабатывают объек-

ты. Кстати, после каждого оператора в С-программах[1] ставят *точку с запятой*. Это вместе с фигурными скобками по форме отличает С-программу от программных кодов на других языках программирования.

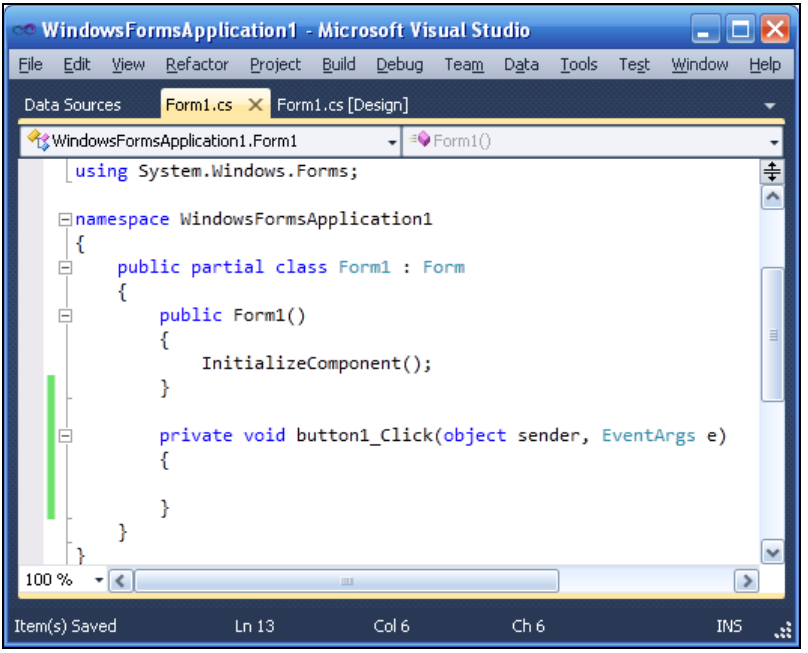


Рис 1.5. Вкладка программного кода

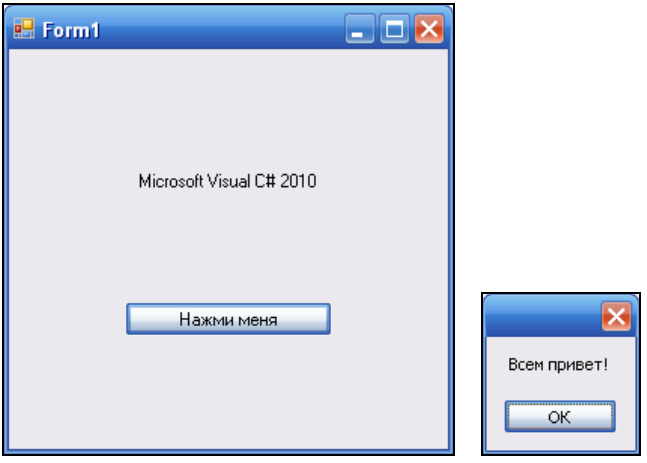


Рис. 1.6. Работающая программа

[1] Автор намеренно здесь указал "С", а не "С#", чтобы подчеркнуть особенность синтаксиса *любой* Си-программы, будь то С++, или С#, или Turbo-C.

Таким образом, мы написали *процедуру обработки события* щелчка (`Click`) на кнопке `button1`. Теперь нажмем клавишу <F5> и проверим работоспособность программы (рис. 1.6).

Поздравляю, вы написали свою первую программу на Visual C#!

Пример 2. Событие *MouseHover*

Немного усложним предыдущую задачу. Добавим еще одну обработку события `MouseHover` мыши для объекта `label1`. Событие `MouseHover` наступает тогда, когда пользователь указателем мыши "зависает" над каким-либо объектом, причем именно "зависает", а не просто перемещает мышь над объектом (от англ. *hover* — реять, парить). Есть еще событие `MouseEnter` (Войти), когда указатель мыши *входит в пределы* области элемента управления (в данном случае метки `label1`).

Чтобы добавить обработку события `MouseHover` мыши для объекта `label1`, следует так же, как это мы делали при обработке события "щелчок на кнопке", в панели **Properties** щелкнуть на значке молнии (**Events**) и двойным щелчком выбрать для объекта `label1` событие `MouseHover`. При этом осуществится переход на вкладку программного кода **Form1.cs** и среда Visual Studio 2010 сгенерирует простой обработчик события:

```
private void label1_MouseHover(object sender, EventArgs e) {  }
```

Между фигурными скобками вставим вызов диалогового окна:

```
MessageBox.Show("Событие Hover!");
```

Теперь проверим возможности программы: нажимаем клавишу <F5>, "зависаем" указателем мыши над `label1`, щелкаем на кнопке `button1`. Все работает!

А теперь я буду немного противоречить сам себе. Я говорил про визуальную технику программирования, направленную на минимизацию написания программного кода. А сейчас хочу сказать про *наглядность*, *оперативность*, *технологичность* работы программиста. Посмотрите на свойства каждого объекта в панели **Properties**. Вы видите, как много строчек. Если вы меняете какое-либо свойство, то оно будет выделено жирным шрифтом. Удобно! Но все-таки еще более удобно свойства объектов *назначать* (устанавливать) *в программном коде*. Почему?

Каждый программист имеет в своем арсенале множество уже отлаженных фрагментов, которые он использует в своей очередной новой программе. Программисту стоит лишь вспомнить, где он программировал ту или иную ситуацию. Программа, которую написал программист, имеет свойство быстро забываться. Если вы посмотрите на строчки кода, которые писали три месяца назад, то будете ощущать, что многое забыли; если прошел год, то вы смотрите на написанную вами программу, как на чужую. Поэтому при написании программ на первое место выходят *понятность*, *ясность*, *очевидность* написанного программного кода. Для этого каждая система программирования имеет какие-либо средства. Кроме того, сам программист придерживается некоторых правил, помогающих ему работать *производительно* и *эффективно*.

Назначать свойства объектов в программном коде удобно или сразу после инициализации компонентов формы (после процедуры `InitializeComponent`), или при обработке события `Form1_Load`, т. е. события загрузки формы в оперативную память при старте программы. Получим простой обработчик этого события. Для этого, как и в предыдущих случаях, можно выбрать нужное событие в панели свойств объекта, а можно еще проще: дважды щелкнуть в пределах проектируемой формы на вкладке **Form1.cs [Design]**. В любом случае получаем пустой обработчик события на вкладке программного кода. Заметим, что для формы таким умалчиваемым событием, для которого можно получить пустой обработчик двойным щелчком, является событие загрузки формы `Form1_Load`, для командной кнопки **Button** и метки **Label** таким событием является одиночный щелчок мышью на этих элементах управления. То есть если дважды щелкнуть в дизайнере формы по кнопке, то получим пустой обработчик `button1_Click` в программном коде, аналогично — для метки **Label**.

Итак, вернемся к событию загрузки формы, для него управляющая среда сгенерировала пустой обработчик:

```
private void Form1_Load(object sender, EventArgs e) { }
```

Здесь в фигурных скобках обычно вставляют свойства различных объектов и даже часто пишут много строчек программного кода. Здесь мы назначим свойству `Text` объекта `label1` значение "Microsoft Visual C# 2010":

```
label1.Text = "Microsoft Visual C# 2010";
```

Аналогично для объекта `button1`:

```
button1.Text = "Нажми меня!";
```

Совершенно необязательно писать каждую букву приведенных команд. Например, для первой строчки достаточно написать "la", уже это вызовет появление раскрывающегося меню, где вы сможете выбрать нужные для данного контекста ключевые слова. Это очень мощное и полезное современное средство, называемое IntellSense, для редактирования программного кода! Если вы от Visual Studio 2010 перешли в другую систему программирования, в которой отсутствует подобный сервис, то будете ощущать сильный дискомфорт.

Вы написали название объекта `label1`, поставили точку. Теперь вы видите раскрывающееся меню, где можете выбрать либо нужное свойство объекта, либо метод (т. е. подпрограмму). В данном случае выберите свойство `Text`.

Как видите, не следует пугаться слишком длинных ключевых слов, длинных названий объектов, свойств, методов, имен переменных. Система подсказок современных систем программирования значительно облегчает всю нетворческую работу. Вот почему в современных программах можно наблюдать такие длинные имена ключевых слов, имен переменных и проч. Я призываю вас, уважаемые читатели, также использовать в своих программах для названия переменных, объектов наиболее ясные, полные имена, причем можно на вашем родном русском языке. Потому что на первое место выходят ясность, прозрачность программирования, а громоздкость названий с лихвой компенсируется системой подсказок.

Далее хотелось бы, чтобы слева вверху формы на синем фоне (в так называемой строке заголовка) была не надпись "Form1", а что-либо осмысленное. Например, слово "Приветствие". Для этого ниже присваиваем эту строку свойству `Text` формы. Поскольку мы изменяем свойство объекта `Form1` внутри подпрограммы обработки события, связанного с формой, следует к форме обращаться через ссылку `this`: `this.Text = "Приветствие"` или `base.Text = "Приветствие"`.

После написания последней строчки кода мы должны увидеть на экране программный код, представленный в листинге 1.1.

Листинг 1.1. Программирование событий

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace WindowsFormsApplication1
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
      }
   // Обработка события загрузки формы:
      private void Form1_Load(object sender, EventArgs e)
      {
         this.Text = "Приветствие";
         label1.Text = "Microsoft Visual C# 2010";
         button1.Text = "Нажми меня";
      }
   // Обработка события щелчок на кнопке:
      private void button1_Click(object sender, EventArgs e)
      {
         MessageBox.Show("Всем привет!");
      }
   // Обработка события, когда указатель мыши "завис" над меткой:
      private void label1_MouseHover(object sender, EventArgs e)
```

```
        {
            MessageBox.Show("Событие Hover!");
        }
    }
}
```

Комментарии, поясняющие работу программы, в окне редактора кода будут выделены зеленым цветом, чтобы в тексте выразительно отделять его от прочих элементов программы. В С-подобных языках комментарий пишут после двух слэшей (`//`) или внутри пар `/* */`. Уважаемые читатели, даже если вам кажется весьма очевидным то, что вы пишете в программном коде, *напишите комментарий*. Как показывает опыт, даже весьма очевидный замысел программиста забывается удивительно быстро. Человеческая память отметает все, что по оценкам организма считается ненужным. Кроме того, даже если текст программы вполне ясен, в начале программы должны быть описаны ее назначение и способ использования, т. е. как бы "преамбула" программы. Далее в примерах мы будем следовать этому правилу.

На рис. 1.7 приведен фрагмент работы программы.

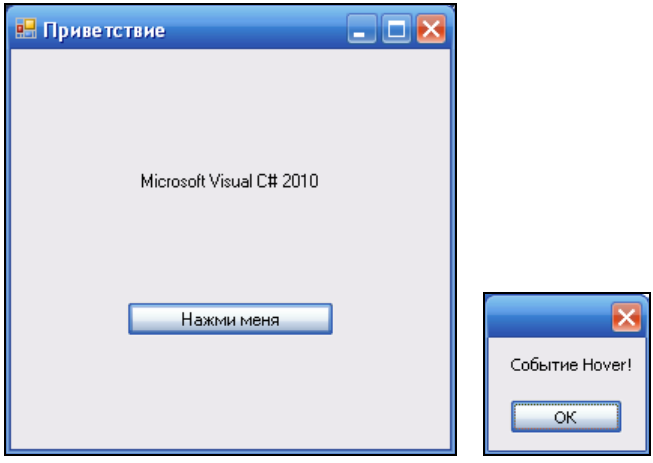


Рис. 1.7. Работа программы

Обычно в редакторах программного кода используется моноширинный шрифт, поскольку все символы такого шрифта имеют одинаковую ширину, в том числе точка и прописная русская буква "Ш". По умолчанию в редакторе программного кода C# 2010 задан шрифт Consolas. Однако если пользователь привык к шрифту Courier New, то его настройку можно изменить, выбрав меню **Tools | Options | Environment | Fonts and Colors**.

Теперь закроем проект (**File | Close Project**). Система предложит нам сохранить проект, сохраним проект под именем Hover. Теперь программный код этой программы можно посмотреть, открыв решение Hover.sln в папке Hover.

Пример 3. Ввод данных через текстовое поле *TextBox* с проверкой типа методом *TryParse*

При работе с формой очень часто ввод данных организуют через элемент управления "текстовое поле" **TextBox**. Напишем типичную программу, которая вводит через текстовое поле число, при нажатии командной кнопки извлекает из него квадратный корень и выводит результат на метку **Label**. В случае ввода не числа сообщает пользователю об этом.

Решая сформулированную задачу, запускаем Visual Studio, выбираем пункт меню **File | New | Project**, затем — шаблон **Windows Forms Application Visual C#** и щелкаем на кнопке **OK.** Далее из панели элементов управления **Toolbox** (если в данный момент вы не видите панель элементов, то ее можно добавить, например, с помощью комбинации клавиш <Ctrl>+<Alt>+<X> или меню **View | Toolbox**) в форму указателем мыши перетаскиваем текстовое поле **TextBox**, метку **Label** и командную кнопку **Button**. Таким образом, в форме будут находиться три элемента управления.

Теперь перейдем на вкладку программного кода, для этого правой кнопкой мыши вызовем контекстное меню и выберем в нем пункт **View Code**. Здесь сразу после инициализации компонентов формы, т. е. после вызова процедуры `InitializeComponent` задаем свойствам формы (к форме обращаемся посредством ссылки `this` или `base`), кнопкам `button1` и текстового поля `textBox1`, метке `label1` следующие значения:

```
this.Text = "Извлечение квадратного корня";
button1.Text = "Извлечь корень";
textBox1.Clear();     // Очистка текстового поля
label1.Text = null;  // или = string.Empty;
```

Нажмем клавишу <F5> для выявления возможных опечаток, т. е. синтаксических ошибок и предварительного просмотра дизайна будущей программы.

Далее программируем событие `button1_Click` — щелчок мышью на кнопке **Извлечь корень**. Создать пустой обработчик этого события удобно, дважды щелкнув мышью на этой кнопке. Между двумя появившимися строчками программируем диагностику правильности вводимых данных, конвертирование строковой переменной в переменную типа `Single` и непосредственное извлечение корня (листинг 1.2).

Листинг 1.2. Извлечение корня с проверкой типа методом `TryParse`

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
```

```
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

// Программа вводит через текстовое поле число, при щелчке на командной кнопке
// извлекает из него квадратный корень и выводит результат на метку label1.
// В случае ввода не числа сообщает пользователю об этом,
// выводя красным цветом предупреждение также на метку label1.

namespace Корень
{
   public partial class Form1 : Form
   {
      public Form1()
      {  // Инициализация компонентов формы
         InitializeComponent();
         button1.Text = "Извлечь корень";
         label1.Text = null;    // или = string.Empty;
         this.Text = "Извлечение квадратного корня";
         textBox1.Clear();      // очистка текстового поля
         textBox1.TabIndex = 0; // установка фокуса в текстовом поле
      }

      private void button1_Click(object sender, EventArgs e)
      {  // Обработка щелчка на кнопке "Извлечь корень"
         Single X;   // - из этого числа будем извлекать корень
         // Преобразование из строковой переменной в Single:
         bool Число_ли = Single.TryParse(textBox1.Text,
            System.Globalization.NumberStyles.Number,
            System.Globalization.NumberFormatInfo.CurrentInfo, out X);
         // Второй параметр - это разрешенный стиль числа (Integer,
         // шестнадцатиричное число, экспоненциальный вид числа и проч.).
         // Третий параметр форматирует значения на основе текущего
         // языка и региональных параметров из
         // Панели управления -> Язык и региональные стандарты
         // число допустимого формата;
         // метод возвращает значение в переменную X
         if (Число_ли == false)
         {  // Если пользователь ввел не число:
            label1.Text = "Следует вводить числа";
            label1.ForeColor = Color.Red;  // красный цвет текста на метке
```

```
            return;                          // выход из процедуры
        }
        Single Y = (Single)Math.Sqrt(X);  // извлечение корня
        label1.ForeColor = Color.Black;   // черный цвет текста на метке
        label1.Text = string.Format("Корень из {0} равен {1:F5}", X, Y);
    }
  }
}
```

Здесь при обработке события "щелчок мышью" на кнопке **Извлечь корень** проводится проверка, введено ли число в текстовом поле. Проверка осуществляется с помощью функции `TryParse`. Первым параметром метода `TryParse` является анализируемое поле `textBox1.Text`. Второй параметр — это разрешаемый для преобразования стиль числа, он может быть целого типа (`Integer`), шестнадцатеричным (`HexNumber`), представленным в экспоненциальном виде и проч. Третий параметр указывает, на какой основе формируется допустимый формат, в нашем случае мы использовали `CurrentInfo`, т. е. на основе текущего языка и региональных параметров. По умолчанию при инсталляции руссифицированной версии Windows разделителем целой и дробной части числа является запятая. Однако эту установку можно изменить, если в Панели управления выбрать значок **Язык и региональные стандарты**, затем на вкладке **Региональные параметры** щелкнуть на кнопке **Настройка** и на появившейся новой вкладке указать в качестве разделителя целой и дробной частей точку вместо запятой. В обоих случаях (и для запятой, и для точки) метод `TryParse` будет работать так, как указано на вкладке **Региональные параметры**.

Четвертый параметр метода `TryParse` возвращает результат преобразования. Кроме того, функция `TryParse` возвращает булеву переменную `true` или `false`, которая сообщает, успешно ли выполнено преобразование. Как видно из текста программы, если пользователь ввел не число (например, введены буквы), то на метку `label1` выводится красным цветом текст "Следует вводить числа". Далее, поскольку ввод неправильный, организован выход из программы обработки события `button1_Click` с помощью оператора `return`. На рис. 1.8 показан фрагмент работы программы.

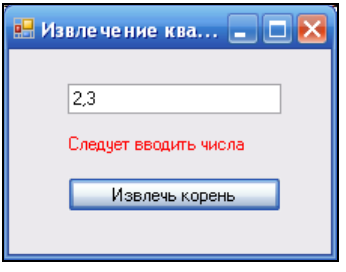


Рис 1.8. Фрагмент работы программы

Как видно из рисунка, функция `TryParse` не восприняла введенные символы "2,3" как число, поскольку автор специально для демонстрации данного примера указал на вкладке **Региональные параметры** точку в качестве разделителя целой и дробной частей.

Если пользователь ввел все-таки число, то будет выполняться следующий оператор извлечения квадратного корня `Math.Sqrt(X)`. Математические функции Visual Studio 2010 являются методами класса `Math`. Их можно увидеть, набрав `Math` и поставив точку (`.`). В выпадающем списке вы увидите множество математических функций: `Abs`, `Sin`, `Cos`, `Min` и т. д. и два свойства — две константы `E = 2.71...` (основание натуральных логарифмов) и `PI = 3,14...` (число диаметров, уложенных вдоль окружности). Функция `Math.Sqrt(X)` возвращает значение типа `double` (двойной точности с плавающей запятой), которое *приводим* с помощью неявного преобразования (`Single`) к переменной одинарной точности.

Последней строчкой обработки события `button1_Click` является формирование строки `label1.Text` с использованием метода `string.Format`. Использованный формат `"Корень из {0} равен {1:F5}"` означает: взять нулевой выводимый элемент, т. е. переменную `X`, и записать эту переменную вместо фигурных скобок; после чего взять первый выводимый элемент, т. е. `Y`, и записать его вместо вторых фигурных скобок в формате с фиксированной точкой и пятью десятичными знаками после запятой.

Нажав клавишу <F5>, проверяем, как работает программа.

Результат работающей программы представлен на рис. 1.9.

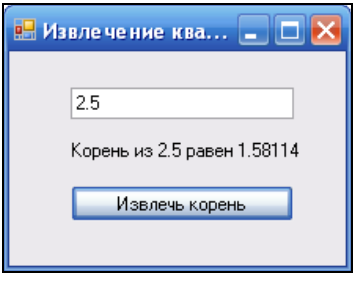


Рис. 1.9. Извлечение квадратного корня

Если появились ошибки, то работу программы следует проверить отладчиком — клавиши <F8> или <F11>. В этом случае управление останавливается на каждом операторе, и вы можете проверить значение каждой переменной, наводя указатель мыши на переменные. Можно выполнить программу до определенной программистом точки *(точки останова)*, используя, например, клавишу <F9> или оператор `Stop`, и в этой точке проверить значения необходимых переменных.

Убедиться в работоспособности этой программы можно, открыв соответствующее решение в папке Корень.

Пример 4. Ввод пароля в текстовое поле и изменение шрифта

Это очень маленькая программа для ввода пароля в текстовое поле, причем при вводе вместо вводимых символов некто, "находящийся за спиной пользователя", увидит только звездочки. Программа состоит из формы, текстового поля **TextBox**, метки **Label**, куда для демонстрации возможностей мы будем копировать пароль (паспорт, т. е. секретные слова) и командной кнопки **Button — Покажи паспорт**.

Перемещаем в форму все названные элементы управления. Текст программы приведен в листинге 1.3.

Листинг 1.3. Ввод пароля

```
// Программа для ввода пароля в текстовое поле, причем при вводе вместо
// вводимых символов некто, "находящийся за спиной пользователя",
// увидит только звездочки

using System;
using System.Drawing;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются
// в данной программе

namespace Passport
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
         base.Text = "Введи пароль";
         textBox1.Text = null; textBox1.TabIndex = 0;
         textBox1.PasswordChar = '*';
         textBox1.Font = new Font("Courier New", 9.0F);
         // или textBox1.Font = new Font(FontFamily.GenericMonospace, 9.0F);
         label1.Text = string.Empty;
         label1.Font = new Font("Courier New", 9.0F);
         button1.Text = "Покажи паспорт";
      }
      private void button1_Click(object sender, EventArgs e)
      {  // Обработка события "щелчок на кнопке"
```

```
            label1.Text = textBox1.Text;
        }
    }
}
```

Как видно из текста программы, сразу после инициализации компонентов формы, т. е. после вызова процедуры `InitializeComponent`, очищаем текстовое поле и делаем его "защищенным от посторонних глаз" с помощью свойства `textBox1.PasswordChar`, каждый введенный пользователем символ маскируется символом звездочки (`*`). Далее мы хотели бы для большей выразительности и читабельности программы, чтобы вводимые звездочки и результирующий текст имели одинаковую длину. Все символы шрифта Courier New имеют одинаковую ширину, поэтому его называют моноширинным шрифтом. Кстати, используя именно этот шрифт, удобно программировать таблицу благодаря одинаковой ширине букв этого шрифта. Еще одним широко используемым моноширинным шрифтом является шрифт Consola. Задаем шрифт, используя свойство `Font` обоих объектов: `textBox1` и `label1`. Число `9.0` означает размер шрифта. Свойство текстового поля `TabIndex = 0` обеспечивает передачу фокуса при старте программы именно в текстовое поле.

Осталось обработать событие `button1_Click` — щелчок на кнопке. Здесь — банальное присваивание текста из поля тексту метки. Программа написана, нажимаем клавишу <F5>. На рис. 1.10 приведен вариант работы данной программы.

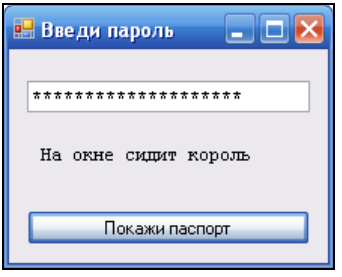


Рис 1.10. Вариант работы программы

При необходимости используйте отладчик (клавиша <F11> или <F10>) для *пошагового выполнения программы* и выяснения всех промежуточных значений переменных путем "зависания" указателя мыши над переменными.

Убедиться в работоспособности программы можно, открыв решение Passport.sln в папке Passport.

Пример 5. Управление стилем шрифта с помощью элемента управления *CheckBox*

Кнопка **CheckBox** (Флажок) также находится на панели элементов управления **Toolbox**. Флажок может быть либо установлен (содержит "галочку"), либо сброшен

(пустой). Напишем программу, которая управляет стилем шрифта текста, выведенного на метку **Label**. Управлять стилем будем посредством флажка **CheckBox**.

Используя панель элементов **Toolbox**, в форму поместим метку `label1` и флажок `checkBox1`. В листинге 1.4 приведен текст программы управления этими объектами.

Листинг 1.4. Управление стилем шрифта

```
// Программа управляет стилем шрифта текста, выведенного на метку
// Label, посредством флажка CheckBox
using System;
using System.Drawing;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются
// в данной программе

namespace CheckBox
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
         this.Text = "Флажок CheckBox";
         checkBox1.Text = "Полужирный"; checkBox1.Focus();
         label1.Text = "Выбери стиль шрифта";
         label1.TextAlign = ContentAlignment.MiddleCenter;
         label1.Font = new System.Drawing.Font("Courier New", 14.0F);
      }
      private void checkBox1_CheckedChanged(object sender, EventArgs e)
      { // Изменение состояния флажка на противоположное
         if (checkBox1.Checked == true) label1.Font =
                     new Font("Courier New", 14.0F, FontStyle.Bold);
         if (checkBox1.Checked == false) label1.Font =
                     new Font("Courier New", 14.0F, FontStyle.Regular);
      }
   }
}
```

Сразу после вызова процедуры `InitializeComponent` задаем начальные значения некоторых свойств объектов `Form1` (посредством ссылки `this`), `label1` и `checkBox1`. Так, тексту флажка, выводимого с правой стороны, присваиваем значение "Полужирный". Кроме того, при старте программы фокус должен находиться

на флажке (`checkBox1.Focus();`), в этом случае пользователь может изменять установку флажка даже клавишей <Пробел>.

Текст метки — "Выбери стиль шрифта", выравнивание метки `TextAlign` задаем посередине и по центру (`MiddleCenter`) относительно всего того места, что предназначено для метки. Задаем шрифт метки Courier New (в этом шрифте все буквы имеют одинаковую ширину) размером 14 пунктов.

Изменение состояния флажка соответствует событию `CheckedChanged`. Чтобы получить пустой обработчик события `CheckedChanged`, следует дважды щелкнуть на элементе `checkBox1` вкладки **Form1.cs [Design]**. Между соответствующими строчками следует записать (см. текст программы): если флажок установлен (т. е. содержит "галочку") `Checked = true`, то для метки `label1` устанавливается тот же шрифт Courier New, 14 пунктов, но `Bold`, т. е. полужирный.

Далее — следующая строчка кода: если флажок не установлен, т. е. `checkBox1.Checked = false`, то шрифт устанавливается `Regular`, т. е. обычный. Очень часто эту ситуацию программируют, используя ключевое слово `else` (иначе), однако это выражение будет выглядеть более выразительно и понятно так, как написали мы.

Программа написана, нажмите клавишу <F5>. Проверьте работоспособность программы. В рабочем состоянии она должна работать примерно так, как показано на рис. 1.11.

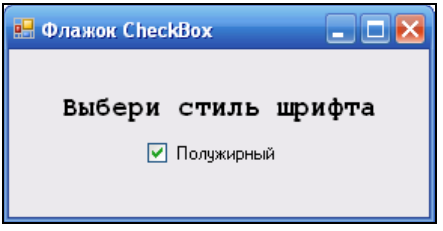


Рис. 1.11. Фрагмент работы программы управления стилем шрифта

Убедиться в работоспособности программы можно, открыв решение CheckBox1.sln в папке CheckBox1.

Пример 6. Побитовый оператор "исключающее ИЛИ"

Несколько изменим предыдущую программу в части обработки события `CheckedChanged` (Изменение состояния флажка). Вместо двух условий `if()` напишем один оператор:

```
label1.Font = new Font("Courier New", 14.0F,
                       label1.Font.Style ^ FontStyle.Bold);
```

Здесь каждый раз при изменении состояния флажка значение параметра `label1.Font.Style` сравнивается с одним и тем же значением `FontStyle.Bold`. Поскольку между ними стоит побитовый оператор `^` (исключающее ИЛИ), он будет назначать `Bold`, если текущее состояние `label1.Font.Style` "не Bold". А если `label1.Font.Style` пребывает в состоянии "Bold", то оператор `^` будет назначать состояние "не Bold". Этот оператор еще называют логическим `XOR`.

Таблица истинности логического `XOR` такова:

```
A Xor B = C
0 Xor 0 = 0
1 Xor 0 = 1
0 Xor 1 = 1
1 Xor 1 = 0
```

В нашем случае мы имеем всегда `B = 1` (`FontStyle.Bold`), а `A` (`Label1.Font.Style`) попеременно то `Bold`, то `Regular` (т. е. "не Bold"). Таким образом, оператор `^` всегда будет назначать противоположное тому, что записано в `label1.Font.Style`.

Как видно, применение побитового оператора привело к существенному уменьшению количества программного кода. Использование побитовых операторов может значительно упростить написание программ со сложной логикой.

Посмотрите, как работает программа, нажав клавишу <F5>.

Теперь добавим в форму еще один элемент управления **CheckBox**. Мы собираемся управлять стилем шрифта `FontStyle` двумя флажками. Один, как и прежде, задает полужирный стиль `Bold` или обычный `Regular`, а второй задает наклонный `Italic` или возвращает в `Regular`. Текст новой программы приведен в листинге 1.5.

Листинг 1.5. Усовершенствованный программный код

```
// Побитовый оператор "^" - исключающее ИЛИ
using System;
using System.Drawing;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются
// в данной программе

namespace CheckBox2
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
         this.Text = "Флажок CheckBox";
         checkBox1.Text = "Полужирный"; checkBox2.Text = "Наклонный";
```

```
            label1.Text = "Выбери стиль шрифта";
            label1.TextAlign = ContentAlignment.MiddleCenter;
            label1.Font = new Font("Courier New", 14.0F);
        }
        private void checkBox1_CheckedChanged(object sender, EventArgs e)
        {
            label1.Font = new System.Drawing.Font(
                "Courier New", 14.0F, label1.Font.Style ^ FontStyle.Bold);
        }
        private void checkBox2_CheckedChanged(object sender, EventArgs e)
        {
            label1.Font = new System.Drawing.Font(
                "Courier New", 14.0F, label1.Font.Style ^ FontStyle.Italic);
        }
    }
}
```

Как видно, здесь принципиально ничего нового нет, только лишь добавлена обработка события изменения состояния флажка `CheckedChanged` для `CheckBox2`. Фрагмент работы программы можно увидеть на рис. 1.12.

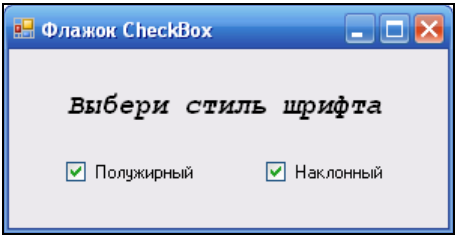


Рис. 1.12. Фрагмент работы усовершенствованной программы

Убедиться в работоспособности программы можно, открыв решение CheckBox2.sln в папке CheckBox2.

Пример 7. Вкладки *TabControl* и переключатели *RadioButton*

Вкладки используются для организации управления и оптимального расположения экранного пространства. Выразительным примером использования вкладок является диалоговое окно **Свойства обозревателя** Internet Explorer. То есть если требуется отобразить большое количество управляемой информации, то весьма уместно использовать вкладки **TabControl**.

Поставим задачу написать программу, позволяющую выбрать текст из двух вариантов, задать цвет и размер шрифта этого текста на трех вкладках **TabControl** с использованием переключателей **RadioButton**.

Фрагмент работы программы приведен на рис. 1.13.

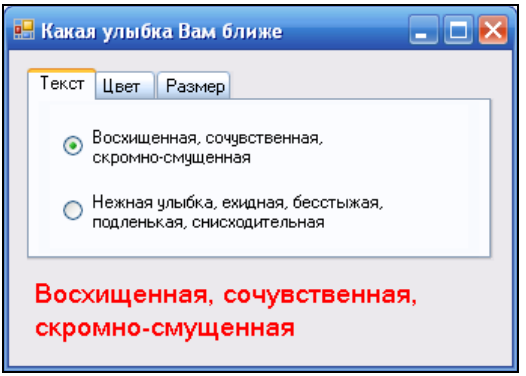


Рис. 1.13. Программа с переключателями и вкладками

Программируя поставленную задачу, создадим новый проект **Windows Forms Application C#**, назовем его, например, ВкладкиTabControl, получим стандартную форму. Затем, используя панель элементов **Toolbox**, в форму перетащим мышью элемент управления **TabControl**. Как видно, по умолчанию имеем две вкладки, а по условию задачи, как показано на рис. 1.13, три вкладки. Добавить третью вкладку можно в конструкторе формы, а можно программно.

Вначале покажем, как добавить третью вкладку в конструкторе. Для этого в свойствах (окно **Properties**) элемента управления `TabControl1` выбираем свойство `TabPages`, в результате попадаем в диалоговое окно **TabPage Collection Edit**, где добавляем (кнопка **Add**) третью вкладку (первые две присутствуют по умолчанию). Эти вкладки нумеруются от нуля, т. е. третья вкладка будет распознаваться как `TabPages(2)`. Название каждой вкладки будем указывать в программном коде.

Для того чтобы читатель в большей степени мог управлять всем процессом, мы покажем, как добавить третью вкладку не в конструкторе, а в программном коде сразу после вызова процедуры `InitializeComponent` (листинг 1.6). Однако, прежде чем перейти на вкладку программного кода, для каждой вкладки выбираем из панели **Toolbox** по два переключателя **RadioButton**, а в форму перетаскиваем метку **Label**. Теперь через щелчок правой кнопкой мыши в пределах формы переключаемся на редактирование программного кода.

Листинг 1.6. Использование вкладок и переключателей

```
// Программа, позволяющая выбрать текст из двух вариантов, задать цвет
// и размер шрифта для этого текста на трех вкладках TabControl
// с использованием переключателей RadioButton
```

```
using System;
using System.Drawing;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются
// в данной программе

namespace ВкладкиTabControl
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
         // Создание третьей вкладки "программно":
         System.Windows.Forms.TabPage tabPage3 =
                           new System.Windows.Forms.TabPage();
         tabPage3.UseVisualStyleBackColor = true;
         // Добавление третьей вкладки в существующий набор
         // вкладок tabControl1:
         this.tabControl1.Controls.Add(tabPage3);
         //this.Visible = false;

         // Добавление переключателей 5 и 6 на третью вкладку:
         tabPage3.Controls.Add(this.radioButton5);
         tabPage3.Controls.Add(this.radioButton6);

         // Расположение переключателей 5 и 6:
         this.radioButton5.Location = new System.Drawing.Point(20, 15);
         this.radioButton6.Location = new System.Drawing.Point(20, 58);

         this.Text = "Какая улыбка Вам ближе";
         // Задаем названия вкладок:
         tabControl1.TabPages[0].Text = "Текст";
         tabControl1.TabPages[1].Text = "Цвет";
         tabControl1.TabPages[2].Text = "Размер";

         // Эта пара переключателей изменяет текст:
         radioButton1.Text =
                  "Восхищенная, сочувственная,\nскромно-смущенная";
         radioButton2.Text = "Нежная улыбка, ехидная, бес" +
                           "стыжая,\nподленькая, снисходительная";
         // или
```

```
        //radioButton2.Text = "Нежная улыбка, бесстыжая," +
        //Environment.NewLine + "подленькая, снисходительная";

        // Эта пара переключателей изменяет цвет текста:
        radioButton3.Text = "Красный";
        radioButton4.Text = "Синий";

        // Эта пара переключателей изменяет размет шрифта:
        radioButton5.Text = "11 пунктов";
        radioButton6.Text = "13 пунктов";

        label1.Text = radioButton1.Text;
    }

    // Далее - обработка событий изменения состояния шести переключателей
    private void radioButton1_CheckedChanged(object sender, EventArgs e)
        { label1.Text = radioButton1.Text; }

    private void radioButton2_CheckedChanged(object sender, EventArgs e)
        { label1.Text = radioButton2.Text; }

    private void radioButton3_CheckedChanged(object sender, EventArgs e)
        { label1.ForeColor = Color.Red; }

    private void radioButton4_CheckedChanged(object sender, EventArgs e)
        { label1.ForeColor = Color.Blue; }

    private void radioButton5_CheckedChanged(object sender, EventArgs e)
        { label1.Font = new Font(label1.Font.Name, 11); }

    private void radioButton6_CheckedChanged(object sender, EventArgs e)
        { label1.Font = new Font(label1.Font.Name, 13); }
  }
}
```

Как видно из текста программы, сразу после вызова процедуры `InitializeComponent` (этот программный код можно было бы задать при обработке события загрузки формы `Form1_Load`) создаем "программно" третью вкладку и добавляем ее в набор вкладок `tabControl1`, созданный в конструкторе. Далее "привязываем" пятый и шестой переключатели к третьей вкладке.

Дальнейшие установки очевидны и не требуют дополнительных комментариев. Заметим, что каждая пара переключателей, расположенных на каком-либо элемен-

те управления (в данном случае на различных вкладках), "отрицают" друг друга, т. е. если пользователь выбрал один, то другой переходит в противоположное состояние. Отслеживать изменения состояния переключателей удобно с помощью обработки событий переключателей `CheckChanged` (см. листинг 1.6). Чтобы получить пустой обработчик этого события в конструкторе формы, следует дважды щелкнуть на соответствующем переключателе и таким образом запрограммировать изменения состояния переключателей.

Убедиться в работоспособности программы можно, открыв решение ВкладкиTabControl.sln в папке ВкладкиTabControl.

Пример 8. Свойство *Visible* и всплывающая подсказка *ToolTip* в стиле *Balloon*

Продемонстрируем возможности свойства `Visible` (Видимый). Программа пишет в метку **Label** некоторый текст, а пользователь с помощью командной кнопки делает этот текст невидимым, а затем опять видимым и т. д. При зависании мыши над кнопкой появляется подсказка "Нажми меня".

Запускаем Visual Studio 2010, далее выбираем пункты меню **File | New | Project | Windows Forms Application C#** и нажимаем кнопку **OK**. Затем из панели элементов управления **Toolbox** в форму перетаскиваем метку **Label**, кнопку **Button** и всплывающую подсказку **ToolTip**. Только в этом случае каждый элемент управления в форме (включая форму) получает свойство `ToolTip on Tip`. Убедитесь в этом, посмотрев свойства (окно **Properties**) элементов управления.

Для кнопки `button1` напротив свойства `ToolTip on Tip` мы могли бы написать "Нажми меня". Однако я предлагаю написать это непосредственно в программном коде. В этом случае программист не будет долго искать соответствующее свойство, когда станет применять данный фрагмент в своей новой программе!

Перейдем на вкладку программного кода — щелчок правой кнопкой мыши в пределах формы и выбор команды **View Code**. Окончательный текст программы представлен в листинге 1.7.

Листинг 1.7. Свойство `Visible` и всплывающая подсказка *ToolTip*

```
// Программа пишет в метку Label некоторый текст, а пользователь
// с помощью командной кнопки делает этот текст либо видимым, либо
// невидимым. Здесь использовано свойство Visible. При зависании мыши
// над кнопкой появляется подсказка "Нажми меня" (свойство ToolTip).
using System;
using System.Drawing;
using System.Windows.Forms;
```

```
// Другие директивы using удалены, поскольку они не используются
// в данной программе

namespace Visible
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
         base.Text = "Житейская мудрость";
         label1.Text = "Сколько ребенка не учи хорошим манерам,\n" +
                       "он будет поступать так, как папа с мамой";
         label1.TextAlign = ContentAlignment.MiddleCenter;
         button1.Text = "Кнопка";
         toolTip1.SetToolTip(button1, "Кнопка\r\nсчастья");
         // Должна ли всплывающая подсказка использовать всплывающее окно:
         toolTip1.IsBalloon = true;
         // Если IsBalloon = false, то используется стандартное
         // прямоугольное окно
      }
      private void button1_Click(object sender, EventArgs e)
      {  // Обработка события "щелчок на кнопке".
         // Можно программировать так:
         // if (label1.Visible == true) label1.Visible = false;
         // else label1.Visible = true;
         // или так:
         // label1.Visible = label1.Visible ^ true;
         // здесь "^" - логическое исключающее ИЛИ,
         // или совсем просто:
         label1.Visible = !label1.Visible;
      }
   }
}
```

Сразу после вызова процедуры `InitializeComponent` свойству `Text` метки присваиваем некоторый текст, "склеивая" его с помощью знака "плюс" (+) из отдельных фрагментов. Использование в текстовой строке символов `"\n"` (или `"\r\n"`) означает перенос текста на новую строку (это так называемый *перевод каретки*). Можно переводить каретку с помощью строки `Environment.NewLine`. В перечислении `Environment` можно выбрать и другие управляющие символы. Свойство метки `TextAlign` располагает текст метки по центру и посередине (`MiddleCenter`). Выра-

жение, содержащее `ToolTip1`, устанавливает (`Set`) текст всплывающей подсказки для кнопки `button1` при "зависании" над ней указателя мыши (рис. 1.14). По умолчанию свойство `IsBalloon` пребывает в состоянии `false`, и при этом во всплывающей подсказке используется стандартное прямоугольное окно, установка `IsBalloon = true` переводит подсказку в стиль комиксов (в стиль Balloon), см. рис. 1.14.

Чтобы в программном коде получить пустой обработчик события "щелчок мышью на кнопке", следует в дизайнере (конструкторе) формы (т. е. на вкладке **Form1.cs[Designer]**) дважды щелкнуть на кнопке `button1`. При обработке этого события, как видно, закомментированы пять строчек, в которых записана логика включения видимости метки или ее выключение. Логика абсолютно понятна: если свойство видимости (`Visible`) *включено* (`true`), то его следует *выключить* (`false`); иначе (`else`) — включить.

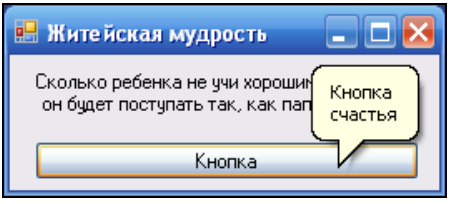


Рис. 1.14. Фрагмент работы программы

Несколько путано, но разобраться можно. И все работает. Проверьте! Кнопку можно нажимать мышью, клавишей <Enter> и клавишей <Пробел>.

Однако можно пойти другим путем. Именно поэтому пять строчек этой сложной логики переведены в комментарий. Мы уже встречались с побитовым оператором ^ (исключающее ИЛИ). Напоминаю, что этот оператор, говоря кратко, выбирает "да" (`true`), сравнивая "нет" и "да", и выбирает "нет" (`false`), сравнивая "да" и "да". Однако можно еще более упростить написание программного кода:

```
label1.Visible = ! label1.Visible;
```

То есть при очередной передаче управления на эту строчку свойство `label1.Visible` будет принимать противоположное значение. Вы убедились, что можно по-разному программировать подобные ситуации.

Как видно, мы использовали много закомментированных строчек программного кода. Очень удобно комментировать строки программного кода в редакторе Visual Studio, вначале выделяя их, а затем использовать комбинацию клавиш <Ctrl>+<K> | <C> (Comment). Чтобы убрать с нескольких строк знак комментария, можно аналогично вначале отметить их, а затем пользоваться уже дугой комбинацией клавиш <Ctrl>+<K> | <U> (Uncomment).

Текст программы можно посмотреть, открыв решение Visible.sln в папке Visible.

Пример 9. Калькулятор на основе комбинированного списка *ComboBox*

Элемент управления **ComboBox** служит для отображения вариантов выбора в выпадающем списке. Продемонстрируем работу этого элемента управления на примере программы, реализующей функции калькулятора. Здесь для отображения вариантов выбора арифметических операций используется комбинированный список **ComboBox**.

После запуска Visual Studio 2010 и выбора шаблона **Windows Forms Application C#** из панели **Toolbox** перетащим в форму два текстовых поля **TextBox**, метку **Label** и комбинированный список **ComboBox**.

Текст программы представлен в листинге 1.8.

Листинг 1.8. Суперкалькулятор

```
// Программа, реализующая функции калькулятора. Здесь для отображения
// вариантов выбора арифметических действий используется комбинированный
// список ComboBox
using System;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются
// в данной программе

namespace ComboBox_Calc
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
         comboBox1.Text = "Выбери операцию";
         comboBox1.Items.AddRange(new string[] {"Прибавить",
                  "Отнять", "Умножить", "Разделить", "Очистить"});
         comboBox1.TabIndex = 2;
         textBox1.Clear(); textBox2.Clear();
         textBox1.TabIndex = 0; textBox2.TabIndex = 1;
         this.Text = "Суперкалькулятор";
         label1.Text = "Равно: ";
      }
      private void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
      {
```

```
        // Обработка события изменения индекса выбранного элемента
        label1.Text = "Равно: ";
        // Преобразование из строковой переменной в Single:
        Single X, Y, Z = 0;
        bool Число_ли1 = Single.TryParse(textBox1.Text,
           System.Globalization.NumberStyles.Number,
           System.Globalization.NumberFormatInfo.CurrentInfo, out X);
        bool Число_ли2 = Single.TryParse(textBox2.Text,
           System.Globalization.NumberStyles.Number,
           System.Globalization.NumberFormatInfo.CurrentInfo, out Y);
        if (Число_ли1 == false || Число_ли2 == false)
        {
           MessageBox.Show("Следует вводить числа!", "Ошибка",
                           MessageBoxButtons.OK, MessageBoxIcon.Error);
           return;
        }
        // Оператор множественного выбора:
        switch (comboBox1.SelectedIndex) // Выбор арифметической операции:
        {
           case 0:   // Выбрали "Прибавить":
              Z = X + Y; break;
           case 1:   // Выбрали "Отнять":
              Z = X - Y; break;
           case 2:   // Выбрали "Умножить":
              Z = X * Y; break;
           case 3:   // Выбрали "Разделить":
              Z = X / Y; break;
           case 4:   // Выбрали "Очистить":
              textBox1.Clear(); textBox2.Clear();
              label1.Text = "Равно: "; return;
        }
        label1.Text = string.Format("Равно {0:F5}", Z);
     }
  }
}
```

В данной программе сразу после вызова процедуры `InitializeComponent` присваиваем начальные значения некоторым свойствам, в том числе задаем коллекцию элементов комбинированного списка: "Прибавить", "Отнять" и т. д. Здесь также задаем табличные индексы `TabIndex` для текстовых полей и комбинированного списка. Табличный индекс определяет *порядок обхода элементов*. Так, при старте программы фокус будет находиться в первом текстовом поле, поскольку мы назна-

чили `textBox1.TabIndex = 0`. Далее при нажатии пользователем клавиши <Tab> будет происходить переход от элемента к элементу соответственно табличным индексам (рис. 1.15).

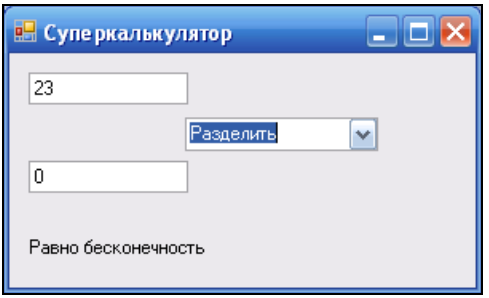


Рис. 1.15. Переход от одного текстового поля к другому

При обработке события "изменение индекса выбранного элемента" `comboBox1_SelectedIndexChanged` с помощью функции `TryParse` проверяем, можно ли текстовые поля преобразовать в число. Первым параметром метода `TryParse` является анализируемое поле. Второй параметр — это разрешаемый для преобразования стиль числа, в данном случае типа `Number`, т. е. десятичное число, которое имеет целую и дробную части. Третий параметр указывает, на какой основе формируется допустимый формат, в нашем случае мы использовали `CurrentInfo`, т. е. на основе текущего языка и региональных параметров. По умолчанию при инсталляции руссифицированной версии Windows разделителем целой и дробной частей числа является запятая. Однако эту установку можно изменить, если в Панели управления выбрать значок **Язык и региональные стандарты**, а затем на вкладке **Региональные параметры** щелкнуть на кнопке **Настройка** и на появившейся новой вкладке указать в качестве разделителя целой и дробной частей точку вместо запятой. В обоих случаях (и для запятой, и для точки) метод `TryParse` будет работать так, как указано на вкладке **Региональные параметры**.

Четвертый параметр метода `TryParse` возвращает результат преобразования. Кроме того, функция `TryParse` возвращает булеву переменную `true` или `false`, которая сообщает, успешно ли выполнено преобразование. Как видно из текста программы, если хотя бы одно поле невозможно преобразовать в число, то программируем сообщение "Следует вводить числа!" и выход из процедуры обработки события с помощью оператора `return`.

Далее оператор `switch` осуществляет множественный выбор арифметической операции в зависимости от индекса выбранного элемента списка `SelectedIndex`. Оператор `switch` передает управление той или иной "метке `case`". Причем, как говорят программисты, оператор множественного выбора в С-подобных языках в отличие от других языков, например Basic, "проваливается", т. е. управление переходит на следующую метку `case`, поэтому приходится использовать оператор `break` для выхода из `switch`.

Последний оператор в процедуре осуществляет формирование строки с помощью метода `string.Format` для вывода ее на метку `label1`. Формат `"{0:F5}"` означает, что значение переменной `z` следует выводить по фиксированному формату с *пятью знаками* после запятой (или точки). Заметьте, в этом примере даже не пришлось программировать событие деления на ноль. Система Visual Studio сделала это за нас (см. рис. 1.15).

Убедиться в работоспособности программы можно, открыв решение ComboBox_Calc.sln в папке ComboBox_Calc.

Пример 10. Вывод греческих букв, символов математических операторов. Кодовая таблица Unicode

Немного ликбеза. Хранение текстовых данных в памяти ЭВМ *предполагает кодирование символов по какому-либо принципу*. Таких кодировок несколько. Каждой кодировке соответствует своя таблица символов. В этой таблице каждой ячейке соответствуют номер в таблице и символ. Мы упомянем такие кодовые таблицы: ASCII, ANSI Cyrillic (другое название этой таблицы — Windows 1251), а также Unicode.

Первые две таблицы являются однобайтовыми, т. е. каждому символу соответствует 1 байт данных. Поскольку в 1 байте — 8 битов, байт может принимать $2^8 = 256$ различных состояний, этим состояниям можно поставить в соответствие 256 разных символов. Так, в таблице ASCII от 0 до 127 — базовая таблица — есть английские буквы, цифры, знаки препинания, управляющие символы. От 128 до 255 — это расширенная таблица, в ней находятся русские буквы и символы псевдографики. Некоторые из этих символов соответствуют клавишам IBM-совместимых компьютеров. Еще эту таблицу называют "ДОСовской" по имени операционной системы MS-DOS, где она применяется. Эта кодировка используется также в Интернете.

В операционной системе Windows используется преимущественно ANSI (Windows 1251). Базовые символы с кодами от 0 до 127 в этих таблицах совпадают, а расширенные — нет. То есть русские буквы в этих таблицах находятся в разных местах таблицы. Из-за этого бывают недоразумения. В ANSI нет символов псевдографики. ANSI Cyrillic — другое название кодовой таблицы Windows 1251.

Существует также двухбайтовый стандарт Unicode. Здесь один символ кодируется двумя байтами. Размер такой таблицы кодирования — $2^{16} = 65\,536$ ячеек. Кодовая таблица Unicode включает в себя практически все современные письменности. Разве что здесь нет старославянских букв. Когда в текстовом редакторе MS Word мы выполняем команду **Вставка | Символ**, то вставляем символ из таблицы Unicode. Также в Блокноте можно сохранять файлы в кодировке Unicode: **Сохранить как | Кодировка Юникод**. В этом случае в Блокноте будут, например,

греческие буквы, математические операторы ∏, Δ, ∑ и проч. Кстати, греческая буква Σ и математический оператор ∑ занимают разные ячейки в Unicode. Размер файла при сохранении в Блокноте будет ровно в два раза большим.

Напишем программу, которая приглашает пользователя ввести радиус *R*, чтобы вычислить длину окружности. При программировании этой задачи длину окружности в метке **Label** назовем греческой буквой β, приведем формулу для вычислений с греческой буквой π = 3,14. Результат вычислений выведем в диалоговое окно `MessageBox` также с греческой буквой.

После традиционного запуска Visual Studio 2010 и выбора шаблона **Windows Forms Application C#** перетащим в форму две метки **Label**, текстовое поле **TextBox** и командную кнопку **Button**. Посмотрите на рис. 1.16, так должна выглядеть форма после программирования этой задачи.

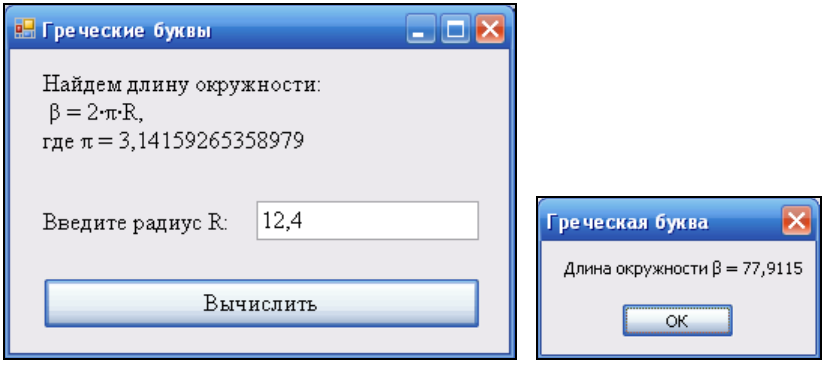


Рис. 1.16. Фрагмент работы программы, использующей символы Unicode

Вывод греческих букв на метку `label1` и в диалоговое окно `MessageBox` можно осуществить, например, таким путем. В текст программы через буфер обмена вставляем греческие буквы из текстового редактора MS Word. Поскольку по умолчанию Visual Studio 2010 сохраняет cs-файлы в формате Unicode, в принципе таким образом можно программировать вывод греческих букв и других символов на форму и на другие элементы управления.

Более технологично пойти другим путем, а именно будем вставлять подобные символы с помощью функции `Convert.ToChar`, а на вход этой функции будем подавать номер символа в таблице Unicode. В этом случае, даже если cs-файл будет сохранен в традиционной для Блокнота кодировке ANSI, программа будет работать корректно. Номер символа в таблице Unicode легко выяснить, выбрав в редакторе MS Word пункты меню **Вставка | Символ**. Здесь в таблице следует найти этот символ и соответствующий ему код знака в шестнадцатеричном представлении. Чтобы перевести шестнадцатеричное представление в десятичное, следует перед шестнадцатеричным числом поставить 0x. Например, после выполнения оператора `n = 0x3B2` в переменной `n` будет записано десятичное число 946. На этом месте в таблице Unicode расположена греческая буква β.

Именно таким образом запрограммирована данная задача (листинг 1.9).

Листинг 1.9. Использование символов Unicode

```
// Программа демонстрирует возможность вывода в текстовую метку, а также
// в диалоговое окно MessageBox греческих букв. Программа приглашает
// пользователя ввести радиус R, чтобы вычислить длину окружности
using System;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются
// в данной программе

namespace Unico
{
   public partial class Form1 : Form
   {
      public Form1()
      {  // Инициализация нового экземпляра класса System.Windows.Forms.Form
         InitializeComponent();
         base.Font = new System.Drawing.Font("Times New Roman", 12.0F);
         base.Text = "Греческие буквы"; button1.Text = "Вычислить";
         // бета = 2 x Пи x R
         label1.Text = string.Format(
            "Найдем длину окружности:\n {0} = 2{1}{2}{1}R,\nгде {2} = {3}",
            Convert.ToChar(0x3B2), Convert.ToChar(0x2219),
         //           0=бета             1 - точка
          Convert.ToChar(0x3C0), Math.PI);
         //           2 - Пи   3 - число Пи
         label2.Text = "Введите радиус R:";
         textBox1.Clear();
      }
      private void button1_Click(object sender, EventArgs e)
      {  // Проверка - число ли введено:
         Single R; // - радиус
         bool Число_ли = Single.TryParse(textBox1.Text,
               System.Globalization.NumberStyles.Number,
               System.Globalization.NumberFormatInfo.CurrentInfo, out R);
         if (Число_ли == false)
         {
            MessageBox.Show("Следует вводить числа!", "Ошибка",
               MessageBoxButtons.OK, MessageBoxIcon.Error); return;
         }
```

```
            Single beta = 2 * (Single)Math.PI * R;
            // 0x3B2 - греческая буква бета
            MessageBox.Show(String.Format("Длина окружности {0} = {1:F4}",
                            Convert.ToChar(0x3B2), beta), "Греческая буква");
        }
    }
}
```

Как видно из программного кода, сразу после вызова процедуры `InitializeComponent` мы задали шрифт Times New Roman, 12 пунктов для формы, этот шрифт будет распространяться на все элементы управления на форме, т. е. на текстовое поле, метку и командную кнопку. Далее, используя метод `String.Format`, инициализировали свойство `Text` метки `label1`. Различные шестнадцатеричные номера соответствуют греческим буквам и арифметической операции "умножить", в инициализации строки участвует также константа $\pi = 3{,}14$. Ее *более* точное значение получаем из `Math.PI`. Escape-последовательность `"\n"` используем для переноса текста на новую строку. Так называемый перевод каретки можно осуществить также с помощью строки `NewLine` из перечисления `Environment`.

Обрабатывая событие `button1_Click` (щелчок на кнопке), мы проверяем с помощью метода `TryParse`, число ли введено в текстовое поле. Если пользователь ввел число (`true`), то метод `TryParse` возвращает значение радиуса `R`. При вычислении длины окружности `beta` приводим значение константы `Math.PI` из типа `Double` к типу `Single` посредством неявного преобразования.

После вычисления длины окружности `beta` выводим ее значение вместе с греческой буквой β — `Convert.ToChar(0x3B2)` в диалоговое окно `MessageBox`. Здесь используем метод `String.Format`. Выражение `"{0:F4}"` означает, что значение переменной `beta` следует выводить по фиксированному формату с четырьмя знаками после запятой.

Данная программа будет корректно отображать греческие буквы, даже если открыть файл Form1.cs текстовым редактором Блокнот и сохранить его в кодировке ANSI. Убедиться в работоспособности программы можно, открыв решение Unico.sln в папке Unico.

Глава 2

Программирование консольных приложений

Пример 11. Ввод и вывод в консольном приложении

Иногда, например, для научных расчетов, требуется организовать какой-нибудь самый простой ввод данных, выполнить, может быть, весьма сложную математическую обработку введенных данных и оперативно вывести на экран результат вычислений.

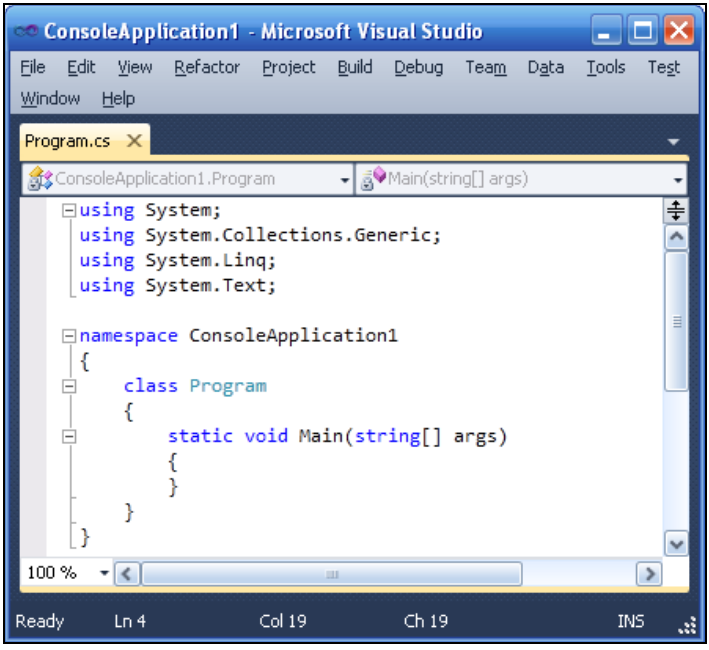


Рис. 2.1. Вкладка программного кода

Такая же ситуация возникает тогда, когда большая программа отлаживается по частям. И для отладки вычислительной части совершенно не важен сервис при вводе данных.

Можно по-разному организовать такую программу, в том числе программируя так называемое *консольное приложение* (от англ. *console* — пульт управления). Под консолью обычно подразумевают экран компьютера и клавиатуру.

Для примера напишем консольное приложение, которое приглашает пользователя ввести два числа, складывает их и выводит результат вычислений на консоль.

Для этого запускаем Visual C# 2010, далее создаем новый проект (**New Project**), выбираем шаблон **C# Console Application**, задаем имя решения (**Name**) — Сумма. После щелчка на кнопке **OK** попадаем сразу на вкладку программного кода (рис. 2.1).

Как видите, здесь управляющая среда Visual Studio 2010 приготовила несколько строк программного кода. При запуске консольного или Windows-приложения C# метод `Main()` является первым вызываемым методом. В фигурных скобках после `Main()` мы вставим собственный программный код (листинг 2.1).

Листинг 2.1. Ввод и вывод данных в консольном приложении

```
// Эта программа складывает два числа и выводит сумму на консоль
using System;
using System.Text;
// Другие директивы using удалены, поскольку они не используются
// в данной программе

namespace Сумма
{
   class Program
   {
      static void Main(string[] args)
      {  // Задаем строку заголовка консоли:
         Console.Title = "Складываю два числа:";
         Console.BackgroundColor = ConsoleColor.Cyan;  // - цвет фона
         Console.ForegroundColor = ConsoleColor.Black; // - цвет текста
         Console.Clear();

         Console.WriteLine("Введите первое слагаемое:");
         String Строка = Console.ReadLine();

         Single X, Y, Z;
         X = Single.Parse(Строка); // - преобразование строковой
                                   //   переменной в число
         Console.WriteLine("Введите второе слагаемое:");
```

```
            Строка = Console.ReadLine();
            Y = Single.Parse(Строка);

            Z = X + Y;

            Console.WriteLine("Сумма = {0} + {1} = {2}", X, Y, Z);
            // Звуковой сигнал частотой 1000 Гц и длительностью 0.5 секунд:
            Console.Beep(1000, 500);
            // Приостановить выполнение программы до нажатия
            // какой-нибудь клавиши:
            Console.ReadKey();
        }
    }
}
```

Итак, в данной программе `Main()` — это стартовая точка, с которой начинается ее выполнение. Обычно консольное приложение выполняется в окне на черном фоне. Чтобы как-то украсить традиционно черное окно консольного приложения, установим цвет фона окна `BackgroundColor` сине-зеленым (`Cyan`), а цвет символов, выводимых на консоль, черным (`Black`). Выводим строки в окно консоли методом `WriteLine`, а для считывания строки символов, вводимых пользователем, используем метод `ReadLine`. Далее объявляем три переменных типа `Single` для соответственно первого числа, второго и значения суммы. Тип данных `Single` применяется тогда, когда число, записанное в переменную, может иметь целую и дробную части. Переменная типа `Single` занимает 4 байта. Для преобразования строки символов, введенных пользователем в числовое значение, используем метод `Parse`.

После вычисления суммы необходимо вывести результат вычислений из оперативной памяти на экран. Для этого воспользуемся форматированным выводом в фигурных скобках метода `WriteLine` объекта `Console`:

```
Console.WriteLine("Сумма = {0} + {1} = {2}", X, Y, Z)
```

Затем выдаем звуковой сигнал Beep, символизирующий об окончании процедуры и выводе на экран результатов вычислений. Последняя строка в программе `Console.ReadKey();` предназначена для приостановки выполнения программы до нажатия какой-нибудь клавиши. Если не добавить эту строку, окно с командной строкой сразу исчезнет и пользователь не сможет увидеть вывод результатов выполнения. Программа написана. Нажмите клавишу <F5>, чтобы увидеть результат. Фрагмент работы данной программы представлен на рис. 2.2.

При организации научных расчетов или в ситуации, когда необходимо отладить расчетную часть большой программы, когда сервис при вводе данных вообще не имеет значения, можно просто присваивать значения переменным при их объявлении. Очень технологичным является вариант, когда данные записываются в текстовый файл с помощью, например, Блокнота (notepad.exe), а в программе предусмотрено чтение текстового файла в оперативную память.

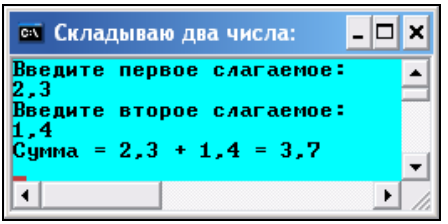


Рис. 2.2. Фрагмент работы консольного приложения

Убедиться в работоспособности программы можно, открыв решение Сумма.sln в папке Сумма.

Пример 12. Вывод на консоль таблицы чисел с помощью форматирования строк

Пользуясь тем, что шрифт вывода на консоль является моноширным, т. е. каждый выводимый символ (например, точка и заглавная буква "Ш") имеет одинаковую ширину, в данном примере покажем, как путем форматирования строк можно вывести таблицу в окно консоли. В этой программе выведем в таблицу извлеченный квадратный корень в цикле от нуля до десяти. Для этого запустим Visual Studio 2010, закажем новый проект шаблона **C# Console Application**, зададим имя решения (**Name**) — ТаблКорней и на вкладке программного кода введем программный код, представленный в листинге 2.2.

Листинг 2.2. Вывод таблицы извлечения квадратного корня на консоль

```
// Консольное приложение задает цвета и заголовок консоли, а затем выводит
// таблицу извлечения квадратного корня от нуля до десяти
using System;
// Другие директивы using удалены, поскольку они не используются
// в данной программе

namespace ТаблКорней
{
   class Program
   {
      static void Main(string[] args)
      {
         Console.BackgroundColor = ConsoleColor.Cyan;
         Console.ForegroundColor = ConsoleColor.Black;
         Console.Title = "Таблица"; Console.Clear();
```

```
            Console.WriteLine("Число  Корень");
            for (double i = 0; i <= 10; i++)
            {
               Console.WriteLine("{0,4} {1,8:F4}", i, Math.Sqrt(i));
            }
            Console.ReadKey();
        }
    }
}
```

Как видно из программного кода, в пошаговом цикле `for` для вывода строк используется метод `WriteLine`, на вход которого подают строку в определенном формате. Причем форматирование производится по тем же правилам, что и в очень часто используемом методе `String.Format`, предназначенном для обработки строк. Использованный формат `"{0,4} {1,8:F4}"` означает: взять нулевой выводимый элемент, т. е. счетчик цикла `i`, и записать его с выравниванием вправо в четырех столбцах, после чего взять первый выводимый элемент, т. е. значение квадратного корня из `i`, и записать его с выравниванием вправо в следующих восьми столбцах в формате с фиксированной точкой и четырьмя десятичными знаками после запятой. В результате работы данной программы получим таблицу извлечения квадратного корня (рис. 2.3).

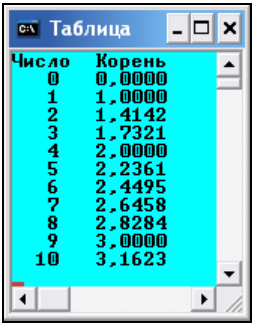


Рис. 2.3. Вывод на консоль таблицы

Подобным форматированием мы будем обрабатывать строки в последующих примерах данной книги, пользуясь методом `String.Format`.

Убедиться в работоспособности программы можно, открыв решение ТаблКорней.sln в папке ТаблКорней.

Пример 13. Вызов *MessageBox.Show* в консольном приложении. Формат даты и времени

Программирование консольного приложения напоминает период конца 1980-х годов, когда появились первые персональные компьютеры с очень слабой произво-

дительностью и небольшой памятью. Для вывода данных Visual C# имеет удобное средство `MessageBox.Show`, и в консольном приложении его вызвать все-таки можно. Покажем на следующем примере, как практически это можно сделать.

Итак, в данном примере следует в консольном приложении вывести в окно `MessageBox` текущую дату и время в различных форматах, чтобы попутно продемонстрировать читателю некоторые возможности форматирования строк с помощью метода `String.Format`. Запустим Visual Studio 2010, выберем шаблон консольного приложения C#. Далее необходимо в проект *добавить ссылку на библиотеку*, содержащую объект `MessageBox`. Для этого в пункте меню **Project** выберем команду **Add Reference**, а на вкладке **.NET** дважды щелкнем на ссылке **System.Windows.Forms**. Теперь в окне **Solution Explorer** (Обозреватель решений), раскрывая пункт **References** (Ссылки), увидим добавленную в наш проект выбранную ссылку. В программном коде вставим строку `using System.Windows.Forms`. Директива `using` используется для импортирования пространства имен, которое содержит класс `MessageBox` (листинг 2.3).

Листинг 2.3. Вызов `MessageBox.Show` в консольном приложении

```
// Консольное приложение выводит в окно MessageBox текущую дату
// и время в различных форматах, используя String.Format
using System;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace ConsoleMessageBox
{
   class Program
   {
      static void Main(string[] args)
      {
         string ДатаВремя = string.Format(
     "(d) - это формат \"короткой\" даты: . . . . . {0:d}\n" +
     "(D) - это формат \"полной\" даты:. . . . . .{0:D}\n" +
     "(t) - это формат \"короткого\" времени: . . . . . {0:t}\n" +
     "(T) - это формат \"длинного\" времени:. . . . . {0:T}\n" +
     "(f) - выводится \"полная\" дата и \"короткое\" время: {0:f}\n" +
     "(F) - выводится \"полная\" дата и \"длинное\" время:. {0:F}\n" +
     "(g) General - короткая дата и короткое время: . . {0:g}\n" +
     "(G) General - \"общий\" формат: . . . . . {0:G}\n" +
     "Пустой формат - такой же, как формат (G). . . . . {0}\n" +
     "(M) - выводится только месяц и число: . . . . . {0:M}\n" +
     "(U) Universal full date/time - время по Гринвичу. . {0:U}\n" +
```

```
        "(Y) - по этому формату выводится только год. . . . . . . {0:Y}\n",
                                        DateTime.Now);
            MessageBox.Show(ДатаВремя, "Время и дата в различных форматах");
        }
    }
}
```

Запустим наше приложение, нажав клавишу <F5>. В результате появится черное окно консоли, а под ним — диалоговое окно `MessageBox` (рис. 2.4).

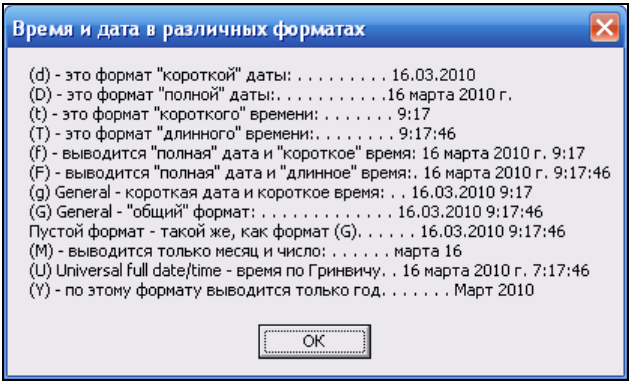


Рис. 2.4. Вывод текущего времени и даты в различных форматах

Теперь избавимся от мешающего нам черного окна консоли. Для этого в окне свойств проекта (**Project | Properties**) на вкладке **Application** в раскрывающемся списке **Output type** щелкнем на пункте **Windows Application**.

Убедиться в работоспособности программы можно, открыв соответствующее решение в папке ConsoleMessageBox.

Пример 14. Вызов функций Visual Basic из программы C#

Покажем, как можно воспользоваться в вашей C#-программе методами другой программной среды, в частности функциями Visual Basic. Решим следующую задачу. Программа приглашает пользователя ввести одно число, затем другое, анализирует, ввел ли пользователь именно числа, а не другие символы, и выводит результат суммирования в диалоговое окно. Программировать будем в консольном приложении C#, при этом воспользуемся функцией ввода `InputBox`, функцией вывода `MsgBox`, а также функцией `IsNumeric`, которая определяет, число ли подано на вход этой функции. Все эти три функции были еще в Visual Basic 6.0.

Для решения этой задачи запустим Visual Studio, выберем проект шаблона консольного приложения C#. Далее необходимо в проект *добавить ссылку на библиотеку* Microsoft.VisualBasic.dll. Для этого в пункте меню **Project** выберем команду **Add Reference**, а на вкладке **.NET** дважды щелкнем на ссылке **Microsoft.VisualBasic**. Теперь в окне **Solution Explorer**, раскрывая пункт **References**, увидим добавленную в наш проект выбранную ссылку. Затем перейдем на вкладку программного кода и введем текст, представленный в листинге 2.4.

Листинг 2.4. Программный код с вызовом функций Visual Basic

```
// В данном консольном приложении Visual C# используем функции Visual Basic.
// Приложение приглашает пользователя ввести два числа, анализирует,
// числа ли ввел пользователь, и выводит результат суммирования на экран.
// При этом используем функции Visual Basic: InputBox,
// IsNumeric (для контроля, число ли ввел пользователь) и MsgBox.

using System;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace VisualBasic1
{
   class Program
   {  // Для вызова функций из Visual Basic добавим ссылку в текущий проект.
      // Для этого в пункте меню Project щелкнем мышью команду
      // Add Reference и в диалоговом окне на вкладке .NET выберем
      // элемент Microsoft.VisualBasic
      static void Main(string[] args)
      {
         string Строка;
         // Бесконечный цикл, пока пользователь не введет именно число:
         for (; ; )
         {
            Строка = Microsoft.VisualBasic.Interaction.
               InputBox("Введите первое число:", "Складываю два числа");
            if (Microsoft.VisualBasic.Information.IsNumeric(Строка) == true)
                break;
         }
         // - преобразование строковой переменной в число:
         Single X = Single.Parse(Строка);
         // Ввод второго числа:
         for (; ; )
         {
            Строка = Microsoft.VisualBasic.Interaction.
```

```
                InputBox("Введите второе число:", "Складываю два числа");
            if (Microsoft.VisualBasic.Information.IsNumeric(Строка) == true)
                break;
        }
        Single Y = Single.Parse(Строка);
        Single Z = X + Y;

        Строка = string.Format("Сумма = {0} + {1} = {2}", X, Y, Z);
        // Вывод результата вычислений на экран:
        Microsoft.VisualBasic.Interaction.MsgBox(Строка);
    }
  }
}
```

Ввод первого числа организуем с помощью бесконечного цикла `for(;;){}`. Функция `InputBox` возвращает строку, введенную пользователем в диалоговом окне (рис. 2.5).

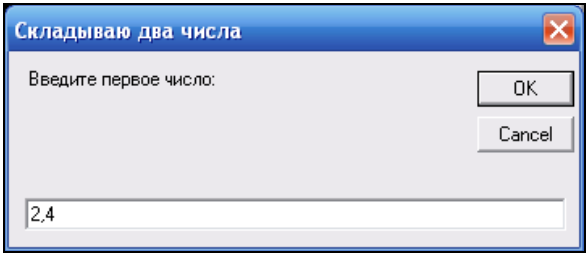


Рис. 2.5. Диалоговое окно ввода данных

Эта строка анализируется функцией `IsNumeric`, и если в строке записано число, то происходит выход `break` из вечного цикла. Аналогично организован ввод второго числа. Для преобразования строки символов, введенных пользователем, в числовое значение используем метод `Parse`. Кстати, для преобразования строки символов в числовое значение платформа .NET Framework содержит в себе метод `TryParse`, который более эффективно решает подобные задачи по проверке типа введенных данных и преобразованию их в числовое значение.

После вычисления суммы необходимо вывести результат вычислений из оперативной памяти в диалоговое окно `MsgBox`. Для этого вначале подготовим строку вывода с помощью метода `String.Format`. Выражение `"Сумма = {0} + {1} = {2}"` означает: взять нулевой выводимый элемент, т. е. переменную `X`, и записать ее вместо первых фигурных скобок, после чего взять первый выводимый элемент, т. е. переменную `Y`, и записать ее вместо вторых фигурных скобок, аналогично — для третьего выводимого элемента.

Результат работы программы представлен на рис. 2.6.

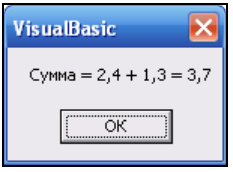


Рис. 2.6. Диалоговое окно вывода данных

Каждый раз, запуская это приложение, нам приходится искать диалоговые окна под черным окном консоли. Чтобы избавиться от этого окна, в окне свойств проекта (**Project | Properties**) на вкладке **Application** в раскрывающемся списке **Output type** щелкнем на пункте **Windows Application**.

Убедиться в работоспособности программы можно, открыв соответствующее решение в папке СсылкаНаVisualBasic.

Глава 3

Инициирование и обработка событий мыши и клавиатуры

Пример 15. Координаты курсора мыши относительно экрана и элемента управления

Напишем программу, которую мы условно называем *мониторингом положения мыши*. Имеем форму, список элементов **ListBox** и два текстовых поля. Они расположены в форме так, как показано на рис. 3.1.

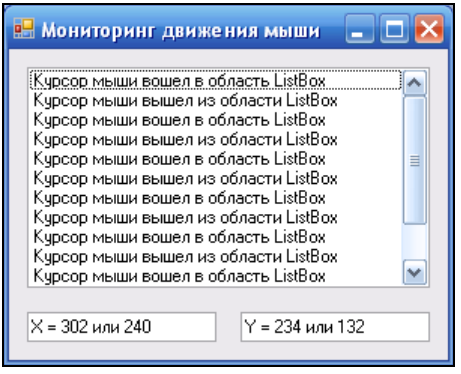


Рис. 3.1. Фрагмент работы программы определения координат курсора мыши

Программа заполняет список **ListBox** данными о местоположении и изменении положения курсора мыши. Кроме того, в текстовых полях отображаются координаты положения курсора мыши относительно экрана, а также относительно элемента управления **ListBox**.

Для программирования этой задачи после запуска Visual Studio 2010 и выбора шаблона **Windows Forms Application C#** из панели элементов управления **Toolbox** перетащим в форму элемент управления **ListBox** и два текстовых поля. В данной

программе нам понадобится обработать три события, относящиеся к объекту `listBox1`. Чтобы получить три соответствующих пустых обработчика в программном коде, следует в конструкторе формы в панели свойств (**Properties**) щелкнуть на пиктограмме молнии и в появившемся списке возможных событий для объекта `listBox1` выбрать следующие три события: `MouseEnter`, `MouseLeave` и `MouseMove`. Соответственно получим три пустых обработчика событий (листинг 3.1).

Листинг 3.1. Координаты курсора мыши относительно экрана и элемента управления

```
// Программа отображает координаты курсора мыши относительно экрана и элемента
// управления. Программа содержит форму, список элементов ListBox и два
// текстовых поля. Программа заполняет список ListBox данными о местоположении
// и изменении положения курсора мыши. Кроме того, в текстовых полях
// отображаются координаты положения курсора мыши относительно экрана
// и элемента управления ListBox.
using System;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace Monitoring
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
         base.Text = "Мониторинг движения мыши";
      }

      // Процедура обработки события, когда указатель мыши оказывается
      // на элементе управления ListBox
      private void listBox1_MouseEnter(object sender, EventArgs e)
      {  // Добавляем в список элементов новую запись
         listBox1.Items.Add("Курсор мыши вошел в область ListBox");
      }

      // Процедура обработки события, когда указатель мыши покидает
      // элемент управления ListBox
      private void listBox1_MouseLeave(object sender, EventArgs e)
      {
         listBox1.Items.Add("Курсор мыши вышел из области ListBox");
      }
```

```
        // Процедура обработки события, происходящего при перемещении
        // указателя мыши по элементу управления ListBox
        private void listBox1_MouseMove(object sender, MouseEventArgs e)
        {  // Свойство объекта Control MousePosition возвращает точку,
           // соответствующую текущему положению мыши относительно
           // левого верхнего угла монитора
           textBox1.Text = string.Format("X = {0} или {1}",
                                         Form1.MousePosition.X, e.X);
           textBox2.Text = string.Format("Y = {0} или {1}",
                                         Form1.MousePosition.Y, e.Y);
        }
    }
}
```

Как видно, при обработке события мыши `MouseEnter`, когда курсор мыши входит в границы элемента управления, в список `ListBox1` *добавляется* (метод `Add`) *запись* "Курсор мыши вошел в область ListBox". При обработке события мыши `MouseLeave`, когда курсор мыши выходит за пределы элемента управления, в список `ListBox` добавляется запись "Курсор мыши вышел из области ListBox". Таким образом, отслеживая поведение мыши, мы заполняем список `ListBox1`.

При обработке события `MouseMove`, когда курсор мыши перемещается в пределах элемента управления `ListBox1`, в текстовые поля записываем координаты *X* и *Y* курсора мыши, пользуясь свойством объекта `Control MousePosition`. Здесь мы получаем координаты положения курсора мыши в системе координат экрана, когда начало координат расположено в левом верхнем углу экрана, ось *x* направлена вправо, а ось *y* — вниз.

Заметим, что аргументы события мыши `e` также содержат в себе текущие координаты курсора мыши, но в системе координат элемента управления, в данном случае `listBox1`. Начало координат этой системы расположено в левом верхнем углу элемента управления `listBox1`, ось *x* также направлена вправо, ось *y* — вниз. Эти координаты получаем из аргументов события `e.X` и `e.Y` и выводим их в текстовое поле, отделяя от предыдущих координат словом "или".

Таким образом, добиваемся *контроля положения курсора мыши*, обрабатывая события мыши. Убедиться в работоспособности программы можно, открыв решение Monitoring.sln папки Monitoring.

Пример 16. Создание элемента управления *Button* "программным" способом и подключение события для него

Мы знаем, как, используя панель элементов управления, в форму перенести мышью нужный элемент. Чтобы сделать разработку программы более управляе-

мой, в данной программе научимся создавать элементы управления в форме "программным" способом, т. е. с помощью написания непосредственно программного кода, не используя при этом панель элементов управления **Toolbox**. Понятно, что название "программным" является более чем условным, поскольку в описываемой нами среде трудно назвать что-либо, что "программным" не является.

Итак, данная программа создаст командную кнопку в форме "программным" способом, задаст свойства кнопки: ее видимость, размеры, положение, надпись на кнопке и подключит событие "щелчок на кнопке".

Для этого создаем новый проект с формой. При этом, как обычно, запускаем Visual Studio 2010, в окне **New Project** выбираем шаблон **Windows Forms Application C#**. Далее вводим программный код, представленный в листинге 3.2.

Листинг 3.2. Создание кнопки "программным" способом

```
// Программа создает командную кнопку в форме "программным" способом,
// т. е. с помощью написания непосредственно программного кода, не используя
// при этом панель элементов управления Toolbox. Программа задает свойства
// кнопки: ее видимость, размеры, положение, надпись на кнопке
// и подключает событие "щелчок на кнопке"
using System;
using System.Drawing;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace NewButton
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
         // Создание кнопки без панели элементов управления:
         Button button1 = new Button();
         // Задаем свойства кнопки:
         button1.Visible = true;
         // Ширина и высота кнопки:
         button1.Size = new Size(100, 30);
         // Расположение кнопки в системе координат формы:
         button1.Location = new Point(100, 80);
         button1.Text = "Новая кнопка";
         // Добавление кнопки в коллекцию элементов управления
         this.Controls.Add(button1);
         // Подписку на событие Click для кнопки можно делать "вручную".
```

```
            // Связываем событие Click с процедурой обработки этого события:
            button1.Click += new EventHandler(button1_Click);
        }
        // Создаем обработчик события Click для кнопки:
        private void button1_Click(object sender, EventArgs e)
        {
            MessageBox.Show("Нажата новая кнопка");
        }
    }
}
```

Как видно, сразу после выполнения процедуры `InitializeComponent` создаем новый объект `button1` стандартного класса кнопок. Задаем свойства кнопки: ее видимость (`Visible`), размеры (`Size`), положение (`Location`) относительно левого нижнего угла формы, надпись на кнопке — "Новая кнопка".

Далее необходимо организовать корректную работу с событием "щелчок на созданной нами командной кнопке". В предыдущих примерах мы для этой цели в конструкторе формы дважды щелкали на проектируемой кнопке, и исполняемая среда автоматически генерировала пустой обработчик этого события в программном коде. Или опять же в конструкторе формы в панели свойств проектируемой кнопки щелкали мышью на значке молнии (**Events**) и в появившемся списке всех событий выбирали необходимое событие. Однако согласно условию задачи мы должны организовать обработку события "программным" способом без использования конструктора формы. Для этого в программном коде сразу после добавления командной кнопки в коллекцию элементов управления ставим оператор "точка" (`.`) после имени кнопки `button1` и в выпадающем списке выбираем необходимое событие `Click`. Затем, как приведено в листинге 3.2, осуществляем так называемую "подписку" на данное событие, т. е. с помощью ключевого слова `EventHandler` связываем событие `Click` с процедурой обработки события `button1_Click`. Такое название процедуры обработки события принято в среде Visual Studio, но не является обязательным, мы могли бы назвать эту процедуру, например, русскими буквами: "КЛИК". Теперь создадим обработчик события `button1_Click`, как показано в листинге 3.2. В этой процедуре предусматриваем вывод сообщения "Нажата новая кнопка". На рис. 3.2 приведен фрагмент работы программы.

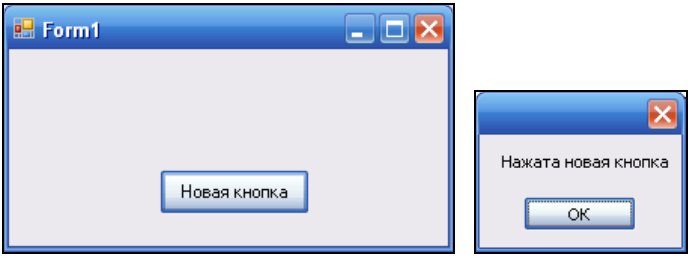


Рис. 3.2. Создание кнопки программным способом

В заключение отметим, что в случае создания пустого обработчика события в конструкторе формы строка подписки на событие формируется автоматически в методе `InitializeComponent` в файле Form1.Designer.cs проекта. Убедиться в работоспособности программы можно, открыв решение NewButton.sln папки NewButton.

Пример 17. Обработка нескольких событий одной процедурой

Для того чтобы события от нескольких элементов управления обрабатывались одной процедурой обработки события, в некоторых языках, например в VB6, было предусмотрено создание массива элементов управления. Однако в современных языках Visual Studio элементы *не могут быть сгруппированы в массивы*. Но можно организовать обработку нескольких событий одной процедурой путем подписки этих событий на одну и ту же процедуру их обработки.

Как это сделать, покажем на примере, когда в форме присутствуют две командные кнопки, и щелчок на любой из них обрабатывается одной процедурой. При этом, используя параметр процедуры `sender`, будем определять, на какой из двух кнопок щелкнули мышью.

Итак, запустим Visual Studio 2010, закажем новый проект шаблона **Windows Forms Application C#**. Затем из панели элементов перенесем в форму две командных кнопки и текстовую метку. Далее через щелчок правой кнопкой мыши перейдем к вкладке программного кода (листинг 3.3).

Листинг 3.3. Связывание двух событий с одной процедурой обработки

```
// В форме имеем две командные кнопки, и при нажатии указателем мыши
// любой из них получаем номер нажатой кнопки. При этом в программе
// предусмотрена только одна процедура обработки событий
using System;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace ДваСобытияОднаПроц
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
         base.Text = "Щелкните на кнопке";
```

```
            // Подписку на событие можно делать "вручную", но при этом в
            // Form1.Designer.cs следует удалить (или закомментировать)
            // соответствующие EventHandler.
            // Связываем события Click от обеих кнопок с одной процедурой КЛИК:
            button1.Click += new EventHandler(this.КЛИК);
            button2.Click += new EventHandler(this.КЛИК);
            label1.Text = null;
            // Подпиской на событие называют связывание названия события
            // с названием процедуры обработки события посредством EventHandler
        }
        private void КЛИК(object sender, EventArgs e)
        {
            // String S = Convert.ToString(sender);
            // получить текст, отображаемый на кнопке, можно таким образом:
            Button Кнопка = (Button)sender;
            // или string НадписьНаКнопке = ((Button)sender).Text;
            label1.Text = "Нажата кнопка " + Кнопка.Text; // или Кнопка.Name
        }
    }
}
```

Как видно из текста программы, сразу после выполнения процедуры `InitializeComponent` осуществляем так называемую подписку на событие, т. е. связываем название события с названием процедуры обработки события `КЛИК` посредством делегата `EventHandler`. Заметим, что оба события `Click` мы связали с одной и той же процедурой `КЛИК`.

Далее создаем процедуру обработки события `КЛИК`, ее параметр `sender` содержит *ссылку на объект-источник события*, т. е. кнопку, нажатую пользователем. С помощью неявного преобразования можно конвертировать параметр `sender` в экземпляр класса `Button` и, таким образом, выяснить все свойства кнопки, которая инициировала событие.

На рис. 3.3 приведен пример работы написанной программы.

Мы убедились в этом разделе, что сведения об объекте, который создал событие, находятся в объектной переменной `sender`. Работу этой программы можно исследовать, открыв соответствующее решение в папке ДваСобытияОднаПроц.

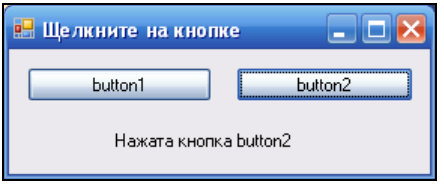


Рис. 3.3. Фрагмент работы программы, определяющей нажатую кнопку

Пример 18. Калькулятор

Обработка *нескольких событий от разных объектов одной процедурой* оказывается весьма полезной при программировании данного приложения. Напишем программу Калькулятор с кнопками-цифрами, выполняющую только арифметические операции, причем управление Калькулятором возможно только мышью.

Запустив Visual Studio 2010 и выбрав шаблон **Windows Forms Application C#**, перетащим из панели **Toolbox** в форму 16 командных кнопок для ввода цифр, арифметических операций, знака "равно" (=) и операции **Очистить**, а также текстовое поле. Вкладка **Form1.cs [Design]** будет иметь примерно такой вид, как показано на рис. 3.4.

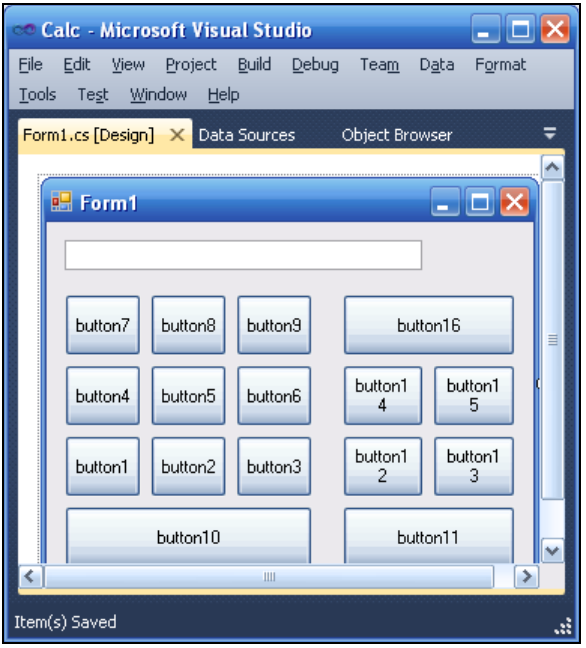


Рис. 3.4. Вкладка конструктора формы

В листинге 3.4 приведен программный код данного приложения.

Листинг 3.4. Калькулятор

```
// Программа Калькулятор с кнопками цифр. Управление калькулятором возможно
// только мышью. Данный калькулятор выполняет лишь арифметические операции
using System;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
```

```
namespace Calc
{
   public partial class Form1 : Form
   {
      string Znak = null;       // знак арифметической операции
      bool Начало_Ввода = true; // ожидание ввода нового числа
      Double Число1, Число2; // Первое и второе числа, вводимые пользователем
      public Form1()
      {
         InitializeComponent();
         this.Text = "Калькулятор";
         button1.Text = "1";  button2.Text = "2";  button3.Text = "3";
         button4.Text = "4";  button5.Text = "5";  button6.Text = "6";
         button7.Text = "7";  button8.Text = "8";  button9.Text = "9";
         button10.Text = "0"; button11.Text = "="; button12.Text = "+";
         button13.Text = "-"; button14.Text = "*"; button15.Text = "/";
         button16.Text = "Очистить";
         textBox1.Text = "0";
         textBox1.TextAlign = HorizontalAlignment.Right;
         // Связываем все события "щелчок на кнопках-цифрах"
         // с обработчиком ЦИФРА:
         this.button1.Click += new System.EventHandler(this.ЦИФРА);
         this.button2.Click += new System.EventHandler(this.ЦИФРА);
         this.button3.Click += new System.EventHandler(this.ЦИФРА);
         this.button4.Click += new System.EventHandler(this.ЦИФРА);
         this.button5.Click += new System.EventHandler(this.ЦИФРА);
         this.button6.Click += new System.EventHandler(this.ЦИФРА);
         this.button7.Click += new System.EventHandler(this.ЦИФРА);
         this.button8.Click += new System.EventHandler(this.ЦИФРА);
         this.button9.Click += new System.EventHandler(this.ЦИФРА);
         this.button10.Click += new System.EventHandler(this.ЦИФРА);
         this.button12.Click += new System.EventHandler(this.ОПЕРАЦИЯ);
         this.button13.Click += new System.EventHandler(this.ОПЕРАЦИЯ);
         this.button14.Click += new System.EventHandler(this.ОПЕРАЦИЯ);
         this.button15.Click += new System.EventHandler(this.ОПЕРАЦИЯ);
         this.button11.Click += new System.EventHandler(this.РАВНО);
         this.button16.Click += new System.EventHandler(this.ОЧИСТИТЬ);
      }
      private void ЦИФРА(object sender, EventArgs e)
      {  // Обработка события нажатия кнопки-цифры.
         // Получить текст, отображаемый на кнопке, можно таким образом:
         Button Кнопка = (Button)sender;
```

```
        String Digit = Кнопка.Text;
        if (Начало_Ввода == true)
        {  // Ввод первой цифры числа:
           textBox1.Text = Digit;
           Начало_Ввода = false; return;
        }
        // "Сцепливаем" полученные цифры в новое число:
        if (Начало_Ввода == false) textBox1.Text = textBox1.Text + Digit;
     }
     private void ОПЕРАЦИЯ(object sender, EventArgs e)
     {  // Обработка события нажатия кнопки арифметической операции:
        Число1 = Double.Parse(textBox1.Text);
        // Получить текст, отображаемый на кнопке, можно таким образом:
        Button Кнопка = (Button)sender;
        Znak = Кнопка.Text;
        Начало_Ввода = true; // ожидаем ввод нового числа
     }
     private void РАВНО(object sender, EventArgs e)
     {  // Обработка нажатия клавиши "равно"
        double Результат = 0;
        Число2 = Double.Parse(textBox1.Text);
        if (Znak == "+") Результат = Число1 + Число2;
        if (Znak == "-") Результат = Число1 - Число2;
        if (Znak == "*") Результат = Число1 * Число2;
        if (Znak == "/") Результат = Число1 / Число2;
        Znak = null;
        // Отображаем результат в текстовом поле:
        textBox1.Text = Результат.ToString();
        Число1 = Результат; Начало_Ввода = true;
     }
     private void ОЧИСТИТЬ(object sender, EventArgs e)
     {  // Обработка нажатия клавиши "Очистить"
        textBox1.Text = "0"; Znak = null; Начало_Ввода = true;
     }
  }
}
```

В этой книге мы принципиально отказались от задания свойств элементов управления и формы в окне **Properties**, чтобы не потеряться в огромном количестве этих свойств и не забывать, какое свойство мы изменили, а какое нет. Исходя из этих соображений, автор задал все свойства объектов сразу после выполнения процедуры `InitializeComponent`. Именно здесь заданы надписи на кнопках,

ноль в текстовом поле, причем этот ноль прижат к правому краю поля: `textBox1.TextAlign = HorizontalAlignment.Right`.

Далее связываем все события `Click` от кнопок-цифр с одной процедурой обработки этих событий `ЦИФРА`. Аналогично все события `Click` от кнопок арифметических операций связываем с одной процедурой `ОПЕРАЦИЯ`.

В процедуре обработки события щелчок на любой из кнопок-цифр `ЦИФРА` в строковую переменную `Digit` копируем цифру, изображенную на кнопке из свойства `Text` так, как мы это делали в предыдущем примере, когда отлавливали нажатие пользователем одной из двух кнопок. Далее необходимо значение `Digit` присвоить свойству `textBox1.Text`, но здесь изначально записан ноль. Если пользователь вводит первую цифру, то вместо нуля нужно записать эту цифру, а если пользователь вводит последующие цифры, то их надо "сцепить" вместе. Для управления такой ситуацией мы ввели булеву (логическую) переменную `Начало_Ввода`. Мы сознательно назвали эту переменную по-русски, *чтобы она выделялась среди прочих переменных*, ведь она играет ключевую роль в программе и участвует в обработке практически *всех событий*. Поскольку мы ввели ее в начале программы, область действия этой переменной — весь класс `Form1`, т. е. эта переменная *"видна"* в процедурах обработки всех событий.

То есть различаем начало ввода числа `Начало_Ввода = true`, когда ноль следует менять на вводимую цифру, и последующий ввод `Начало_Ввода = false`, когда очередную цифру следует добавлять справа. Таким образом, если это уже не первая нажатая пользователем кнопка-цифра (`Начало_Ввода = false`), то "сцепливаем" полученную цифру с предыдущими введенными цифрами, иначе — просто запоминаем первую цифру в текстовом поле `textBox1`.

При обработке событий "щелчок указателем мыши по кнопкам" арифметических операций `+`, `-`, `*`, `/` в процедуре `ОПЕРАЦИЯ` преобразуем первое введенное пользователем число из текстового поля в переменную `value1` типа `Double`. Строковой переменной `Znak` присваивается символьное представление арифметической операции. Поскольку пользователь нажал кнопку арифметической операции, ожидаем, что следующим действием пользователя будет ввод очередного числа, поэтому присваиваем булевой переменной `Начало_Ввода` значение `true`. Заметьте, что обрабатывая два других события: нажатие кнопки "равно" и нажатие кнопки **Очистить**, мы также устанавливаем логическую переменную `Начало_Ввода` в состояние `true` (т. е. начинаем ввод числа).

В процедуре обработки события нажатия кнопки "равно" преобразуем второе введенное пользователем число в переменную типа `Double`. Теперь, поскольку знак арифметической операции нам известен, известны также оба числа, мы можем выполнить непосредственно арифметическую операцию. После того как пользователь получит результат, например `result = value1 + value2`, возможно, он захочет с этим результатом выполнить еще какое-либо действие, поэтому этот результат записываем в первую переменную `value1`. Заметьте, что в этой программе мы сознательно не предусмотрели обработку исключительной ситуации *деления на ноль*, поскольку среда Visual Studio 2010 (впрочем, как и ее предыдущие версии) взяла на

себя обработку этой ситуации. Когда в строковую переменную попадает очень большое число, в эту переменную система пишет слово "бесконечность" (рис. 3.5).

Убедиться в работоспособности программы можно, открыв решение Calc.sln в папке Calc.

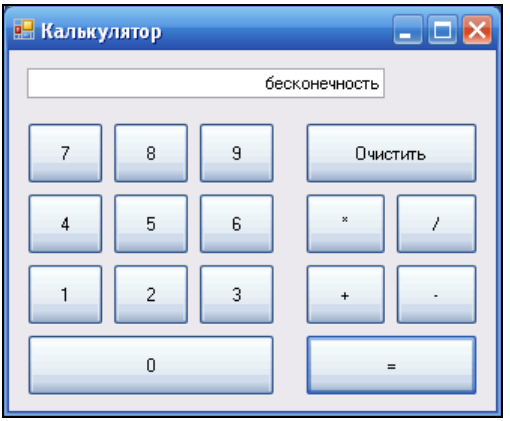


Рис. 3.5. Фрагмент работы калькулятора

Пример 19. Ссылка на другие ресурсы *LinkLabel*

Элемент управления **LinkLabel** позволяет *создавать в форме ссылки* на Web-страницы, подобно гиперссылкам в HTML-документах, ссылки на открытие файлов какими-либо программами, ссылки на просмотр содержания логических дисков, папок и проч.

Напишем программу, которая с помощью элемента управления **LinkLabel** обеспечит ссылку для посещения почтового сервера **www.mail.ru**, ссылку для просмотра папки C:\Windows\ и ссылку для запуска текстового редактора Блокнот.

Для программирования этой задачи после запуска Visual Studio 2010 выберем шаблон **Windows Forms Application C#**, затем из панели **Toolbox** перетащим в форму три элемента управления **LinkLabel**. Равномерно разместим их в форме. Далее, следуя нашим традициям, не будем задавать никаких свойств этим элементам в окне **Properties**. Все начальные значения свойств укажем в программном коде сразу после вызова процедуры `InitializeComponent` (листинг 3.5).

Листинг 3.5. Ссылки на ресурсы

```
// Программа обеспечивает ссылку для посещения почтового сервера www.mail.ru,
// ссылку для просмотра папки C:\Windows\ и ссылку для запуска текстового
// редактора Блокнот с помощью элемента управления LinkLabel
```

```
using System;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace СсылкиLinkLabel
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
         this.Text = "Щелкните по ссылке:";
         linkLabel1.Text = "www.mail.ru";
         linkLabel2.Text = @"Папка C:\Windows\";
         linkLabel3.Text = "Вызвать \"Блокнот\"";
         this.Font = new System.Drawing.Font("Consolas", 12.0F);
         linkLabel1.LinkVisited = true;
         linkLabel2.LinkVisited = true;
         linkLabel3.LinkVisited = true;
         // Подписка на события: все три события обрабатываются
         // одной процедурой:
         linkLabel1.LinkClicked += new System.Windows.Forms.
                     LinkLabelLinkClickedEventHandler(this.ССЫЛКА);
         linkLabel2.LinkClicked += new System.Windows.Forms.
                     LinkLabelLinkClickedEventHandler(this.ССЫЛКА);
         linkLabel3.LinkClicked += new System.Windows.Forms.
                     LinkLabelLinkClickedEventHandler(this.ССЫЛКА);
      }
      private void ССЫЛКА(object sender, LinkLabelLinkClickedEventArgs e)
      {
         LinkLabel ссылка = (LinkLabel)sender;
         switch (ссылка.Name) // Выбор ссылки:
         {
            case "linkLabel1":
               System.Diagnostics.Process.Start(
                       "IExplore.exe", "http://www.mail.ru"); break;
            case "linkLabel2":
               System.Diagnostics.Process.Start("C:\\Windows\\"); break;
            case "linkLabel3":
               System.Diagnostics.Process.Start("Notepad", "text.txt");
               break;
         }
      }
   }
}
```

Как видно из программного кода, в свойстве `Text` каждой из ссылок **LinkLabel** задаем текст, из которого пользователь поймет назначение каждой ссылки. В задании свойства `Text` ссылки `LinkLabel3` для того, чтобы слово "Блокнот" было в двойных кавычках, используем escape-последовательность (`\"`) . Для большей выразительности задаем шрифт Consolas, 12 пунктов. Это шрифт моноширный, кстати, по умолчанию редактор в Visual Studio 2010 имеет также шрифт Consolas. Поскольку свойство `LinkVisited = true`, то соответствующая ссылка отображается как уже посещавшаяся (изменяется цвет).

Так же как и в предыдущих разделах, организуем обработку всех трех событий `Click` по каждой из ссылок одной процедурой обработки `ССЫЛКА`. В этой процедуре, так же как и в программе о трех кнопках и калькуляторе, в зависимости от имени объекта (ссылки), создающего события (`linkLabel1`, `linkLabel2`, `linkLabel3`), мы вызываем одну из трех программ: либо Internet Explorer, либо Windows Explorer, либо Блокнот. Информация об объекте, создающем событие `Click`, записана в объектную переменную `sender`. Она позволяет распознавать объекты (ссылки), создающие события. Чтобы "вытащить" эту информацию из `sender`, объявим переменную `ссылка` типа `LinkLabel` и с помощью неявного преобразования выполним конвертирование параметра `sender` в экземпляр класса `LinkLabel`. В этом случае переменная `ссылка` будет содержать все свойства объекта-источника события, в том числе свойство `name`, с помощью которого мы сможем распознавать выбранную ссылку. Идентифицируя по свойству `name` каждую из ссылок, с помощью метода `Start` вызываем либо Internet Explorer, либо Windows Explorer, либо Блокнот. Вторым параметром метода `Start` является имя ресурса, подлежащее открытию. Именем ресурса может быть или название Web-страницы, или имя текстового файла.

Фрагмент работы обсуждаемой программы приведен на рис. 3.6. Убедиться в ее работоспособности можно, открыв соответствующее решение в папке СсылкиLinkLabel.

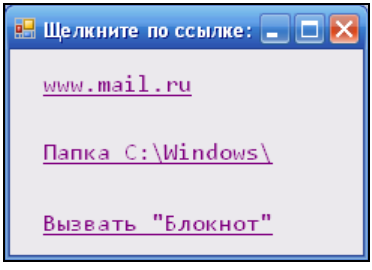


Рис. 3.6. Ссылки на ресурсы

Пример 20. Обработка событий клавиатуры

События клавиатуры (клавишные события) создаются в момент нажатия или отпуская ее клавиш. Различают событие `KeyPress`, которое *генерируется в момент нажатия клавиши*. При удержании клавиши в нажатом состоянии оно генерирует-

ся непрерывно с некоторой частотой. С помощью этого события можно распознать нажатую клавишу, если только она не является так называемой *модифицирующей*, т. е. <Alt>, <Shift> и <Ctrl>. А вот для того чтобы распознать, нажата ли модифицирующая клавиша <Alt>, <Shift> или <Ctrl>, следует обработать либо событие `KeyDown`, либо событие `KeyUp`. Событие `KeyDown` генерируется в *первоначальный* момент нажатия клавиши, а событие `KeyUp` — в момент *отпускания* клавиши.

Напишем программу, информирующую пользователя о тех клавишах и комбинациях клавиш, которые тот нажал. Запустим Visual Studio 2010, выберем проект шаблона **Windows Forms Application C#**, затем из панели **Toolbox** перетащим в форму две текстовых метки **Label**. Далее, поскольку нам потребуются клавишные события формы: `KeyPress`, `KeyDown`, `KeyUp`, получим пустые обработчики этих событий традиционным образом. То есть в панели **Properties** щелкнем на пиктограмме молнии (**Events**), а затем в списке всех возможных событий выберем каждое из названных событий клавиатуры. Программный код приведен в листинге 3.6.

Листинг 3.6. Обработка событий клавиатуры

```
// Программа, информирующая пользователя о тех клавишах и
// комбинациях клавиш, которые тот нажал
using System;
using System.Drawing;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace Key
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
         // Устанавливаем шрифт с фиксированной шириной (моноширный):
         base.Font = new Font(FontFamily.GenericMonospace, 14.0F);
         base.Text = "Какие клавиши нажаты сейчас:";
         label1.Text = string.Empty; label2.Text = string.Empty;
      }
      private void Form1_KeyPress(object sender, KeyPressEventArgs e)
      {  // Здесь событие нажатия клавиши: при удержании
         // клавиши генерируется непрерывно
         label1.Text = "Нажатая клавиша: " + e.KeyChar;
      }
      private void Form1_KeyDown(object sender, KeyEventArgs e)
```

```
    {  // Здесь обрабатываем мгновенное событие первоначального
       // нажатия клавиши
       label2.Text = string.Empty;
       // Если нажата клавиша <Alt>
       if (e.Alt == true) label2.Text += "Alt: Yes\n";
       else               label2.Text += "Alt: No\n";

       // Если нажата клавиша <Shift>
       if (e.Shift == true) label2.Text += "Shift: Yes\n";
       else                 label2.Text += "Shift: No\n";

       // Если нажата клавиша <Ctrl>
       if (e.Control == true) label2.Text += "Ctrl: Yes\n";
       else                   label2.Text += "Ctrl: No\n";

       label2.Text += "Код клавиши: " + e.KeyCode + "\nKeyData: " +
                      e.KeyData + "\nKeyValue: " + e.KeyValue;
    }
    private void Form1_KeyUp(object sender, KeyEventArgs e)
    {
       // Очистка меток при освобождении клавиши
       label1.Text = string.Empty; label2.Text = string.Empty;
    }
  }
}
```

В первую метку `label1` записываем сведения о нажатой обычной (т. е. не модифицирующей и не функциональной) клавише при обработке события `KeyPress`. Во вторую метку из аргумента события `e` (`e.Alt`, `e.Shift` и `e.Control`) получаем сведения, была ли нажата какая-либо модифицирующая клавиша (либо их комбинация). Обработчик события `KeyUp` очищает обе метки при освобождении клавиш.

На рис. 3.7 приведен фрагмент работы программы.

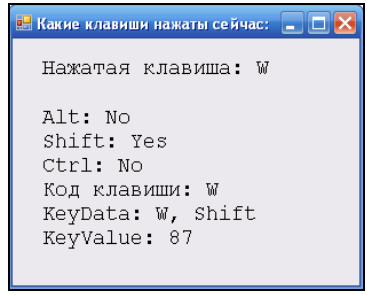


Рис. 3.7. Фрагмент работы программы, определяющей нажатую клавишу

Убедиться в работоспособности программы можно, открыв решение Key.sln в папке Key.

Пример 21. Разрешаем вводить в текстовое поле только цифры

Обычно для диагностики вводимых числовых данных мы пользовались функцией `TryParse`. Эта функция возвращает `true`, если на ее вход подаются числовые данные, и `false` в противном случае. Покажем, как можно совершенно по-другому решить задачу контроля вводимых пользователем данных. Можно вообще *не давать возможность пользователю вводить нечисловые данные*. Обратите внимание, как происходит ввод числовых данных в программе Калькулятор системы Windows, здесь программа просто не даст возможности пользователю ввести нечисловой символ. Продемонстрируем и мы такое решение на следующем примере.

Данная программа анализирует каждый символ, вводимый пользователем в текстовое поле формы. Если символ не является числовым, то текстовое поле получает запрет на ввод такого символа. Таким образом, программа *не дает* возможность пользователю ввода нечисловых данных.

Запустим систему Visual Studio 2010, в окне **New Project** выберем шаблон **Windows Forms Application C#**. На панели элементов **Toolbox** найдем текстовое поле **TextBox** и перетащим его в форму. Текст программы показан в листинге 3.7.

Листинг 3.7. Контроль вводимых пользователем числовых данных (вариант 1)

```
// Программа анализирует каждый символ, вводимый пользователем в текстовое
// поле формы. Если символ не является цифрой или Backspace, то текстовое поле
// получает запрет на ввод такого символа.
using System;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace Numbers
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
         this.Text = "Введите число"; textBox1.Clear();
      }
      private void textBox1_KeyPress(object sender, KeyPressEventArgs e)
```

```
      {  // Разрешаем ввод только десятичных цифр и Backspace:
         if (char.IsDigit(e.KeyChar) == true) return;
         if (e.KeyChar == (char)Keys.Back) return;
         e.Handled = true; // - запрет на ввод других вводимых символов
      }
   }
}
```

Как видно из программного кода, сразу после выполнения процедуры `InitializeComponent` задаем текст строки заголовка "Введите число" и очищаем текстовое поле `TextBox1.Clear()`. Самое интересное начинается при обработке события нажатия клавиши в текстовом поле `textBox1_KeyPress`. Пустой обработчик этого события мы получаем, как и предыдущих программах, т. е. на вкладке конструктора формы в панели свойств **Properties**, щелкнув на символе молнии (**Events**), выбираем в списке всех возможных событий событие `KeyPress` для текстового поля. Управляющая среда Visual Studio 2010 генерирует при этом пустую процедуру обработки данного события.

В этой процедуре можно легко определить, какую клавишу нажал пользователь, из аргумента события `e`. Символ, соответствующий нажатой клавише, содержится в свойстве аргумента `e.KeyChar`. На вход функции `IsDigital` подаем это свойство (т. е. исследуемый символ), а на выходе получаем заключение, является ли исследуемый символ цифрой (`true` или `false`). Аргумент события `e` имеет замечательное свойство `Handled`, которое либо запрещает получение данного события текстовым полем (`true`), либо разрешает (`false`). Всякий раз при очередной работе процедуры `textBox1_KeyPress` изначально свойство `e.Handled = false`, т. е. получение данного события текстовым полем разрешено. Последней строкой в процедуре запрещаем ввод символов в текстовое поле, но если пользователь вводит цифру или нажал клавишу <Backspace>, то этот запрет мы обходим с помощью оператора `return`. Таким образом, мы добиваемся игнорирования текстовым полем нецифровых символов.

Интерфейс рассматриваемого приложения показан на рис. 3.8.

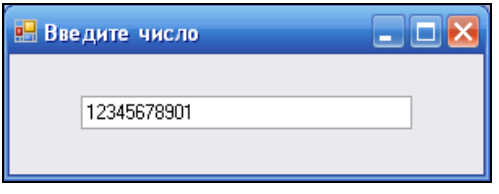


Рис. 3.8. Контроль введенных данных

Убедиться в работоспособности программы можно, открыв соответствующее решение в папке NumbersOnly.

Пример 22. Разрешаем вводить в текстовое поле цифры, а также разделитель целой и дробной части числа

Мы совсем забыли, уважаемые читатели, что число, вводимое пользователем, может иметь дробную часть после точки или запятой. Причем выяснить, что именно установлено в вашей системе — точка или запятая в качестве разделителя целой и дробной частей числа, можно, например, запустив Калькулятор Windows. Здесь среди экранных кнопок увидим кнопку либо с десятичной точкой, либо с десятичной запятой. Очень легко поменять данную установку системы (обычно по умолчанию в русифицированной версии Windows — запятая). Для этого следует в Панели управления выбрать значок **Язык и региональные стандарты**, затем на вкладке **Региональные параметры** щелкнуть на кнопке **Настройка** и на появившейся новой вкладке указать в качестве разделителя целой и дробной частей либо точку, либо запятую. Нам нужно добиться того, чтобы текстовое поле разрешало ввод только того разделителя, которое указано на вкладке **Региональные параметры**.

Для решения данной задачи запустим Visual Studio 2010, выберем проект шаблона **Windows Forms Application C#**. На панели элементов **Toolbox** найдем текстовое поле **TextBox** и перетащим его в форму. Текст программы представлен в листинге 3.8.

Листинг 3.8. Контроль вводимых пользователем числовых данных (вариант 2)

```
// Программа разрешает ввод в текстовое поле только цифровых символов,
// а также разделитель целой и дробной частей числа (т. е. точки или запятой)
using System;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace ТолькоЧисло_ТчкОгЗпт
{
   public partial class Form1 : Form
   {  // Разделитель целой и дробной частей числа может быть
      // точкой "." или запятой "," в зависимости от
      // установок в пункте Язык и региональные стандарты
      // Панели управления ОС Windows:
      System.Globalization.CultureInfo Культ = System.Globalization.
                                               CultureInfo.CurrentCulture;
      string ТчкИлиЗпт;
      public Form1()
```

```
      {
         InitializeComponent();
         this.Text = "Введите число";
         // Выясняем, что установлено на данном ПК в качестве
         // разделителя целой и дробной частей: точка или запятая
         ТчкИлиЗпт = Культ.NumberFormat.NumberDecimalSeparator;
      }
      private void textBox1_KeyPress(object sender, KeyPressEventArgs e)
      {
         bool ТчкИлиЗптНАЙДЕНА = false;
         // Разрешаю ввод десятичных цифр:
         if (char.IsDigit(e.KeyChar) == true) return;
         // Разрешаю ввод <Backspace>:
         if (e.KeyChar == (char)Keys.Back) return;
         // Поиск ТчкИлиЗпт в textBox, если IndexOf() == -1, то не найдена:
         if (textBox1.Text.IndexOf(ТчкИлиЗпт) != -1)
            ТчкИлиЗптНАЙДЕНА = true;
         // Если ТчкИлиЗпт уже есть в textBox, то запрещаем вводить и ее,
         // и любые другие символы:
         if (ТчкИлиЗптНАЙДЕНА == true) { e.Handled = true; return; }
         // Если ТчкИлиЗпт еще нет в textBox, то разрешаем ее ввод:
         if (e.KeyChar.ToString() == ТчкИлиЗпт) return;
         // В других случаях - запрет на ввод:
         e.Handled = true;
      }
   }
}
```

Как видно из текста программы, вначале выясняем, что установлено в данной системе в качестве разделителя целой дробной части: точка или запятая. Этот разделитель записываем в строковую переменную `ТчкИлиЗпт`, которая видна из всех процедур данной программы, поскольку объявлена вне всех процедур. Далее, как и в предыдущем примере, в процедуре обработки события `KeyPress` разрешаем ввод десятичных цифр и нажатие клавиши <Backspace> путем обхода с помощью `return` последнего оператора процедуры `e.Handled = true`, запрещающего ввод символа в текстовое поле.

В данной задаче мы имеем некоторую сложность с разрешением ввода разделителя целой и дробной частей, поскольку разрешить его ввод мы можем только один раз, но при этом надо помнить, что пользователь может его удалить и ввести в другом месте числовой строки. Эту проблему мы решили следующим образом, каждый раз при очередном нажатии клавиши, разрешив ввод десятичных цифр, в текстовом поле ищем искомый разделитель. Если он найден, то запрещаем ввод

любых нецифровых символов, включая злосчастный разделитель. *А если не найден*, то разрешаем его ввод.

На рис. 3.8 показан фрагмент работы программы.

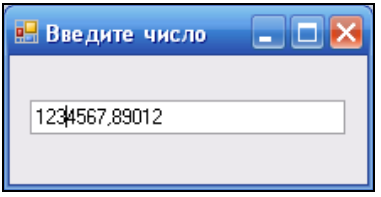


Рис. 3.9. Кроме цифр программа разрешает ввод десятичной запятой

Убедиться в работоспособности программы можно, открыв соответствующее решение в папке ТолькоЧисло+ТчкОrЗпт.

Глава 4

Чтение, запись текстовых и бинарных файлов, текстовый редактор

Пример 23. Чтение/запись текстового файла в кодировке Unicode. Обработка исключений *try...catch*

Очень распространенной задачей является сохранение данных на диске в текстовом формате (не в двоичном). В этом случае сохраненные данные можно читать, редактировать любым текстовым редактором, например Блокнотом или TextEdit. Читать текстовые данные также можно и в своей программе.

Казалось бы, это очень простая задача. Например, чтение текстового файла сводится буквально к нескольким строчкам:

```
// Создание экземпляра StreamReader для чтения из файла
System.IO.StreamReader Reader = new System.IO.StreamReader("C:\\Text1.txt");
// Считывание содержимого текстового файла в строку
string Stroka = Reader.ReadToEnd();
Reader.Close();
```

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 среду Visual Studio 2010 и закажем новый проект шаблона **Windows Forms Application C#**. Далее в форму перенесем текстовое поле и две командные копки. 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 от-

крытия файла, а другая — для сохранения файла на машинном носителе. В листинге 4.1 приведен текст данной программы.

Листинг 4.1. Чтение/запись текстового файла в кодировке Unicode

```
// Программа для чтения/записи текстового файла в кодировке Unicode
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace TXT_Unicode
{
   public partial class Form1 : Form
   {
      string filename = @"C:\Text1.txt";
      public Form1()
      {
         InitializeComponent();
         textBox1.Multiline = true; textBox1.Clear();
         textBox1.TabIndex = 0;
         textBox1.Size = new System.Drawing.Size(268, 112);
         button1.Text = "Открыть"; button1.TabIndex = 0;
         button2.Text = "Сохранить";
         base.Text = "Здесь кодировка Unicode";
      }
      private void button1_Click(object sender, System.EventArgs e)
      {  // Щелчок на кнопке "Открыть".
         // Русские буквы будут корректно читаться,
         // если файл в кодировке UNICODE:
         try
         {
            // Создание объекта StreamReader для чтения из файла:
            var Читатель = new System.IO.StreamReader(filename);
            // Непосредственное чтение всего файла в текстовое поле:
            textBox1.Text = Читатель.ReadToEnd();
            Читатель.Close(); // закрытие файла
            // Читать текстовый файл в кодировке UNICODE в массив строк
            // можно также таким образом (без Open и Close):
            // string[] МассивСтрок = System.IO.
            //                   File.ReadAllLines(@"C:\Text1.txt");
         }
         catch (System.IO.FileNotFoundException Ситуация)
         {  // Обработка исключительной ситуации:
```

```
            MessageBox.Show(Ситуация.Message + "\nНет такого файла",
                            "Ошибка", MessageBoxButtons.OK,
                             MessageBoxIcon.Exclamation);
         }
         catch (System.Exception Ситуация)
         {
            // Отчет о других ошибках:
            MessageBox.Show(Ситуация.Message, "Ошибка",
               MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
         }
      }
      private void button2_Click(object sender, System.EventArgs e)
      { // Щелчок на кнопке Сохранить:
         try
         {  // Создание объекта StreamWriter для записи в файл:
            var Писатель = new System.IO.StreamWriter(filename, false);
            Писатель.Write(textBox1.Text);
            Писатель.Close();
            // Сохранить текстовый файл можно также таким образом
            // (без Close), причем, если файл уже существует,
            // то он будет заменен:
            // System.IO.File.WriteAllText(@"C:\tmp.tmp", textBox1.Text);
         }
         catch (System.Exception Ситуация)
         {  // Отчет обо всех возможных ошибках:
            MessageBox.Show(Ситуация.Message, "Ошибка",
               MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
         }
      }
   }
}
```

Несколько слов о блоках `try`, которые, как видно, используются 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 C:\Text1.txt. Обычно в этой ситуации пользуются элементом управления **OpenFileDialog** для выбора файла. Мы не стали использовать этот элемент управ-

ления 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 назвали его русскими буквами. Затем следует чтение файла `filename` методом `ReadToEnd()` в текстовое поле `textBox1.Text` и закрытие файла методом `Close()`.

При обработке события "щелчок на кнопке" **Сохранить** организована запись файла на диск аналогично через объект `Писатель`. При создании объекта `Писатель` первым аргументом является `filename`, 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 рис. 4.1 приведен фрагмент работы программы.

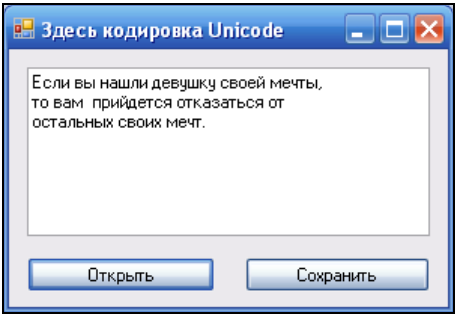


Рис. 4.1. Чтение/запись текстового файла в кодировке Unicode

Сделаем очень важное примечание. Запись текстового файла с помощью данной программы будет происходить *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 следить, чтобы кодировка была Unicode.

Однако по умолчанию в редакторах обычно используется кодировка ANSI. Кодировку ANSI с русскими буквами называют Windows 1251. Некоторые редакторы, например, RPad ("русский" Блокнот) вообще не работают с Unicode. Таким образом, на сегодняшний день пока что записывать в кодировке Unicode — это как бы эксклюзив. Возможно, даже наверняка, в ближайшее время многое изменится в пользу Unicode, поскольку информационные технологии меняются очень быстро.

Если в Блокноте подготовить текстовый файл в обычной кодировке ANSI, то прочитать русские буквы данной программой не получится, хотя английские буквы отобразятся в текстовом поле *без проблем*. Почему? Дело в том, что приведенная в тексте данной программы технология описана в учебных пособиях по Visual C#, в MSDN, на сайтах, подобных **http://msdn.microsoft.com/library/rus/**. Однако чаще это все англоязычные источники, а для английских текстов переход от одной кодировки к другой оказывается практически незаметным. Например, английские буквы кодировок Windows 1251, ASCII и Unicode совпадают, а с русскими буквами всегда

возникают недоразумения. Об этом говорят программисты на различных форумах в Интернете.

Программистам эти недоразумения следует учитывать. Разрешению *этого недоразумения* посвящена следующая программа.

Убедиться в работоспособности данной программы можно, открыв решение TXT_Unicode.sln в папке TXT_Unicode.

Пример 24. Чтение/запись текстового файла в кодировке Windows 1251

В данном примере также на экранной форме имеем текстовое поле и две командных кнопки, назначение этих элементов управления аналогично предыдущему примеру. Однако чтение и запись текстового файла в этом примере происходит в кодировке Windows 1251 (ANSI). Поскольку структура данной программы аналогична предыдущей, сразу обратимся к ее коду в листинге 4.2.

Листинг 4.2. Чтение/запись текстового файла в кодировке Windows 1251

```
// Программа для чтения/записи текстового файла в кодировке Windows 1251
using System;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace TXT_1251
{
   public partial class Form1 : Form
   {
      string filename = @"C:\Text2.txt";
      public Form1()
      {
         InitializeComponent();
         textBox1.Multiline = true; textBox1.Clear();
         textBox1.Size = new System.Drawing.Size(268, 112);
         button1.Text = "Открыть"; button1.TabIndex = 0;
         button2.Text = "Сохранить";
         this.Text = "Здесь кодировка Windows 1251";
      }
      private void button1_Click(object sender, EventArgs e)
      {  // Щелчок на кнопке Открыть
         try
         {  // Чтобы русские буквы читались бы корректно, объявляем
            // объект Кодировка:
```

```
            var Кодировка = System.Text.Encoding.GetEncoding(1251);
            // Создание экземпляра StreamReader для чтения из файла
            var Читатель = new System.IO.StreamReader(filename, Кодировка);
            textBox1.Text = Читатель.ReadToEnd();
            Читатель.Close();
            // Читать текстовый файл в кодировке Windows 1251 в массив строк
            // можно также таким образом (без Open и Close):
            // string[] МассивСтрок = System.IO.
            //         File.ReadAllLines(@"C:\Text2.txt", Кодировка);
         }
         catch (System.IO.FileNotFoundException Ситуация)
         {  // Обработка исключительной ситуации:
            MessageBox.Show(Ситуация.Message + "\nНет такого файла",
                            "Ошибка", MessageBoxButtons.OK,
                            MessageBoxIcon.Exclamation);
         }
         catch (Exception Ситуация)
         {  // Отчет о других ошибках
            MessageBox.Show(Ситуация.Message, "Ошибка",
               MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
         }
      }
      private void button2_Click(object sender, EventArgs e)
      {  // Щелчок на кнопке Сохранить:
         try
         {
            var Кодировка = System.Text.Encoding.GetEncoding(1251);
            // Создание экземпляра StreamWriter для записи в файл:
            var Писатель = new System.IO.StreamWriter(filename,
                                          false, Кодировка);
            Писатель.Write(textBox1.Text);
            Писатель.Close();
            // Сохранить текстовый файл можно также таким образом (без Close),
            //  причем, если файл уже существует, то он будет заменен:
            // System.IO.File.WriteAllText(@"C:\tmp.tmp",
            //                             textBox1.Text, Кодировка);
         }
         catch (System.Exception Ситуация)
         {  // Отчет обо всех возможных ошибках:
            MessageBox.Show(Ситуация.Message, "Ошибка",
               MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
         }
      }
   }
}
```

Этот текст программы отличается от предыдущего 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 `filename` 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 `filename`, 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.

Убедиться в работоспособности программы можно, открыв решение TXT_1251.sln в папке TXT_1251.

Пример 25. Простой текстовый редактор. Открытие и сохранение файла. Событие формы *Closing*

Итак, мы уже знаем, что существуют технологии чтения/записи текстового файла для нужной кодовой страницы. Таким образом, мы имеем основные компоненты для написания текстового редактора. Запустив систему Visual Studio 2010 и щелкнув по шаблону **Windows Forms Application C#**, перенесем из панели элементов управления **Toolbox** в форму текстовое поле **TextBox** и меню **MenuStrip**. Используя элемент управления **MenuStrip**, создадим один пункт меню **Файл** и три пункта подменю: **Открыть**, **Сохранить как** и **Выход** (рис. 4.2).

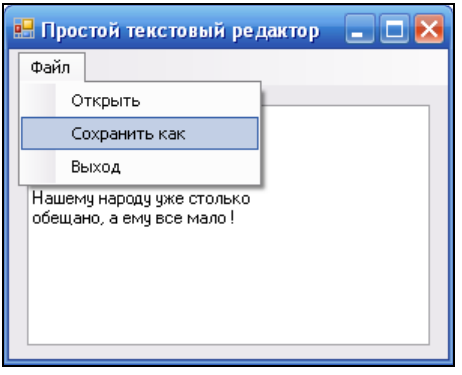


Рис. 4.2. Простой текстовый редактор

Из панели **Toolbox** нам понадобятся еще элементы управления **OpenFileDialog** и **SaveFileDialog**, также перенесем их в форму, хотя их очень легко объявить и ис-

пользовать непосредственно в программном коде. Итоговый текст программы "Простой текстовый редактор" представлен в листинге 4.3, и автор заранее приносит свои извинения за слишком длинный программный код, однако короче уже нельзя! Ксати, чтобы увеличить количество строчек программного кода, которое вы хотели бы одновременно видеть на экране, удобно пользоваться комбинацией клавиш <Shift>+Alt>+<Enter> (**Full Screen**).

Листинг 4.3. Простой текстовый редактор

```
// Простой текстовый редактор
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace TXT_edit
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
         textBox1.Multiline = true; textBox1.Clear();
         textBox1.Size = new System.Drawing.Size(268, 160);
         this.Text = "Простой текстовый редактор";
         openFileDialog1.FileName = @"C:\Text2.txt";
         openFileDialog1.Filter =
            "Текстовые файлы (*.txt)|*.txt|All files (*.*)|*.*";
         saveFileDialog1.Filter =
            "Текстовые файлы (*.txt)|*.txt|All files (*.*)|*.*";
      }
      private void открытьToolStripMenuItem_Click(object sender,
                                    System.EventArgs e)
      {  // Вывести диалог открытия фала
         openFileDialog1.ShowDialog();
         if (openFileDialog1.FileName == null) return;
         // Чтение текстового файла:
         try
         {  // Создание экземпляра StreamReader для чтения из файла
            var Читатель = new System.IO.StreamReader
                        (openFileDialog1.FileName,
                         System.Text.Encoding.GetEncoding(1251));
            // - здесь заказ кодовой страницы Win1251 для русских букв
            textBox1.Text = Читатель.ReadToEnd();
```

```
            Читатель.Close();
        }
        catch (System.IO.FileNotFoundException Ситуация)
        {
            MessageBox.Show(Ситуация + "\nНет такого файла", "Ошибка",
                MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
        }
        catch (System.Exception Ситуация)
        {
            // Отчет о других ошибках
            MessageBox.Show(Ситуация.Message, "Ошибка", MessageBoxButtons.OK,
                            MessageBoxIcon.Exclamation);
        }
    }
    private void сохранитьКакToolStripMenuItem_Click(object sender,
                                                     System.EventArgs e)
    {  // Пункт меню "Сохранить как"
        saveFileDialog1.FileName = openFileDialog1.FileName;
        if (saveFileDialog1.ShowDialog() == DialogResult.OK) Запись();
    }
    void Запись()
    {
        try
        {  // Создание экземпляра StreamWriter для записи в файл:
            var Писатель = new System.IO.StreamWriter
                          (saveFileDialog1.FileName, false,
                           System.Text.Encoding.GetEncoding(1251));
            // - здесь заказ кодовой страницы Win1251 для русских букв
            Писатель.Write(textBox1.Text);
            Писатель.Close(); textBox1.Modified = false;
        }
        catch (System.Exception Ситуация)
        {  // Отчет обо всех возможных ошибках
            MessageBox.Show(Ситуация.Message, "Ошибка", MessageBoxButtons.OK,
                            MessageBoxIcon.Exclamation);
        }
    }
    private void выходToolStripMenuItem_Click(object sender,
                                              System.EventArgs e)
    { this.Close(); }

    private void Form1_FormClosing(object sender, FormClosingEventArgs e)
    {
```

```
        if (textBox1.Modified == false) return;
        // Если текст модифицирован, то спросить, записывать ли файл?
        DialogResult MBox = MessageBox.Show(
                 "Текст был изменен. \nСохранить изменения?",
                 "Простой редактор", MessageBoxButtons.YesNoCancel,
                                     MessageBoxIcon.Exclamation);
        // YES — диалог;  NO — выход;  CANCEL - редактировать
        if (MBox == DialogResult.No) return;
        if (MBox == DialogResult.Cancel) e.Cancel = true;
        if (MBox == DialogResult.Yes)
        {
           if (saveFileDialog1.ShowDialog() == DialogResult.OK)
           {
              Запись(); return;
           }
           else e.Cancel = true; // Передумал выходить из ПГМ
        } // DialogResult.Yes
     }
  }
}
```

Как видно их текста программы, сразу после завершения работы процедуры `InitializeComponent` присваиваем начальные значения некоторым переменным. В частности для открытия (`Open`) и сохранения (`Save`) файлов заказываем фильтр (`Filter`) для текстовых файлов `*.txt`, а также для всех файлов `*.*`.

При обработке события "щелчок на пункте меню" **Открыть** выводим стандартный диалог открытия файлов `OpenFileDialog`, и если полученное в результате диалога имя файла не пусто (`null`), то организуем чтение текстового файла. Эта процедура в точности соответствует процедуре чтения файла из предыдущего раздела, разве что упрощен заказ на чтение в кодировке Windows 1251.

Аналогично написана обработка события "щелчок на пункте меню" **Сохранить как** (см. рис. 4.2). Выводится стандартный диалог `SaveFileDialog` сохранения файла, и если пользователь щелкнул на кнопке **OK** (или **Сохранить**), то вызывается процедура `Запись()`. Если нет, то пользователь отправляется редактировать текст. Процедура `Запись()` также в точности соответствует процедуре записи текстового файла из предыдущего раздела. Как видно, в процедуре `Запись()` попытка (`Try`) записи заканчивается оператором `textBox1.Modified = false`. Свойство `Modified` отслеживает изменения в тестовом поле. Понятно, что сразу после записи в файл следует это свойство перевести в состояние `false`.

На мой взгляд, наибольший интерес представляет организация выхода из программы. Выйти из программы можно либо через пункт меню **Выход**, либо закрывая программу традиционными методами Windows, т. е. нажатием комбинации клавиш <Alt>+<F4>, кнопки **Закрыть** или кнопки системного меню (слева вверху)

(обработка *события закрытия формы* `FormClosing`). Момент закрытия формы отслеживаем с помощью события формы `FormClosing`, которое происходит во время закрытия формы. Обратите внимание, выход по пункту меню **Выход** организован не через метод `Application.Exit()`, а через закрытие формы `this.Close()`. Почему? Потому что метод `Application.Exit()` не вызывает событие формы `FormClosing`.

Обычно выход из любого редактора (текстового, графического, табличного и т. д.) реализуется по следующему алгоритму. Если пользователь не сделал никаких изменений в редактируемом файле, то выйти из программы. Изменения в тестовом поле регистрируются свойством `textBox1.Modified`. Если в документе имеются несохраненные изменения (`textBox1.Modified = true`), а пользователь хочет выйти из программы, то выводят диалоговое окно (рис. 4.3).

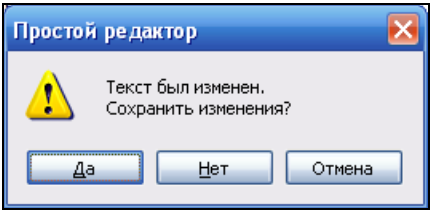


Рис. 4.3. Диалоговое окно при выходе из программы

В программе следует обработать каждый из возможных ответов пользователя по алгоритму, представленному на рис. 4.4.

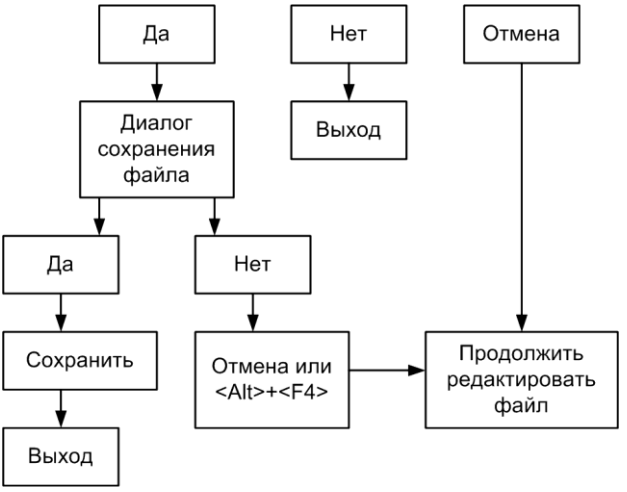


Рис. 4.4. Алгоритм обработки ответа пользователя программы

Обращаю внимание читателей *на ветвь алгоритма "Отмена"* (Cancel). Это случай, когда пользователь передумал выходить из программы и желает вернуться

к редактированию файла. Для реализации этого случая (см. листинг 4.3) обработка события `FormClosing` предусматривает булево свойство `e.Cancel`, которому можно присвоить значение `true`, означающее отказ от закрытия программы (пользователь передумал), т. е. в этом случае процедура `Form1_FormClosing` не закончится выходом из программы.

Аналогично, если пользователь согласился сохранить данные, то он попадает в стандартный диалог сохранения файла, и если при этом он передумал (диалог сохранения закрыт кнопкой **Отмена** или комбинацией клавиш <Alt>+<F4>), то следует предложить пользователю продолжить редактирование файла: `e.Cancel = true`. Как видно, в процедуре `Form1_FormClosing`, к сожалению, не удается избежать вложенных операторов условия.

Убедиться в работоспособности программы можно, открыв соответствующее решение в папке ТекстовыйРедактор.

Пример 26. Программа тестирования знаний студента по какому-либо предмету

В связи с внедрением в образование так называемого Болонского процесса — процесса сближения и гармонизации систем образования стран Европы с целью создания единого европейского пространства высшего образования — процедура проверки знаний студентов осуществляется в том числе посредством тестирования по различным предметам преподавания. Причем тестированию уделяется наибольшее внимание. В данном примере создадим инструмент для тестирования студентов, напишем программу, которая читает заранее подготовленный преподавателем текстовый файл с вопросами по какому-либо предмету, выводит в экранную форму каждый вопрос с вариантами ответов. Студент выбирает правильный вариант ответа, а в конце тестирования программа подводит итоги проверки знаний, выставляет общую оценку и, в качестве обоснования поставленной оценки, показывает вопросы, на которые студент ответил неправильно.

Фрагмент такого текстового файла для проверки знаний по информатике представлен в листинге 4.4.

Листинг 4.4. Содержимое текстового файла для тестирования студента по информатике

```
Информатика и программирование
1/6. Основные компоненты вычислительной системы:
процессор, звуковая карта, видеокарта, монитор, клавиатура;
монитор, клавиатура, материнская плата, процессор
процессор, ОП, внешняя память, монитор, клавиатура
3
```

```
2/6. Во время исполнения прикладная программа хранится:
в ПЗУ
в процессоре
в оперативной памяти
3
3/6. Иерархию усложнения данных можно представить в виде:
Бит - Байт - Поле - Запись - Файл - База Данных
Запись - Файл - Бит - Байт - База Данных - Поле
База Данных - Байт - Поле - Запись - Бит - Файл
1
4/6. Укажите строку, содержащую неверное утверждение
1 Кбайт = 1024 байт; 1 Гбайт = 1024 Мбайт
1 Мбайт это примерно миллион байт; 1 байт = 8 бит
1 Гбайт это примерно миллион байт; 1 Мбайт = 1024 Кбайт
3
5/6. Экспоненциальное представлення числа -1,84E-04 соответствует числу:
-0,000184
-0,00184
-18400
1
6/6. Текстовые данные кодируют с использованием:
таблиц размещения файлов FAT, NTFS и др.
таблиц символов Windows 1251, Unicode, ASCII и др.
структурированного языка запросов SQL
2
```

Структура этого файла такова. Первой строкой приведенного текстового файла является название предмета или темы, по которой проводится тестирование. Это название программа будет выводить в строке заголовка экранной формы. Для краткости изложения в данном тесте приводятся только шесть вопросов. На компакт-диске, прилагаемом к данной книге, представлен более полный файл test_полный.txt, с помошью которого автор тестирует знания своих студентов. Как видно из листинга 4.4, каждый вопрос имеет три варианта ответа. Строка с числом 1, 2 или 3 означает номер правильного ответа. Программа будет считывать это число и сравнивать с номером варианта, который выбрал тестируемый. Дробь, например, 4/6 приведена для того, чтобы тестируемый понимал, в какой точке траектории в данный момент находится, т. е. он отвечает на четвертый вопрос, а всего их шесть.

Таким образом, задача понятна, приступаем к ее программированию. Запустим Visual Studio 2010, закажем новый проект шаблона **Windows Forms Application C#**. Затем из панели элементов перенесем в форму две командных кнопки, текстовую метку и три переключателя **RadioButton**. Далее через щелчок правой кнопкой мыши перейдем к вкладке программного кода (листинг 4.5).

Листинг 4.5. Программа тестирования знаний студента

```
// Программа тестирует студента по какому-либо предмету обучения
using System;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace Тестирование
{
   public partial class Form1 : Form
   {
      int СчетВопросов = 0;     // Счет вопросов
      int ПравилОтветов = 0;    // Количество правильных ответов
      int НеПравилОтветов = 0; // Количество неправильных ответов
      string[] НеПравилОтветы; // Массив вопросов,
                               // на которые даны неправильные ответы
      int НомерПравОтвета;      // Номер правильного ответа
      int ВыбранОтвет;          // Номер ответа, выбранный студентом
      // Чтобы русские буквы читались корректно, объявляем объект Кодировка:
      System.Text.Encoding Кодировка =
          System.Text.Encoding.GetEncoding(1251);
      System.IO.StreamReader Читатель;
      public Form1()
      {
         InitializeComponent();
         button1.Text = "Следующий вопрос";
         button2.Text = "Выход";
         // Подписка на событие изменение состояния
         // переключателей RadioButton:
         radioButton1.CheckedChanged +=
            new System.EventHandler(ИзмСостПерекл);
         radioButton2.CheckedChanged +=
            new System.EventHandler(ИзмСостПерекл);
         radioButton3.CheckedChanged +=
            new System.EventHandler(ИзмСостПерекл);
         НачалоТеста();
      }
      void НачалоТеста()
      {
         try
         {  // Создание экземпляра StreamReader для чтения из файла
            Читатель = new System.IO.
```

```
            StreamReader(System.IO.Directory.GetCurrentDirectory()
                        + @"\test.txt", Кодировка);
         this.Text = Читатель.ReadLine(); // Название предмета
         // Обнуление всех счетчиков:
         СчетВопросов = 0; ПравилОтветов = 0; НеПравилОтветов = 0;
         НеПравилОтветы = new string[100];
      }
      catch (Exception Ситуация)
      {  // Отчет о всех ошибках
         MessageBox.Show(Ситуация.Message, "Ошибка",
            MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
      }
      ЧитатьСледВопрос();
   }
   void ЧитатьСледВопрос()
   {
      label1.Text = Читатель.ReadLine();
      // Считывание вариантов ответа:
      radioButton1.Text = Читатель.ReadLine();
      radioButton2.Text = Читатель.ReadLine();
      radioButton3.Text = Читатель.ReadLine();
      // Выясняем, какой ответ - правильный:
      НомерПравОтвета = int.Parse(Читатель.ReadLine());
      // Переводим все переключатели в состояние "выключено":
      radioButton1.Checked = false; radioButton2.Checked = false;
      radioButton3.Checked = false;
      // Первая кнопка не активна, пока студент не выберет вариант ответа
      button1.Enabled = false;
      СчетВопросов = СчетВопросов + 1; // Проверка, конец ли файла:
      if (Читатель.EndOfStream == true) button1.Text = "Завершить";
   }
   private void ИзмСостПерекл(object sender, EventArgs e)
   {  // Кнопка "Следующий вопрос" становится активной, и ей передаем фокус
      button1.Enabled = true; button1.Focus();
      RadioButton Переключатель = (RadioButton)sender;
      string tmp = Переключатель.Name;
      // Выясняем номер ответа, выбранный студентом:
      ВыбранОтвет = int.Parse(tmp.Substring(11));
   }
   private void button1_Click(object sender, EventArgs e)
   {  // Щелчок на кнопке
      //  "Следующий вопрос/Завершить/Начать тестирование снач".
```

```
// Счет правильных ответов:
if (ВыбранОтвет == НомерПравОтвета)
   ПравилОтветов = ПравилОтветов + 1;
if (ВыбранОтвет != НомерПравОтвета)
{  // Счет неправильных ответов:
   НеПравилОтветов = НеПравилОтветов + 1;
   // Запоминаем вопросы с неправильными ответами:
   НеПравилОтветы[НеПравилОтветов] = label1.Text;
}
if (button1.Text == "Начать тестирование сначала")
{
   button1.Text = "Следующий вопрос";
   // Переключатели становятся видимыми, доступными для выбора:
   radioButton1.Visible = true; radioButton2.Visible = true;
   radioButton3.Visible = true;
   // Переход к началу файла
   НачалоТеста(); return;
}
if (button1.Text == "Завершить")
{
   Читатель.Close(); // Закрываем текстовый файл
   // Переключатели делаем невидимыми:
   radioButton1.Visible = false; radioButton2.Visible = false;
   radioButton3.Visible = false;
   // Формируем оценку за тест:
   label1.Text = string.Format("Тестирование завершено.\n" +
       "Правильных ответов: {0} из {1}.\n" +
       "Оценка в пятибалльной системе: {2:F2}.", ПравилОтветов,
       СчетВопросов, (ПравилОтветов * 5F) / СчетВопросов);
   // 5F - это максимальная оценка
   button1.Text = "Начать тестирование сначала";
   // Вывод вопросов, на которые Вы дали неправильный ответ
   string Str = "СПИСОК ВОПРОСОВ, НА КОТОРЫЕ ВЫ ДАЛИ " +
                "НЕПРАВИЛЬНЫЙ ОТВЕТ:\n\n";
   for (int i = 1; i <= НеПравилОтветов; i++)
      Str = Str + НеПравилОтветы[i] + "\n";
   // Если есть неправильные ответы, то вывести через MessageBox
   // список соответствующих вопросов:
   if (НеПравилОтветов != 0)
        MessageBox.Show(Str, "Тестирование завершено");
}
```

```
            if (button1.Text == "Следующий вопрос") ЧитатьСледВопрос();
        }
        private void button2_Click(object sender, EventArgs e)
        {  // Щелчок на кнопке "Выход"
            this.Close();
        }
    }
}
```

В начале программы объявляем переменные, которые должны быть "видны" из всех процедур класса `Form1`. Сразу после выполнения процедуры `InitializeComponent` организуем подписку на событие "изменение состояния переключателей" **RadioButton** одной процедурой `ИзмСостПерекл`. В данной программе изменение состояния любого из трех переключателей будем обрабатывать одной процедурой `ИзмСостПерекл`.

Далее в процедуре `НачалоТеста` открываем файл test.txt, в котором содержится непосредственно тест, и читаем первую строку с названием предмета или темы, подлежащей тестированию. При этом обнуляем счетчик всех вопросов и счетчики вопросов, на которые студент дал правильные и неправильные ответы. Затем вызываем процедуру `ЧитатьСледВопрос`, которая читает очередной вопрос, варианты ответов на него и номер варианта правильного ответа. Тут же проверяем, не достигнут ли конец читаемого программой файла. Если достигнут, то меняем надпись на первой кнопке на "Завершить". В данной программе надпись на первой кнопке является как бы флагом, который указывает, по какой ветви в программе следует передавать управление.

При выборе студентом того или иного варианта испытуемый может сколь угодно раз щелкать на разных переключателях, пока не выберет окончательно вариант ответа. Программа будет фиксировать выбранный вариант только на этапе щелчка на кнопке **Следующий вопрос**. В процедуре обработки события "изменение состояния переключателей" выясняем, какой из вариантов ответа выбрал студент, но делаем вывод, правильно ли ответил студент или нет, только при обработке события "щелчок" на первой кнопке.

В процедуре обработки события щелчок на первой кнопке ведем счет правильных и неправильных ответов, а также запоминаем в строковый массив вопросы, на которые студент дал неверный ответ. Если достигнут конец файла и надпись на кнопке стала "Завершить", то закрываем текстовый файл, все переключатели делаем невидимыми (уже выбирать нечего) и формируем оценку за прохождение теста, а также через `MessageBox` выводим список вопросов, на которые испытуемый дал ошибочный ответ.

Фрагмент работы тестирующей программы представлен на рис. 4.5.

На рис. 4.6 показан финальный фрагмент работы тестирующей программы, где выведено обоснование оценки тестирования со списком вопросов, на которые студент ответил неправильно.

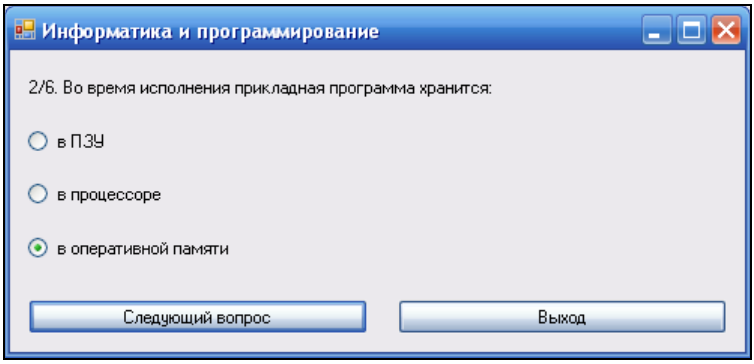


Рис. 4.5. Интерфейс тестирующей программы

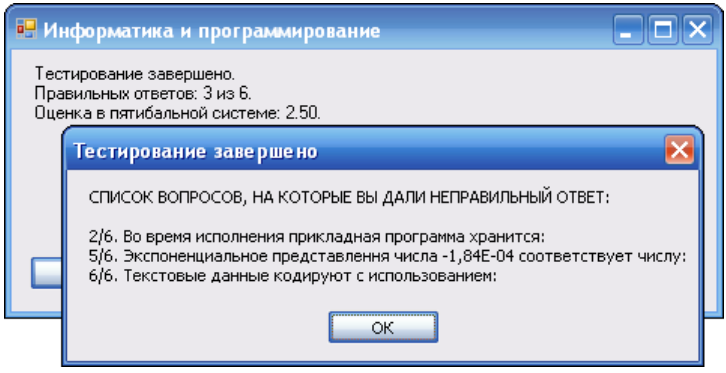


Рис. 4.6. Финальный фрагмент работы программы

Вы можете совершенствовать данную программу. Например, можно добавить элемент управления **Timer**, чтобы ограничить время сдачи теста. Убедиться в работоспособности программы можно, открыв решение Тестирование.sln в папке Тестирование.

Пример 27. Простой RTF-редактор

Читателю, вероятно, известно, что Visual C# 2010 имеет элемент управления **RichTextBox** (форматированное текстовое поле), так же как и предыдущие версии Visual C#. Этот элемент управления позволяет осуществлять форматирование текста в *стандарте RTF* (один из форматов MS Word). В формате RTF в текст вводятся специальные коды форматирования, несущие информацию о гарнитуре, размерах шрифтов, стилях символов и абзацев, выравнивании и других возможностях форматирования.

Напишем очень простой RTF-редактор, который читает как RTF-файлы, так и обычные текстовые файлы в кодировке Windows 1251, но сохраняет файлы на диск

в любом случае в формате RTF. Для этой цели перенесем из панели элементов управления **Toolbox** элементы управления **RichTextBox**, меню **MenuStrip**, **SaveFileDialog** и **OpenFileDialog**. В выпадающем меню **Файл** предусмотрим такие пункты меню, как показано на рис. 4.7: **Открыть в формате RTF**, **Открыть в формате Win1251**, **Сохранить в формате RTF** и **Выход**. Текст программы приведен в листинге 4.6.

Листинг 4.6. Простой RTF-редактор

```
// Программа простейшего RTF-редактора
using System;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace RTF_edit
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
         base.Text = "Простой RTF-редактор"; richTextBox1.Clear();
         openFileDialog1.FileName = @"c:\Text2.txt";
         saveFileDialog1.Filter = "Файлы RTF (*.RTF)|*.RTF";
         // Подписка на обработку двух событий одной процедурой:
         this.открытьВФорматеRTFToolStripMenuItem.Click += new System.
                                 EventHandler(this.ОТКРЫТЬ);
         this.открытьВФорматеWin1251ToolStripMenuItem.Click += new System.
                                 EventHandler(this.ОТКРЫТЬ);
      }
      private void ОТКРЫТЬ(object sender, EventArgs e)
      {  // Процедура обработки событий открытия файла в двух разных форматах.
         // Выясняем, в каком формате открыть файл:
         ToolStripMenuItem t = (ToolStripMenuItem)sender;
         string format = t.Text; // - читаем надпись на пункте меню
         try
         {  // Открыть в каком-либо формате:
            if (format == "Открыть в формате RTF")
            {
               openFileDialog1.Filter = "Файлы RTF (*.RTF)|*.RTF";
               openFileDialog1.ShowDialog();
               if (openFileDialog1.FileName == null) return;
```

```
            richTextBox1.LoadFile(openFileDialog1.FileName);
         }
         if (format == "Открыть в формате Win1251")
         {
            openFileDialog1.Filter = "Текстовые файлы (*.txt)|*.txt";
            openFileDialog1.ShowDialog();
            if (openFileDialog1.FileName == null) return;
            richTextBox1.LoadFile(openFileDialog1.FileName,
                                  RichTextBoxStreamType.PlainText);
         }
         richTextBox1.Modified = false;
      }
      catch (System.IO.FileNotFoundException Exc)
      {
         MessageBox.Show(Exc.Message + "\nНет такого файла", "Ошибка",
            MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
      }
      catch (Exception Exc)
      {  // Отчет о других ошибках
         MessageBox.Show(Exc.Message, "Ошибка",
            MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
      }
   }
   private void сохранитьВФорматеRTFToolStripMenuItem_Click(
                               object sender, EventArgs e)
   {
      saveFileDialog1.FileName = openFileDialog1.FileName;
      if (saveFileDialog1.ShowDialog() == DialogResult.OK) Запись();
   }
   void Запись()
   {
      try
      {
         richTextBox1.SaveFile(saveFileDialog1.FileName);
         richTextBox1.Modified = false;
      }
      catch (Exception Exc)
      {  // Отчет обо всех возможных ошибках:
         MessageBox.Show(Exc.Message, "Ошибка",
            MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
      }
   }
```

```
      private void выходToolStripMenuItem_Click(object sender, EventArgs e)
      {
         this.Close();
      }
      private void Form1_FormClosing(object sender, FormClosingEventArgs e)
      {
         if (richTextBox1.Modified == false) return;
         // Если текст модифицирован, то выясняем, записывать ли файл?
         DialogResult MBox = MessageBox.Show(
                     "Текст был изменен. \nСохранить изменения?",
                     "Простой редактор", MessageBoxButtons.YesNoCancel,
                                         MessageBoxIcon.Exclamation);
         // YES — диалог;  NO — выход;  CANCEL - редактирование
         if (MBox == DialogResult.No) return;
         if (MBox == DialogResult.Cancel) e.Cancel = true;
         if (MBox == DialogResult.Yes)
         {
            if (saveFileDialog1.ShowDialog() == DialogResult.OK)
               { Запись(); return; }
            else e.Cancel = true; // Передумал выходить из ПГМ
         } // - DialogResult.Yes
      }
   }
}
```

Структура программы аналогична программе простого текстового редактора, описанного в предыдущем разделе. Сразу после выполнения процедуры `InitializeComponent` задаем начальные значения некоторым переменным (см. текст программы). Здесь же выполняем подписку на обработку одной процедурой `ОТКРЫТЬ` двух событий — выбор пунктов меню **Открыть в формате RTF** и **Открыть в формате Win1251**. Подробно особенности обработки событий, создаваемых разными объектами, обсуждены нами в примерах 17, 18 и др. *(см. главу 3)*. Из этих примеров читатели знают, что сведения об объекте, создавшем событие, находятся в объектной переменной `sender`. Какой пункт меню указал пользователь, можно узнать, конвертировав переменную `sender` в объект `t` класса `ToolStripMenuItem`. В таком случае мы можем прочитать в свойстве `Text` название пункта меню, которое выбрал пользователь. Таким образом, в строковую переменную `format` попадает или строка "Открыть в формате RTF", или строка "Открыть в формате Win1251". Метод `LoadFile` объекта `richTextBox1` загружает либо файл в формате RTF, либо файл в обычном текстовом формате. Перехватчик ошибок `catch` сообщает пользователю либо о том, что такого файла нет, либо если пользователь использует пункт меню **Открыть в формате RTF** для открытия текстового файла, он получает сообщение "Недопустимый формат файла".

Сохранение файла (рис. 4.7) выполняется также с использованием стандартного диалога **SaveFileDialog**. Непосредственное сохранение файла удобнее всего выполнять в отдельной процедуре `Запись()`, поскольку эту процедуру необходимо вызывать также при выходе из программы, когда в документе имеются несохраненные изменения `richTextBox1.Modified = True`.

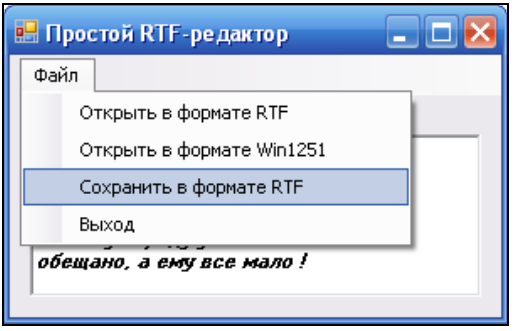


Рис. 4.7. Простой RTF-редактор

В основе процедуры `Запись()` также лежит блок `try...catch`: выполнить попытку (`try`) сохранения файла (`SaveFile`) и при этом перехватить (`catch`) возможные недоразумения и сообщить о них пользователю в диалоговом окне `MessageBox`.

Выход из программы организован абсолютно так же, как и в программе из предыдущего примера. Вдобавок обработаны два события — пункт меню **Выход** и всевозможные закрытия программы традиционными способами Windows. Предусмотрен диалог с пользователем в случае имеющих место несохраненных данных.

Замечу, для закрытия приложения *следует осторожно* пользоваться методом `Exit` объекта `Application` (можно сказать, с оглядкой). Этот метод подготавливает приложение к закрытию. Да, метод `Application.Exit()` *не вызывает* события формы `Closing`. Но попробуйте проследить за поведением программы после команды `Application.Exit` с *помощью отладчика* (клавиша <F11>). Вы убедитесь, что после команды `Application.Exit` управление перейдет следующему оператору, затем — следующему, и так до конца процедуры. Если на пути встретится функция `MessageBox`, то программа выдаст это диалоговое окно, и это несмотря на то, что уже давно была команда `Application.Exit`. Аналогично ведет себя метод `Close` элемента `Form` (если вы работаете с проектом **Windows Application**), который вызывается таким образом: `this.Close()`. Да, `this.Close` вызывает событие формы `Closing`. Этот метод закрывает форму и *освобождает все ресурсы*. Но для освобождения ресурсов после команды `this.Close` управление также пройдет все операторы процедуры. Таким образом, для немедленного выхода из процедуры следует комбинировать названные методы с `return`.

Убедиться в работоспособности программы можно, открыв решение RTF_edit.sln в папке RTF_edit.

Пример 28. 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, тем труднее текст программы для понимания. А наша задача — выразительно и ярко демонстрировать технологии в максимально простой форме.

Программа, представленная в данном разделе, имеет такие скромные возможности: открыть в стандартном диалоге 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.

Таким образом, чтобы создать данную программу, следует в форму перенести следующие элементы управления: текстовое поле **TextBox**, меню **MenuStrip** с пунктами меню: **Открыть**, **Печатать** и **Выход**, а также элементы управления **OpenFileDialog** и **PrintDocument**. Текст программы представлен в листинге 4.7.

Листинг 4.7. Печать текстового документа

```
// Программа позволяет открыть в стандартном диалоге текстовый файл,
// просмотреть его в текстовом поле без возможности изменения текста
// (ReadOnly) и при желании пользователя вывести этот текст на принтер.
using System;
using System.Drawing;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace TXT_print
{
   public partial class Form1 : Form
   {
      System.IO.StreamReader Читатель;
      public Form1()
      {
         InitializeComponent();
         base.Text = "Открытие текстового файла и его печать";
         textBox1.Multiline = true; textBox1.Clear();
         textBox1.Size = new System.Drawing.Size(268, 112);
         textBox1.ScrollBars = ScrollBars.Vertical;
         textBox1.ReadOnly = true;
         // До тех пор, пока файл не прочитан в текстовое поле,
         // не должен быть виден пункт меню "Печать..."
```

```
    печатьToolStripMenuItem.Visible = false;
    openFileDialog1.FileName = null;
}
private void открытьToolStripMenuItem_Click(object sender, EventArgs e)
{  // Щелчок на пункте меню "Открыть":
    openFileDialog1.Filter =
            "Текстовые файлы (*.txt)|*.txt|All files (*.*)|*.*";
    openFileDialog1.ShowDialog();
    if (openFileDialog1.FileName == null) return;
    try
    {  // Создание потока StreamReader для чтения из файла
       Читатель = new System.IO.StreamReader(openFileDialog1.FileName,
                   System.Text.Encoding.GetEncoding(1251));
       // - здесь заказ кодовой страницы Win1251 для русских букв
       textBox1.Text = Читатель.ReadToEnd();
       Читатель.Close();
       печатьToolStripMenuItem.Visible = true;
    }
    catch (System.IO.FileNotFoundException Exc)
    {
       MessageBox.Show(Exc.Message + "\nНет такого файла", "Ошибка",
       MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
    }
    catch (Exception Exc)
    {  // Отчет о других ошибках:
       MessageBox.Show(Exc.Message, "Ошибка",
       MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
    }
}
private void печатьToolStripMenuItem_Click(object sender, EventArgs e)
{  // Пункт меню "Печать"
    try
    {
       Читатель = new System.IO.StreamReader(openFileDialog1.FileName,
                    System.Text.Encoding.GetEncoding(1251));
       // - здесь заказ кодовой страницы Win1251 для русских букв
       try { printDocument1.Print(); }
       finally { Читатель.Close(); }
    }
    catch (Exception ex) { MessageBox.Show(ex.Message); }
}
private void printDocument1_PrintPage(object sender, System.Drawing.
```

```
                                Printing.PrintPageEventArgs e)
    {
       Single linesPerPage = 0;
       Single yPos = 0; int count = 0;
       Single leftMargin = e.MarginBounds.Left;
       Single topMargin = e.MarginBounds.Top;
       String line = null;
       Font printFont = new Font("Times New Roman", 12.0F);
       // Вычисляем количество строк на одной странице
       linesPerPage = e.MarginBounds.Height /
                      printFont.GetHeight(e.Graphics);
       // Печатаем каждую строку файла
       while (count < linesPerPage)
       {
          line = Читатель.ReadLine();
          if (line == null) break; // выход из цикла
          yPos = topMargin + count * printFont.GetHeight(e.Graphics);
          // Печать строки
          e.Graphics.DrawString(line, printFont, Brushes.Black,
                           leftMargin, yPos, new StringFormat());
          count += 1;
       }
       // Печать следующей страницы, если есть еще строки файла
       if (line != null) e.HasMorePages = true;
       else e.HasMorePages = false;
    }
    private void выходToolStripMenuItem_Click(object sender, EventArgs e)
    {  // Выход из программы
       base.Close();
    }
  }
}
```

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 работы программы 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 тек-

стовых 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**. Этот пункт написан автором по технологии, приведенной в справочной системе по C# (MSDN). 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, `leftMargin` — значение отступа от левого края, и т. д. Назначаем шрифт печати — Times New Roman, 12 пунктов.

Далее в цикле `while` программа читает каждую строку `line` из файла — `Читатель.ReadLine()`, а затем распечатывает ее командой (методом) `DrawString`. Здесь используется графический объект `Graphics`, который получаем из аргумента события `e`.

В переменной `count` происходит счет строк. Если количество строк оказывается бóльшим, чем число строк на странице `linesPerPage`, то происходит выход из цикла, поскольку страница распечатана. Если есть еще страницы, а программа выясняет это, анализируя содержимое переменной `line`, если ее содержимое отличается от значения `null` (`line != null`), то аргументной переменной `e.HasMorePages` назначаем `true`, что инициирует опять событие `PrintPage` и подпрограмма `PrintDocument1.Print` начинает свою работу вновь. И так, пока не закончатся все страницы `e.HasMorePages = False` для печати на принтере.

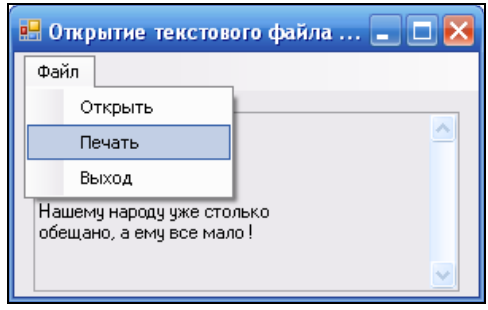


Рис. 4.8. Фрагмент работы программы печати текстового файла

На рис. 4.8 показан интерфейс приложения.

Убедиться в работоспособности программы можно, открыв решение TXT_print.sln в папке TXT_print.

Пример 29. 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 имеем 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 командных кнопки: **Читать** и **Сохранить**. Таким образом, получим пользовательский интерфейс, показанный на рис. 4.9. Текст программы приведен в листинге 4.8.

Листинг 4.8. Чтение/запись бинарных файлов

```
// Программа для чтения/записи бинарных файлов с использованием потока данных
using System;
using System.Windows.Forms;
using System.IO; // - добавляем пространство имен для сокращения программного
кода
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace Read_Write_bin
{
   public partial class Form1 : Form
```

```
    {
        public Form1()
        {
            InitializeComponent();
            base.Text = "Успеваемость студента"; label1.Text = "Номер п/п";
            label2.Text = "Фамилия И.О."; label3.Text = "Средний балл";
            textBox1.Clear(); textBox2.Clear(); textBox3.Clear();
            button1.Text = "Читать"; button2.Text = "Сохранить";
        }
        private void button1_Click(object sender, EventArgs e)
        {  // ЧТЕНИЕ БИНАРНОГО ФАЙЛА.
            // Если такого файла нет
            if (File.Exists(@"C:\student.usp") == false) return;
            // Создание потока Читатель
            var Читатель = new BinaryReader(File.OpenRead(@"C:\student.usp"));
            try
            {
                int Номер_пп = Читатель.ReadInt32();
                string ФИО = Читатель.ReadString();
                Single СредБалл = Читатель.ReadSingle();
                textBox1.Text = Convert.ToString(Номер_пп);
                textBox2.Text = Convert.ToString(ФИО);
                textBox3.Text = Convert.ToString(СредБалл);
            }
            finally { Читатель.Close(); }
        }
        private void button2_Click(object sender, EventArgs e)
        {  // ЗАПИСЬ БИНАРНОГО ФАЙЛА.
            // Создаем поток Писатель для записи байтов в файл
            BinaryWriter Писатель = new BinaryWriter(
                  File.Open(@"C:\student.usp", FileMode.Create));
            try
            {
                int Номер_пп = Convert.ToInt32(textBox1.Text);
                string ФИО = Convert.ToString(textBox2.Text);
                Single СредБалл = Convert.ToSingle(textBox3.Text);
                Писатель.Write(Номер_пп);
                Писатель.Write(ФИО);
                Писатель.Write(СредБалл);
            }
            finally { Писатель.Close(); }
        }
    }
}
```

Как видно, сразу после выполнения процедуры `InitializeComponent` организована инициализация (присвоение начальных значений) элементам формы: текстовых полей, меток и кнопок.

Запись файла на диск происходит при обработке события `button2.Click`, т. е. щелчок мышью на кнопке **Сохранить** (рис. 4.9). Для этого создаем поток байтов `Писатель` для открытия файла student.usp. Если такой файл не существует, то он создается (`Create`), а если файл уже есть, то он перезаписывается. Как видно, для упрощения программы мы не использовали элемент управления `OpenFileDialog` для открытия файла в диалоге.

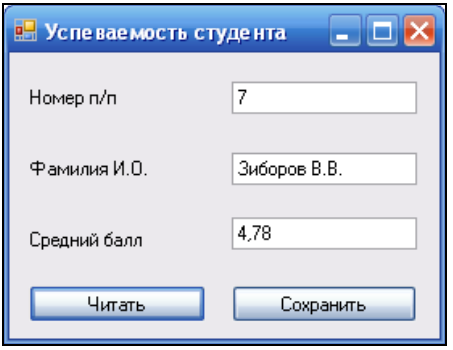


Рис. 4.9. Фрагмент работы программы чтения/записи бинарных файлов

Далее преобразуем записанное в текстовых полях в более естественные типы данных. Номер по порядку `Номер_пп` — это тип `int`, 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 стандартный диалог, поэтому вначале процедуры выясняем, существует ли такой файл. Если файла C:\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 этот фрагмент программного кода читается (воспринимается) программистом легко.

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 типа `int`, аналогично функ-

циям `ReadString` и `ReadSingle`. Далее осуществлено конвертирование этих переменных в строковый тип `Convert.ToString`. Как видно, 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 данной программы может быть по пути добавления в файл сведений о других студентах. В таком случае при чтении файла будет неопределенность количества студентов. Тогда следует обработать ситуацию достижения конца файла:

```
catch (EndOfStreamException e)
```

а затем закрыть файл.

Убедиться в работоспособности программы можно, открыв решение Read_Write_bin.sln в папке Read_Write_bin.

Глава 5

Редактирование графических данных

Пример 30. Простейший вывод отображения графического файла в форму

Поставим задачу вывода в форму какого-нибудь изображения растрового графического файла формата BMP, JPEG, PNG или других форматов. Для решения этой задачи запустим Visual Studio 2010, выберем шаблон **Windows Forms Application C#**. Двойной щелчок на проекте формы приведет нас на вкладку программного кода. Работать с графикой в форме можно по-разному. Покажем работу с графикой через переопределение метода `OnPaint`.

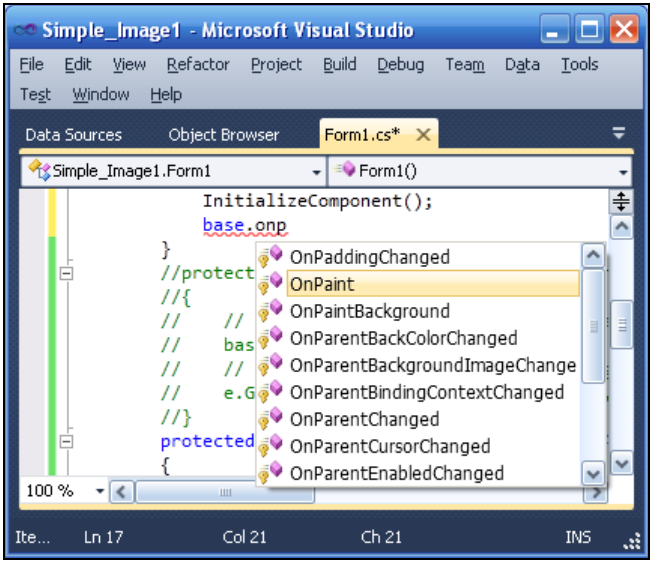


Рис. 5.1. Часть списка методов и свойств объекта `Form1`

Метод `OnPaint` является членом класса `Form`. Этот метод можно увидеть, набрав `"base."` внутри какой-нибудь процедуры, в выпадающем списке *методов и свойств* объекта `Form1` (рис. 5.1).

Метод `OnPaint` можно *переопределить*, т. е. добавить к уже существующим функциям собственные. Для этого в окне программного кода напишем:

```
protected override void
```

и в появившемся раскрывающемся списке выберем `OnPaint`. Система сама сгенерирует необходимые строчки процедуры, *подлежащей переопределению*:

```
protected override void OnPaint(PaintEventArgs e)
{
   base.OnPaint(e);
}
```

Теперь этот программный код следует дополнить командами для вывода в форму изображения из растрового файла poryv.png (листинг 5.1).

Листинг 5.1. Вывод растрового изображения в форму (вариант 1)

```
// Программа выводит в форму растровое изображение из графического файла
using System.Drawing;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace Simple_Image1
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
      }
      protected override void OnPaint(PaintEventArgs e)
      {
         base.OnPaint(e);
         base.Text = "Рисунок";
         // Размеры формы
         this.Width = 200; this.Height = 200;
         // Создаем объект для работы с изображением
         Image Рисунок = new Bitmap("C:\\poryv.png");
         // Вывод изображения в форму
         e.Graphics.DrawImage(Рисунок, 5, 5);
         // x=5, y=5 - это координаты левого верхнего угла рисунка в
         // системе координат формы: ось x - вниз, ось y - вправо
      }
   }
}
```

Как видно из текста программы, вначале указываем размеры формы с помощью свойств `Width` и `Height`, хотя размеры формы *удобно регулировать* на вкладке конструктора формы "визуально". Далее создаем объект `Рисунок` для работы с изображением с указанием пути к файлу рисунка. Затем обращаемся непосредственно к методу рисования изображения в форме `DrawImage`, извлекая графический объект `Graphics` из аргумента `e` процедуры `OnPaint`.

Фрагмент работы программы показан на рис. 5.2.

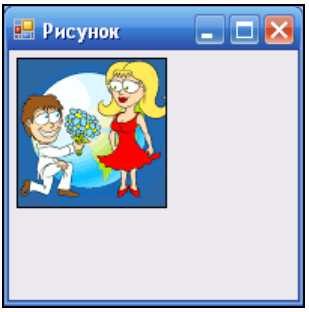


Рис. 5.2. Вывод растрового изображения в форму

Заметим, что это не единственный способ работы с графикой. Другой способ — это вызвать тот же метод `OnPaint` косвенно через событие формы `OnPaint`. Такой способ работы с графикой представлен в листинге 5.2. Создаем процедуру обработки данного события обычным образом, т. е. на вкладке конструктора формы в панели свойств **Properties** щелкнем значок молнии и в появившемся списке всех событий для объекта `Form1` выберем событие `Paint`. Объект `Graphics` получаем из аргумента `e` события `Paint`.

Листинг 5.2. Вывод растрового изображения в форму (вариант 2)

```
// Простейший вывод изображения в форму
using System.Drawing;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace Simple_Image2
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
         base.Text = "Рисунок";
      }
```

```
      private void Form1_Paint(object sender, PaintEventArgs e)
      {  // Событие рисования формы.
         // Создаем объект для работы с изображением
         Image Рисунок = new Bitmap("C:\\poryv.png");
         // Вывод изображения в форму
         e.Graphics.DrawImage(Рисунок, 5, 5);
      }
   }
}
```

Покажем *еще один способ* вывода графики в форму. В этом способе при щелчке на командной кнопке происходит непосредственное создание объекта класса `Graphics`. Программный код представлен в листинге 5.3.

Листинг 5.3. Вывод растрового изображения в форму (вариант 3)

```
using System;
using System.Drawing;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace Simple_Image3
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
         base.Text = "Рисунок";
         button1.Text = "Показать рисунок";
      }
      private void button1_Click(object sender, EventArgs e)
      {  // Событие "щелчок на кнопке"
         var Рисунок = new Bitmap("C:\\poryv.png");
         Graphics Графика = CreateGraphics();
         // или Graphics Графика = this.CreateGraphics();
         Графика.DrawImage(Рисунок, 5, 5);
      }
   }
}
```

Убедиться в работоспособности данных программ можно, открыв соответствующие решения в папках Simple_Image1, Simple_Image2 и Simple_Image3. В за-

ключение замечу, что с рассмотренными в данном разделе методами можно работать *не только* для вывода изображений графических файлов в форму, но и решать многие *другие задачи, связанные с графикой*.

Пример 31. Использование элемента *PictureBox* для отображения растрового файла с возможностью прокрутки

Обычно для отображения точечных рисунков, рисунков из метафайлов, значков, рисунков из файлов в форматах BMP, JPEG, GIF, PNG и проч. используется объект `PictureBox`. Часто рисунок оказывается слишком большим и не помещается целиком в пределах элемента управления **PictureBox**. Можно воспользоваться свойством элемента `SizeMode`, указав ему значение `StretchImage`. В этом случае изображение будет вытягиваться или сужаться, чтобы в точности соответствовать размеру **PictureBox**. Чаще всего такой подход не устраивает разработчика, поскольку изображение значительно деформируется. Напрашивается решение в организации возможности прокрутки изобажения (`AutoScroll`), но такого свойства у **PictureBox** нет. Зато такое свойство есть у элемента управления **Panel**. То есть, разместив **PictureBox** на элементе **Panel** с установленным свойством `AutoScroll = true` и при этом для **PictureBox** указав `SizeMode = AutoSize`, имеем шанс решить задачу прокрутки изображения.

Запустим Visual Studio 2010, выберем проект шаблона **Windows Forms Application C#**. Из панели **Toolbox** перетащим на форму элемент управления **Panel**, а на него поместим элемент **PictureBox**. Далее перейдем на вкладку программного кода и введем текст, представленный на листинге 5.4.

Листинг 5.4. Вывод изображения на *PictureBox* с возможностью прокрутки

```
// Программа выводит изображение из растрового файла в PictureBox,
// размещенный на элементе управления Panel, с возможностью прокрутки
// изображения
using System.Drawing;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
 namespace БольшойРисунокСкроллинг
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
```

```
            pictureBox1.Image = Image.FromFile("C:\\Pis.JPG");
            // или Bitmap Изображение = new Bitmap("C:\\011.BMP");
            // а затем pictureBox1.Image = (Image)Изображение;
            pictureBox1.SizeMode = PictureBoxSizeMode.AutoSize;
            panel1.AutoScroll = true;
            this.Text = "Скроллинг";
        }
    }
}
```

Фрагмент работы программы приведен на рис. 5.3.

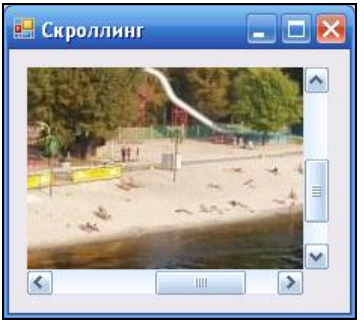


Рис. 5.3. Вывод изображения с возможностью прокрутки

Убедиться в работоспособности данной программы можно, открыв соответствующее решение в папке БольшойРисунокСкроллинг.

Пример 32. Рисование в форме указателем мыши

В данном примере покажем, как можно рисовать мышью в форме. То есть задача состоит в том, чтобы написать программу, позволяющую при нажатой левой или правой кнопке мыши рисовать в форме. Если пользователь отпустит кнопку мыши, то рисование прекращается. В проектируемой форме следует предусмотреть кнопку **Стереть**, которая предназначена для очистки формы.

Вначале надо создать форму с командной кнопкой, как мы это делали прежде. Текст программы представлен в листинге 5.5.

Листинг 5.5. Рисование на форме указателем мыши

```
// Программа позволяет при нажатой левой или правой кнопке мыши
// рисовать в форме
```

```
using System;
using System.Drawing;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace РисМышью
{
   public partial class Form1 : Form
   {
      bool Рисовать_ли = false; // Не рисовать
      public Form1()
      {
         InitializeComponent();
         this.Text = "Рисую мышью в форме";
         button1.Text = "Стереть";
      }
      private void Form1_MouseDown(object sender, MouseEventArgs e)
      {  // Если нажата кнопка мыши - MouseDown, то рисовать
         Рисовать_ли = true;
      }
      private void Form1_MouseUp(object sender, MouseEventArgs e)
      {  // Если кнопку мыши отпустили, то НЕ рисовать
         Рисовать_ли = false;
      }
      private void Form1_MouseMove(object sender, MouseEventArgs e)
      {  // Рисование прямоугольника, если нажата кнопка мыши
         if (Рисовать_ли == true)
         {  // Рисовать прямоугольник в точке (e.X, e.Y)
            Graphics Графика = CreateGraphics();
            Графика.FillRectangle(new SolidBrush(Color.Red),
                                  e.X, e.Y, 10, 10);
            // 10x10 пикселов — размер сплошного прямоугольника
            // e.X, e.Y — координаты указателя мыши
         }
      }
      private void button1_Click(object sender, EventArgs e)
      {  // Стереть, методы очистки формы:
         Graphics Графика = CreateGraphics();
         Графика.Clear(SystemColors.Control);
         // Графика.Clear(Color.FromName("Control"));
         // Графика.Clear(Color.FromKnownColor(KnownColor.Control));
         // Графика.Clear(ColorTranslator.FromHtml("#EFEBDE"));
```

```
            // this.Refresh();    // Этот метод перерисовывает форму
        }
    }
}
```

Здесь в начале программы объявлена переменная `Рисовать_ли` логического типа (`bool`) со значением `false`. Эта переменная либо позволяет (`Рисовать_ли = true`) рисовать в форме при перемещении мыши (событие `MouseMove`), либо не разрешает делать это (`Рисовать_ли = false`). Область действия переменной `Рисовать_ли` — весь класс `Form1`, т. е. изменить или выяснить ее значение можно в любой процедуре этого класса.

Значение переменной `Рисовать_ли` может изменить либо событие `MouseUp` (кнопку мыши отпустили, рисовать нельзя, `Рисовать_ли = false`), либо событие `MouseDown` (кнопку мыши нажали, рисовать можно, `Рисовать_ли = true`). При перемещении мыши с нажатой кнопкой программа создает графический объект `Graphics` пространства имен `System.Drawing`, используя метод `CreateGraphics()`, и рисует прямоугольник `FillRectangle()`, заполненный красным цветом, размером 10×10 пикселов. `e.X`, `e.Y` — координаты указателя мыши, которые так же являются координатами левого верхнего угла прямоугольника.

На рис. 5.4 приведен пример рисования в форме. Чтобы стереть все нарисованное в форме, следует нажать кнопку **Стереть**. При этом вызывается метод `Refresh()`, предназначенный для *перерисовывания* формы. Здесь в комментарии приведены варианты реализации очистки формы от всего нарисованного на ней пользователем. Например, путем создания графического объекта `CreateGraphics()` и закрашивания формы в ее первоначальный цвет `KnownColor.Control`.

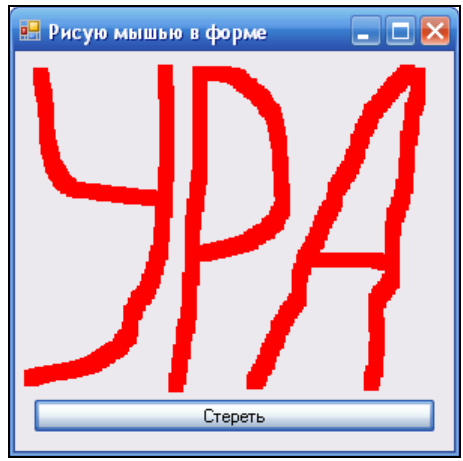


Рис. 5.4. Рисование мышью в форме

Заметим, что можно было бы очистить область рисования более короткой командой `Clear(Color.White)`, т. е. закрасить форму белым цветом (`White`) либо

выбрать другой цвет из списка 146 цветов после ввода точки за словом `Color`. Однако ни один из 146 цветов не является первоначальным цветом формы (`BackColor`). Поэтому задаем этот цвет через другие константы цвета, представленные в перечислении `Color.FromKnownColor`. Также можно задать цвет как `Color.FromName("Control")`. Можно использовать функцию перевода шестнадцатеричного кода цвета `ColorTranslator.FromHtml()`. Оба эти варианта представлены в комментарии. Цвет `#EFEBDE` является шестнадцатеричным представлением нужного нам цвета.

Очистить форму от всего нарисованного на ней можно также, свернув ее, а затем восстановив. Рисовать в форме можно как левой, так и правой кнопками мыши.

Убедиться в работоспособности программы можно, открыв решение РисМышью.sln в папке РисМышью.

Пример 33. Рисование в форме графических примитивов (фигур)

В векторных чертежах *графическим примитивом* называют элементарные составляющие чертежа: отрезок, дуга, символ, окружность и др. Здесь мы имеем дело с растровой графикой, но в данном случае подход тот же — по координатам рисуем те же фигуры. Система координат такая: начало координат — это левый верхний угол формы, ось *Ox* направлена вправо, а *Oy* — вниз.

Наша задача состоит в том, чтобы рисовать в форме окружность, отрезок, прямоугольник, сектор, текст, эллипс и закрашенный сектор. Выбор того или иного графического примитива осуществить через элемент управления **ListBox** (Список). Причем при рисовании очередного графического примитива нужно "стереть" предыдущий рисунок.

Для решения этой задачи создадим форму и перетащим в нее из окна **Toolbox** элемент управления **ListBox**. Далее — двойной щелчок в пределах формы, где сразу после выполнения процедуры `InitializeComponent` *создадим список* графических примитивов, заполняя коллекцию (`Items`) элементов списка `listBox1` (листинг 5.6).

Листинг 5.6. Рисование на форме графических примитивов

```
// Программа позволяет рисовать в форме графические примитивы:
// окружность, отрезок, прямоугольник, сектор, текст, эллипс и
// закрашенный сектор. Выбор того или иного графического примитива
// осуществляется с помощью элемента управления ListBox
using System;
using System.Drawing;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются
```

```
// в данной программе
namespace РисФигур
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
         this.Text = "Выбери графический примитив";
         listBox1.Items.AddRange(new Object[] {"Окружность", "Отрезок",
            "Прямоугольник", "Сектор", "Текст", "Эллипс",
            "Закрашенный сектор"});
         Font = new Font("Times New Roman", 14);
      }
      private void listBox1_SelectedIndexChanged(object sender, EventArgs e)
      {  // Здесь вместо этого события можно было бы обработать
         // событие listBox1.Click.
         // Создание графического объекта
         Graphics Графика = this.CreateGraphics();
         // Создание пера для рисования им фигур
         Pen Перо = new Pen(Color.Red);
         // Создание кисти для "закрашивания" фигур
         Brush Кисть = new SolidBrush(Color.Red);
         // Очистка области рисования путем ее окрашивания
         // в первоначальный цвет формы
         Графика.Clear(SystemColors.Control);
         // или Графика.Clear(Color.FromName("Control"));
         // или Графика.Clear(ColorTranslator.FromHtml("#EFEBDE"))
         switch (listBox1.SelectedIndex) // Выбор фигуры:
         {
            case 0:   // - выбрана окружность:
               Графика.DrawEllipse(Перо, 50, 50, 150, 150); break;
            case 1:   // - выбран отрезок:
               Графика.DrawLine(Перо, 50, 50, 200, 200); break;
            case 2:   // - выбран прямоугольник:
               Графика.DrawRectangle(Перо, 50, 30, 150, 180); break;
            case 3:   // - выбран сектор:
               Графика.DrawPie(Перо, 40, 50, 200, 200, 180, 225); break;
            case 4:   // - выбран текст:
               string s = "Каждый во что-то верит, но\n" +
                          "жизнь преподносит сюрпризы.";
```

```
                Графика.DrawString(s, Font, Кисть, 10, 100); break;
            case 5:   // - выбран эллипс:
                Графика.DrawEllipse(Перо, 30, 30, 150, 200); break;
            case 6:   // - выбран закрашеный сектор:
                Графика.FillPie(Кисть, 20, 50, 150, 150, 0, 45); break;
        }
    }
  }
}
```

В программе, обрабатывая событие изменения выбранного индекса в списке `listBox1` (хотя с таким же успехом в этой ситуации можно обрабатывать щелчок на выбранном элементе списка), создаем графический объект `Графика`, перо для рисования им фигур и кисть для "закрашивания" ею фигур. Далее очищаем область рисования путем окрашивания формы в первоначальный цвет `"Control"` или `"#EFEBDE"` (как записано в комментарии), используя метод `Clear()` объекта `Графика`:

```
Графика.Clear(SystemColors.Control);
```

При очищении области рисования оставляем цвет формы первоначальным — `"Control"`. Кстати, этот цвет можно назвать цветом Microsoft: это цвет Windows Explorer, Internet Explorer и проч.

После очистки формы, используя свойство `SelectIndex`, которое указывает на номер выбранного пользователем элемента списка (от 0 до 6), рисуем выбранную фигуру. На рис. 5.5 представлен фрагмент работы программы.

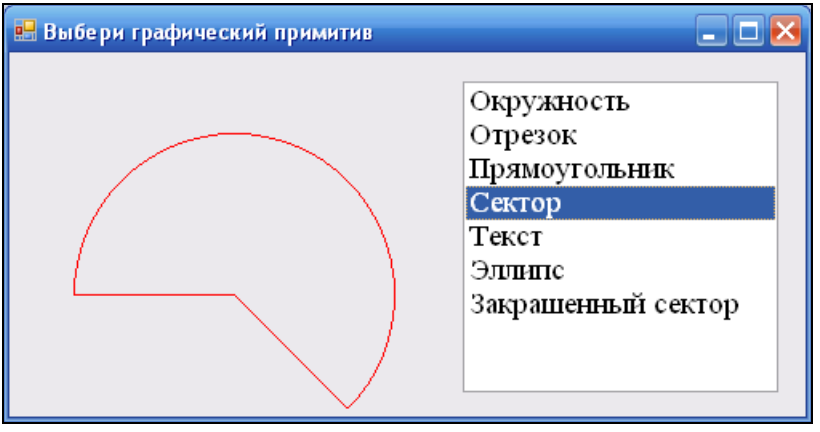


Рис. 5.5. Рисование графического примитива на форме

Убедиться в работоспособности программы можно, открыв решение РисФигур.sln в папке РисФигур.

Пример 34. Выбор цвета с использованием *ListBox*

В этом примере мы ставим задачу написать программу, которая *меня*ет цвет фона формы `BackColor`, перебирая константы цвета, предусмотренные в Visual Studio 2010, с помощью элемента управления **ListBox**.

Для решения данной задачи запустим Visual Studio 2010, выберем проект шаблона **Windows Forms Application C#**. На вкладке дизайнера формы из панели элементов **Toolbox** перетащим на форму список элементов **ListBox**. На вкладке программного кода **Form1.cs** введем текст, представленный в листинге 5.7.

Листинг 5.7. Выбор цвета с помощью элемента управления *ListBox* (вариант 1)

```
// Программа меняет цвет фона формы BackColor, перебирая константы цвета,
// предусмотренные в Visual Studio 2010, с помощью элемента управления ListBox
using System;
using System.Drawing;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace ВыборЦвета1
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
         // Получаем массив строк имен цветов из перечисления KnownColor
         string[] ВсеЦвета = Enum.GetNames(typeof(KnownColor));
         listBox1.Items.Clear();
         // Добавляем имена всех цветов в список listBox1:
         listBox1.Items.AddRange(ВсеЦвета);
         // Сортировать записи в алфавитном порядке
         listBox1.Sorted = true;
      }
      private void listBox1_SelectedIndexChanged(object sender, EventArgs e)
      {  // Цвет Transparent является "прозрачным", он не поддерживается
         // для формы:
         if (listBox1.Text == "Transparent") return;
         this.BackColor = Color.FromName(listBox1.Text);
         this.Text = "Цвет: " + listBox1.Text;
      }
   }
}
```

Как видно из программного кода, сразу после вызова `InitializeComponent`, используя метод `Enum.GetNames`, получим массив имен цветов в строковом представлении. Теперь этот массив очень легко добавить в список (коллекцию) `ListBox` методом `AddRange`. Если вы еще не написали обработку события изменения выбранного индекса, попробуйте уже на данном этапе запустить текущий проект (клавиша <F5>). Вы увидите форму и заполненные строки элемента управления **ListBox** цветами из перечисления `KnownColor`. Обрабатывая событие изменения выбранного индекса в списке **ListBox**, предпоследней строкой назначаем выбранный пользователем цвет формы (`BackColor`). Один из цветов перечисления `KnownColor` — цвет `Control` ("умалчиваемый" цвет формы), который является базовым цветом во многих программах Microsoft, в том числе Windows Explorer, Internet Explorer, Visual Studio 2010 и проч. Кроме того, здесь цветов больше, чем в константах цветов (структуре) `Color` (в структуре `Color` нет цвета `Control`). Один из цветов — `Transparent` — является "прозрачным", и для фона формы он не поддерживается. Поэтому если пользователь выберет этот цвет, то произойдет выход из процедуры (`return`), и цвет формы не изменится.

На рис. 5.6 приведен пример работы программы. Как видно, пользователь выбрал цвет **Control**, который соответствует цвету формы по умолчанию.

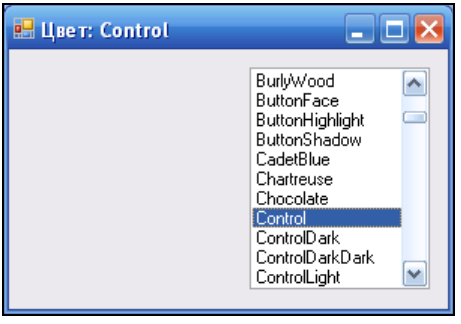


Рис. 5.6. Закраска формы выбранным цветом

Эту программу можно написать более элегантно с использованием цикла `foreach` при заполнении списка формы именами всех цветов (листинг 5.8).

Листинг 5.8. Выбор цвета с помощью элемента управления *ListBox* (вариант 2)

```
using System;
using System.Drawing;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace ВыборЦвета2
{
   public partial class Form1 : Form
```

```
   {
      public Form1()
      {
         InitializeComponent();
         // Получаем массив строк имен цветов из перечисления KnownColor.
         // Enum.GetNames возвращает массив имен констант
         // в указанном перечислении.
         string[] ВсеЦвета = Enum.GetNames(typeof(KnownColor));
         listBox1.Items.Clear();
         // Добавляем имена всех цветов в список listBox1:
         foreach (string s in ВсеЦвета)
            if (s != "Transparent") listBox1.Items.Add(s);
            // Цвет Transparent является "прозрачным",
            // он не поддерживается для формы
      }
      private void listBox1_SelectedIndexChanged(object sender, EventArgs e)
      {
         this.BackColor = Color.FromName(listBox1.Text);
         this.Text = "Цвет: " + listBox1.Text;
      }
   }
}
```

Как видно, цикл `foreach` обеспечивает заполнение списка `listBox1` именами цветов в строковом представлении кроме цвета `Transparent`, поэтому теперь его даже не надо "отсеивать" в процедуре обработки события изменения выбранного индекса.

Мы упомянули 167 констант или 146 цветов из структуры `Color`. Вообще говоря, в Visual Studio 2010 можно управлять гораздо бóльшим количеством цветов. Система программирования Visual Studio работает с так называемой *RGB-моделью* управления цветом. Согласно этой модели любой цвет может быть представлен как комбинация красного (Red), зеленого (Green) и синего (Blue) цветов. Долю каждого цвета записывают в один байт: 0 означает отсутствие доли этого цвета, а максимум (255) — максимальное присутствие этого цвета в общей сумме, т. е. в результирующем цвете. Например, функция `Color.FromArgb(int red, int green, int blue)` возвращает цвет, базируясь на этой модели. Информация о цвете элементарной точки (пиксела) может быть записана в три байта, т. е. 24 бита. При этом говорят, что глубина цвета равна 24 разрядам. Максимальное число, которое можно записать в 24 бита, равно $2^{24} - 1 = 16\,777\,215$ или приблизительно 17 млн. Это означает, что при глубине цвета, равной 24, можно управлять 17 млн цветов (цветовых оттенков).

Предлагаю следующую технологию использования любого цвета при разработке программ. Вначале выбираем цвет, который нам хотелось бы использовать (при

запуске каких-либо программ или в Интернете на каком-либо сайте). Существуют программы, сообщающие цвет пиксела, на котором находится курсор мыши, в нескольких принятых кодировках. Одну такую очень маленькую бесплатную программку Pixie вы можете скачать из Интернета по адресу: **http://natty.port5.com** или **http://www.nattyware.com**. Программа сообщает вам цвет пиксела в нескольких форматах, в том числе в формате HTML, например `#EFEBDE` (этот цвет соответствует цвету `Control`). Этот цвет подаете на вход функции `ColorTranslator.FromHtml()` для перевода в цвет, понятный той или иной процедуре (методу) C#.

Убедиться в работоспособности этих программ можно, открыв соответствующие решения в папках ВыборЦвета1 и ВыборЦвета2.

Пример 35. Печать графических примитивов

В данном разделе приведен пример вывода на печать (на принтер) изображения эллипса. Понятно, что таким же образом можно распечатывать и другие графические примитивы: прямоугольники, отрезки, дуги и т. д. (см. методы объекта `Graphics`). Для написания данной программы из панели элементов **Toolbox** в форму перенесем элемент управления **PrintDocument**. Текст программы приведен в листинге 5.9.

Листинг 5.9. Печать графических примитивов

```
// Программа выводит на печать (на принтер) изображение эллипса. Понятно, что
// таким же образом можно распечатывать и другие графические примитивы:
// прямоугольники, отрезки, дуги и т. д. (см. методы объекта Graphics)
using System.Drawing;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace ПечатьЭллипса
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
         printDocument1.Print();
      }
      private void printDocument1_PrintPage(object sender,
                  System.Drawing.Printing.PrintPageEventArgs e)
```

```
      {
         // Выводится на печать эллипс красного цвета внутри
         // ограничивающего прямоугольника с вершиной в точке (200, 250),
         // шириной 300 и высотой 200
         Pen Перо = new Pen(Color.Red);
         //e.Graphics.DrawEllipse(Перо, new Rectangle(200, 250, 300, 200));
         e.Graphics.DrawEllipse(Перо, 50, 50, 150, 150);
      }
   }
}
```

Как видно, с целью максимального упрощения программы для генерации события `PrintPage` сразу после выполнения процедуры `InitializeComponent` вызываем метод `PrintDocument1.Print`. В обработчике события `PrintPage` вызываем метод `DrawEllipse` для построения эллипса без заливки. В комментарии приведен вариант построения эллипса другим способом.

Убедиться в работоспособности программы можно, открыв решение ПечатьЭллипса.sln в папке ПечатьЭллипса.

Пример 36. Печать BMP-файла

В данном разделе программа выводит на печать графический файл формата BMP. На логическом диске C: заранее подготовим графический файл формата BMP и назовем его C:\pic.bmp. Этот файл будет распечатываться на принтере программой, которую мы напишем. Итак, запустим Visual Studio 2010, выберем шаблон **Windows Forms Application C#**. Затем добавим в форму из панели элементов **Toolbox** командную кнопку **Button** и объект **PrintDocument**. Программный код представлен в листинг 5.10.

Листинг 5.10. Печать BMP-файла

```
// Эта программа выводит на печать файл с расширением bmp
using System;
using System.Drawing;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace ПечатьBMPфайла
{
   public partial class Form1 : Form
   {
      public Form1()
```

```
      {
         InitializeComponent();
         this.Text = @"Печать файла C:\pic.bmp";
         button1.Text = "Печать";
      }
      private void button1_Click(object sender, EventArgs e)
      {  // Пользователь щелкнул на кнопке
         try
         {
            printDocument1.Print();
         }
         catch (Exception exc)
         {
            MessageBox.Show("Ошибка печати на принтере\n", exc.Message);
         }
      }
      private void printDocument1_PrintPage(object sender,
                  System.Drawing.Printing.PrintPageEventArgs e)
      {  // Это событие возникает, когда вызывают метод Print().
         // Рисование содержимого BMP-файла
         e.Graphics.DrawImage(Image.FromFile(@"C:\pic.bmp"),
                              e.Graphics.VisibleClipBounds);
         // Следует ли распечатывать следующую страницу?
         e.HasMorePages = false;
      }
   }
}
```

Как видно, при нажатии пользователем кнопки вызывается метод `printDocument1.Print`. Этот метод создает событие `PrintPage`, которое обрабатывается в обработчике `printDocument1_PrintPage`. Для вывода на принтер вызывается метод `DrawImage` рисования содержимого BMP-файла.

Убедиться в работоспособности программы можно, открыв решение ПечатьBMPфайла.sln в папке ПечатьBMPфайла.

Пример 37. Построение графика

Задача состоит в том, чтобы, используя в качестве исходных данных, например, объемы продаж каких-либо товаров по месяцам, построить график по точкам. Понятно, что таким же образом можно построить любой график для других прикладных целей.

Для решения этой задачи запустим Visual Studio 2010, далее выберем новый проект шаблона **Windows Forms Application C#**, при этом получим стандартную форму. Перенесем из панели элементов **Toolbox** в форму элемент управления **PictureBox** и кнопку **Button**. Далее перейдем на вкладку программного кода и введем текст, представленный в листинге 5.11.

Листинг 5.11. Программа для вывода графика на форму

```
// Программа рисует график продаж по месяцам. Понятно, что таким же образом
// можно построить любой график по точкам для других прикладных целей
using System;
using System.Drawing;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace График
{
   public partial class Form1 : Form
   {
      // Исходные данные для построения графика (то есть исходные точки):
      string[] Months = new string[] {"Янв", "Фев", "Март", "Апр", "Май",
                                      "Июнь", "Июль", "Авг", "Сент",
                                      "Окт", "Нояб", "Дек"};
      int[] Sales = new int[] {335, 414, 572, 629, 750, 931,
                               753, 599, 422, 301, 245, 155};
      Graphics Графика;
      // Далее, создаем объект Bitmap, который имеет
      // тот же размер и разрешение, что и PictureBox
      Bitmap Растр;
      int ОтступСлева = 35; int ОтступСправа = 15;
      int ОтступСнизу = 20; int ОтступСверху = 10;
      int ДлинаВертОси, ДлинаГоризОси, YГоризОси, Xmax, XНачЭпюры;
      // Шаг градуировки по горизонтальной и вертикальной осям:
      double ГоризШаг; int ВертШаг;
      //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
      public Form1()
      {
         InitializeComponent();
         this.Text = " Построение графика ";
         button1.Text = "Нарисовать график";
         Растр = new Bitmap(pictureBox1.Width, pictureBox1.Height,
                            pictureBox1.CreateGraphics());
```

```
    // pictureBox1.BorderStyle = BorderStyle.FixedSingle;
    YГоризОси = pictureBox1.Height - ОтступСнизу;
    Xmax = pictureBox1.Width - ОтступСправа;
    ДлинаГоризОси = pictureBox1.Width - (ОтступСлева + ОтступСправа);
    ДлинаВертОси = YГоризОси - ОтступСверху;
    ГоризШаг = (double)(ДлинаГоризОси / Sales.Length);
    ВертШаг = (int)(ДлинаВертОси / 10);
    XНачЭпюры = ОтступСлева + 30;
} //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
private void button1_Click(object sender, EventArgs e)
{  // Обработка события "щелчок на кнопке":
    Графика = Graphics.FromImage(Растр);
    РисуемОси();
    РисуемГоризЛинии();
    РисуемВертЛинии();
    РисованиеЭпюры();
    pictureBox1.Image = Растр;
    Графика.Dispose();
} //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
private void РисуемОси()
{
    Pen Перо = new Pen(Color.Black, 2);
    // Рисование вертикальной оси координат:
    Графика.DrawLine(Перо, ОтступСлева, YГоризОси,
                     ОтступСлева, ОтступСверху);
    // Рисование горизонтальной оси координат:
    Графика.DrawLine(Перо, ОтступСлева, YГоризОси,
                           Xmax, YГоризОси);
    for (int i = 1; i <= 10; i++)
    {  // Рисуем "усики" на вертикальной координатной оси:
       int Y = YГоризОси - i * ВертШаг;
       Графика.DrawLine(Перо, ОтступСлева - 5, Y, ОтступСлева, Y);
       // Подписываем значения продаж через каждые 100 единиц:
       Графика.DrawString((i * 100).ToString(), new Font("Arial", 8),
                           Brushes.Black, 2, Y - 5);
    }
    // Подписываем месяцы на горизонтальной оси:
    for (int i = 0; i <= Months.Length - 1; i++)
       Графика.DrawString(Months[i], new Font("Arial", 8),
                          Brushes.Black,
              ОтступСлева + 18 + (int)(i * ГоризШаг), YГоризОси + 4);
} //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
private void РисуемГоризЛинии()
```

```
      {
         Pen ТонкоеПеро = new Pen(Color.LightGray, 1);
         for (int i = 1; i <= 10; i++)
         {  // Рисуем горизонтальные почти "прозрачные" линии:
            int Y = YГоризОси - ВертШаг * i;
            Графика.DrawLine(ТонкоеПеро, ОтступСлева + 3, Y, Xmax, Y);
         }
         ТонкоеПеро.Dispose();
      } //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
      private void РисуемВертЛинии()
      {  // Рисуем вертикальные почти "прозрачные" линии
         Pen ТонкоеПеро = new Pen(Color.Bisque, 1);
         for (int i = 0; i <= Months.Length - 1; i++)
         {
            int X = XНачЭпюры + (int)(ГоризШаг * i);
            Графика.DrawLine(ТонкоеПеро, X, ОтступСверху, X, YГоризОси - 4);
         }
         ТонкоеПеро.Dispose();
      } //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
      private void РисованиеЭпюры()
      {
         double ВертМасштаб = (double)ДлинаВертОси / 1000;
         int[] Y = new int[Sales.Length]; // значения ординат на экране
         int[] X = new int[Sales.Length]; // значения абсцисс на экране
         for (int i = 0; i <= Sales.Length - 1; i++)
         {  // Вычисляем графические координаты точек:
            Y[i] = YГоризОси - (int)(Sales[i] * ВертМасштаб);
            // Отнимаем значения продаж, поскольку ось Y экрана
            // направлена вниз
            X[i] = XНачЭпюры + (int)(ГоризШаг * i);
         }
         // Рисуем первый кружок:
         Pen Перо = new Pen(Color.Blue, 3);
         Графика.DrawEllipse(Перо, X[0] - 2, Y[0] - 2, 4, 4);
         for (int i = 0; i <= Sales.Length - 2; i++)
         {  // Цикл по линиям между точками:
            Графика.DrawLine(Перо, X[i], Y[i], X[i + 1], Y[i + 1]);
            // Отнимаем 2, поскольку диаметр (ширина) точки = 4:
            Графика.DrawEllipse(Перо, X[i + 1] - 2, Y[i + 1] - 2, 4, 4);
         }
      }
   }
}
```

Как видно из текста программы, вначале объявляем некоторые переменные, чтобы они были видны из всех процедур класса. Строковый массив `Months` содержит названия месяцев, которые пользователь нашего программного кода может менять в зависимости от контекста строящегося графика. В любом случае записанные строки в этом массиве будут отображаться по горизонтальной оси графика. Массив целых чисел `Sales` содержит объемы продаж по каждому месяцу, они соответствуют вертикальным ординатам графика. Оба массива должны иметь между собой одинаковую размерность, но не обязательно равную двенадцати.

При обработке события "щелчок мыши на кнопке" **Button** создаем объект класса `Graphics`, используя элемент управления **PictureBox**, а затем, вызывая соответствующие процедуры, поэтапно рисуем координатные оси, сетку из горизонтальных и вертикальных линий и непосредственно эпюру. Чтобы успешно, минимальными усилиями, с возможностью дальнейшего совершенствования программы построить график, следует как можно более понятно назвать некоторые ключевые, часто встречающиеся интервалы и координаты на рисунке. Из названия этих интервалов будет следовать смысл. Скажем, переменная `ОтступСлева` хранит число пикселов, на которое следует отступать, чтобы строить на графике, например, вертикальную ось продаж. Кроме очевидных названий упомянем переменную `YГоризОси`, это графическая ордината (ось *x* направлена слева направо, а ось *y* — сверху вниз) горизонтальной оси графика, на которой подписываются месяцы. Переменная `XMax` содержит в себе значение максимальной абсциссы (см. рис. 5.7), правее которой уже никаких построений нет. Переменная `XНачЭпюры` — это значение абсциссы первой построенной точки графика. Используя такие понятные из контекста названия переменных, да еще и на русском языке, мы значительно облегчаем весь процесс программирования, упрощаем сопровождение и модификацию программы.

Построенный данной программой график показан на рис. 5.7.

Убедиться в работоспособности программы можно, открыв решение График.sln в папке График.

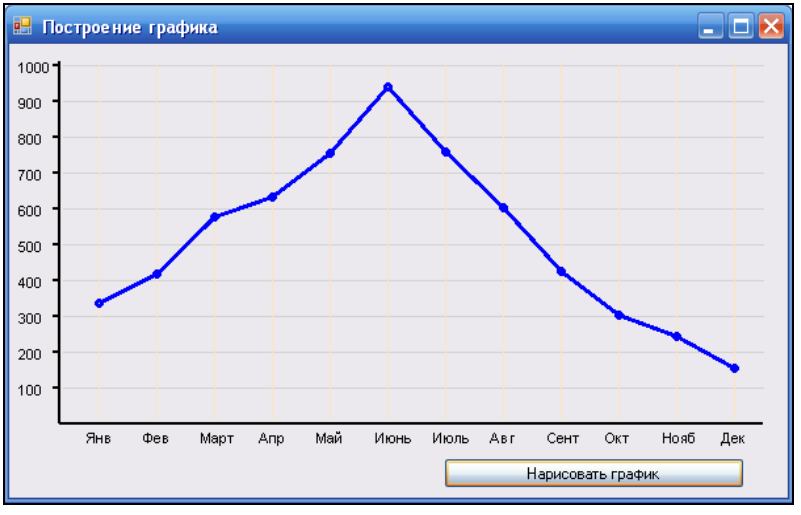


Рис 5.7. График объемов продаж по месяцам

Глава 6

Управление буфером обмена с данными в текстовом и графическом форматах

Пример 38. Буфер обмена с данными в текстовом формате

Напишем программу для управления буфером обмена с данными в текстовом формате. Эта программа будет позволять записывать какой-либо текст в буфер обмена (БО), а затем извлекать этот текст из БО. Для этой цели 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 какой-либо текст в БО, нужно записать его в верхнее поле, выделить (с помощью клавиш управления курсором при нажатой клавише <Shift>), а затем нажать кнопку **Записать в БО**. Нажимая кнопку **Записать в БО**, мы как бы моделируем комбинацию клавиш <Ctrl>+<C>.

Далее записанный в БО текст можно читать в каком-либо текстовом редакторе или вывести в нижнее текстовое поле прямо в нашей форме. Для этого служит кнопк **Извлечь из БО** (см. рис. 6.1). Текст данной программы приведен в листинге 6.1.

Листинг 6.1. Запись текстовых данных в буфер обмена и их чтение

```
// Эта программа имеет возможность записи какого-либо текста
// в буфер обмена, а затем извлечения этого текста из буфера обмена
using System;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace БуферОбменаTXT
```

```
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
         this.Text = "Введите текст в верхнее поле";
         textBox1.Clear();  textBox2.Clear();  textBox1.TabIndex = 0;
         button1.Text = "Записать в БО";
         button2.Text = "Извлечь из БО";
      }
      private void button1_Click(object sender, EventArgs e)
      {  // Записать выделенный в верхнем поле текст в БО
         if (textBox1.SelectedText != String.Empty)
         {
            Clipboard.SetDataObject(textBox1.SelectedText);
            textBox2.Text = String.Empty;
         }
         else
            textBox2.Text = "В верхнем поле текст не выделен";
      }

      private void button2_Click(object sender, EventArgs e)
      {  // Объявление объекта-получателя из БО
         IDataObject Получатель = Clipboard.GetDataObject();
         // Если данные в БО представлены в текстовом формате...
         if (Получатель.GetDataPresent(DataFormats.Text) == true)
            // то записать их в Text тоже в текстовом формате
            textBox2.Text = Получатель.GetData(DataFormats.Text).ToString();
         else
            textBox2.Text = "Запишите что-либо в буфер обмена";
      }
   }
}
```

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, нажимая кнопку **Извлечь из БО**, в нижнем поле пользователь нашей программы должен увидеть содержимое буфера обмена. Для этого объявляем объектную переменную `Получатель`. Это объект-получатель из буфера обмена.

Данная 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.

Пример работы приложения показан на рис. 6.1.

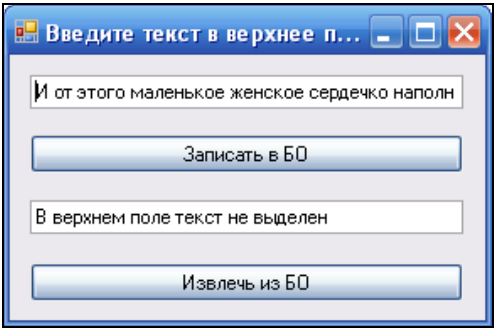


Рис. 6.1. Запись текстовых данных в буфер обмена и их чтение

Убедиться в работоспособности программы можно, открыв решение БуферОбменаTXT.sln в папке БуферОбменаTXT.

Пример 39. Элемент управления *PictureBox*. Буфер обмена с растровыми данными

Обсудим программу, которая оперирует буфером обмена, когда тот содержит изображение. Здесь мы будем использовать элемент управления **PictureBox** (графическое окно), который способен отображать в своем поле растровые файлы различных форматов, в том числе BMP, JPEG, PNG, GIF и др. Возможность отображения в **PictureBox** GIF-файлов позволяет просматривать анимацию, т. е. движущееся изображение, в экранной форме вашей программы. Пример отображения такой GIF-анимации вы можете посмотреть, открыв решение PictureBoxGif.sln в папке PictureBoxGif. Однако элемент управления **PictureBox** не способен отображать Flash-файлы в формате SWF, которые сочетают в себе векторную графику, растровую графику и воспроизведение звука. Замечу, что файлы SWF воспроизводит элемент управления **Microsoft WebBrowser**, который мы обсудим в *главе 8*.

Рассматриваемая программа выводит в поле элемента управления **PictureBox** изображение из растрового файла (например, PNG). При этом изображение записывается в БО. Пользователь может убедиться в этом, например, запустив Paint — стандартный графический редактор ОС Windows. Далее пользователь может поместить в БО любое изображение с помощью какого-нибудь графического редак-

тора, например того же Paint, ACDSee или др. Затем, нажав кнопку **Извлечь из БО** нашей программы, получить в форме содержимое БО.

Для создания данной программы в форму из панели **Toolbox** перетащим элементы управления **PictureBox** (графическое окно) и кнопку **Button**. Текст программы приведен в листинге 6.2.

Листинг 6.2. Обмен графическими данными через буфер обмена

```
// Программа оперирует буфером обмена, когда тот содержит изображение
using System;
using System.Drawing;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace БуферОбменаBitmap
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
         base.Text = "Содержимое БО:"; button1.Text = "Извлечь из БО";
         // Записать в PictureBox изображение из файла
         pictureBox1.Image = Image.FromFile(
          System.IO.Directory.GetCurrentDirectory() + @"\poryv.png");
         // Записать в БО изображение из графического окна формы
         Clipboard.SetDataObject(pictureBox1.Image);
      }
      private void button1_Click(object sender, EventArgs e)
      {  // Объявление объекта-получателя из буфера обмена
         IDataObject Получатель = Clipboard.GetDataObject();
         Bitmap Растр;
         // Если данные в БО представлены в формате Bitmap...
         if (Получатель.GetDataPresent(DataFormats.Bitmap) == true)
         {  // то записать эти данные из БО в переменную Растр
            // в формате Bitmap
            Растр = (Bitmap)Получатель.GetData(DataFormats.Bitmap);
            pictureBox1.Image = Растр;
         }
      }
   }
}
```

Как видно из текста программы, при загрузке программы в графическое окно формы `pictureBox1` записываем какой-нибудь файл, например poryv.png. Этот файл необходимо поместить в ту же папку, где расположен exe-файл данной программы, т. е. в папку Debug (при отладке приложения). Функция `Directory.GetCurrentDirectory()` возвращает полный путь текущей папки. Данная функция относится к пространству имен `System.IO`. Далее по команде

```
Clipboard.SetDataObject(pictureBox1.Image)
```

происходит запись содержимого графического окна в буфер обмена.

Теперь пользователь может проверить содержимое БО с помощью какого-либо графического редактора, например Paint. Далее пользователь может записать что-либо в буфер обмена, опять же используя какой-нибудь графический редактор. Нажатие кнопки **Извлечь из БО** нашей программы в форме приведет к появлению изображения, находящегося в буфере обмена. Фрагмент работы программы представлен на рис. 6.2.

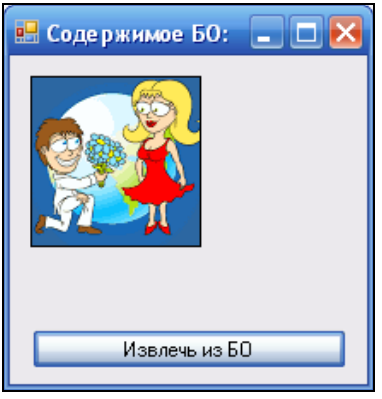


Рис. 6.2. Извлечение изображения из буфера обмена

В программном коде при обработке события "щелчок на кнопке" **Извлечь из БО** объявляем объектную переменную `Получатель`. Это объект, с помощью которого будем получать изображение, записанное в буфер обмена. Эта переменная сознательно названа по-русски для большей выразительности. Далее следует проверка: записаны ли данные, представленные в БО, в формате растровой графики `Bitmap`. Если да, то записать данные из БО в переменную `Растр` в формате `Bitmap` с помощью объектной переменной `Получатель`, используя неявное преобразование в переменную типа `Bitmap`. Далее, чтобы изображение появилось в элементе управления `pictureBox1` в форме, присваиваем свойству `Image` значение переменной `Растр`.

Убедиться в работоспособности программы можно, открыв решение БуферОбменаBitmap.sln в папке БуферОбменаBitmap.

Пример 40. Имитация нажатия комбинации клавиш <Alt>+<PrintScreen>. Вызов функции Microsoft API

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> можно использовать соответствующую функцию Microsoft API. В то же время 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>. В этом примере мы рассмотрим обе эти возможности.

Данная программа сознательно максимально упрощена, чтобы более выразительно показать ее суть. Здесь в форме имеем только две кнопки. При обработке события "щелчок на первой кнопке" будем имитировать нажатие комбинации клавиш <Alt>+<PrintScreen> с помощью функции Microsoft API, а при обработке события "щелчок на второй кнопке" будем имитировать нажатие этих же клавиш методом `Send` класса `SendKeys`. Результаты нажатия любой из кнопок будут совершенно одинаковы. При этом форму можно как угодно менять — растягивать по ширине и по высоте, но после нажатия любой из кнопок в форме изображение формы запишется в буфер обмена. Следует убедиться в этом: запустить Paint и извлечь графическую копию формы из буфера обмена. Текст программы представлен в листинге 6.3.

Листинг 6.3. Имитация нажатия комбинации клавиш <Alt>+<PrintScreen>

```
// Программная имитация нажатия клавиш <Alt>+<PrintScreen> с помощью
// функции Microsoft Windows API, а также методом Send класса SendKeys
using System;
using System.Windows.Forms;
using System.Runtime.InteropServices;  // - следует добавить эту директиву
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace AltPrintScreen
{
   public partial class Form1 : Form
   {  // Следует добавить пространство имен
```

```
      //   using System.Runtime.InteropServices.
      // Объявление функции Microsoft API
      [DllImport("user32.dll")]
      extern static void keybd_event(byte bVk, byte bScan,
                              int dwFlags, int dwExtraInfo);
      public Form1()
      {
         InitializeComponent();
      }
      private void button1_Click(object sender, EventArgs e)
      {  // Вызов функции Microsoft API
         keybd_event(44, 1, 0, 0);
      }
      private void button2_Click(object sender, EventArgs e)
      {  // Метод SendKeys.Send посылает сообщение активному приложению
         // о нажатии клавиш <Alt>+<PrintScreen>
         SendKeys.Send("%{PRTSC}");
         // string QQ = Keys.PrintScreen.ToString();
      }
   }
}
```

В этой программе при обработке события "щелчок на первой кнопке" мы воспользовались функцией Microsoft Windows API (Application Programming Interface, интерфейс программирования приложений). Функция `keybd_event()` позволяет имитировать (эмулировать) нажатие пользователем клавиш. Функция находится в динамической библиотеке user32.dll (см. программный код). В динамической библиотеке user32.dll также присутствуют управляющие функции Windows для обработки сообщений, таймеров, меню и взаимодействия. Эту библиотеку вы можете увидеть в системном каталоге C:\Windows\System32. В dll-файлах (dynamic link library) находятся скомпилированные классы.

Как видно, сначала функцию API следует объявить с указанием типов аргументов, подаваемых на вход функции `keybd_event()`. Параметр `"user32"` или `"user32.dll"` определяет имя используемой библиотеки. В скобках записан список аргументов функции `keybd_event()`. Мы можем задавать любые имена аргументов, но их типы должны соответствовать формату функции API.

При обработке события "щелчок на кнопке" `Button1_Click()` мы программируем вызов API-функции `keybd_event()` по формату, заданному в объявлении функции. Из четырех аргументов функции `keybd_event()` важным является первый, он задает код клавиши <PrintScreen>. Можно задать шестнадцатеричный код, равный `0xH2CS`, однако мы задали десятичный код клавиши <PrintScreen>, равный 44.

Также представляет интерес второй аргумент функции `keybd_event()`. Второй аргумент, равный единице, эмулирует одновременное нажатие клавиши <Alt> с клавишей <PrintScreen>. Таким образом, вызывая функцию `keybd_event(44, 1, 0, 0)`, мы эмулируем нажатие комбинации клавиш <Alt>+<PrintScreen>, а при `keybd_event(44, 0, 0, 0)` — нажатие только <PrintScreen>. С помощью API-функции `keybd_event()` можно эмулировать нажатие любых клавиш, подавая на вход функции коды соответствующих клавиш.

Обрабатывая событие "щелчок на второй кнопке", мы решаем абсолютно ту же задачу, но более современными средствами. Метод `Send` класса `SendKeys` посылает сообщение активному приложению о нажатии комбинации клавиш <Alt>+ +<PrintScreen>. Кодом клавиши <PrintScreen> является код `"{PRTSC}"`. Чтобы указать сочетание клавиши <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

Таким образом, результаты нажатия обеих клавиш одинаковы. Убедиться в работоспособности программы можно, открыв решение AltPrintScreen.sln в папке AltPrintScreen.

Пример 41. Запись содержимого буфера обмена в BMP-файл

Напишем программу, которая читает буфер обмена, и если данные в нем представлены в формате растровой графики, то записывает эти данные в BMP-файл. Текст этой программы приведен в листинге 6.4.

Листинг 6.4. Запись содержимого буфера обмена в BMP-файл

```
// Программа читает буфер обмена, и если данные в нем представлены
// в формате растровой графики, то записывает их в BMP-файл
using System;
using System.Drawing;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace БуферОбменаSaveBMP
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
```

```
            this.Text = "Сохраняю копию БО в BMP-файл";
            button1.Text = "Сохранить";
        }
        private void button1_Click(object sender, EventArgs e)
        {  // Объявление объекта-получателя из буфера обмена
            IDataObject Получатель = Clipboard.GetDataObject();
            Bitmap Растр;
            // Если данные в буфере обмена представлены в формате Bitmap...
            if (Получатель.GetDataPresent(DataFormats.Bitmap) == true)
            {  // то записать их из БО в переменную Растр в формате Bitmap
               Растр = (Bitmap)Получатель.GetData(DataFormats.Bitmap);
               // Сохранить изображение в файле Clip.bmp
               Растр.Save(@"C:\Clip.BMP");
               //this.Text = @"Сохранено в файле C:\Clip.BMP";
               //button1.Text = "Еще записать?";
               MessageBox.Show
                  (@"Изображение из БО записано в файл C:\Clip.BMP", "Успех");
            }
            else
             // В БО нет данных в формате изображений
               MessageBox.Show("В буфере обмена нет данных в формате Bitmap",
                               "Запишите какое-либо изображение в БО");
        }
    }
}
```

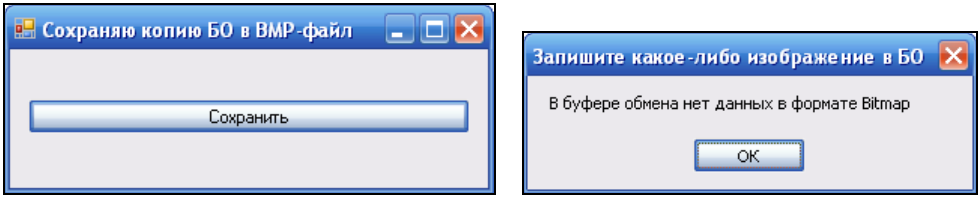


Рис. 6.3. Запись изображения из буфера обмена в файл

В программном коде при обработке события "щелчок на кнопке" **Сохранить** объявляем объектную переменную `Получатель`. Это объект-получатель из буфера обмена. Далее следует проверка: записаны ли данные, представленные в буфере обмена, в формате растровой графики `Bitmap`. Если да, то записать данные из буфера обмена в переменную `Растр` в формате `Bitmap` с помощью объектной переменной `Получатель`, используя неявное преобразование в переменную типа `Bitmap`. Сохранить изображение `Растр` на винчестер очень просто, воспользовавшись методом `Save()`. Чтобы не запутать читателя и упростить программу, автор не стал ор-

ганизовывать запись файла в диалоге. При необходимости читатель сделает это самостоятельно, воспользовавшись элементом управления **SaveFileDialog**. Фрагмент работы данной программы представлен на рис. 6.3.

Убедиться в работоспособности программы можно, открыв решение БуферОбменаSaveBMP.sln в папке БуферОбменаSaveBMP.

Пример 42. Использование таймера *Timer*

Автор несколько раз встречал задачу записи моментальных снимков экрана (screen shot) в растровые файлы с некоторым промежутком времени, например пять секунд. Кстати, такая задача была выставлена на тендер на сайте оффшорного программирования (**www.rentacoder.com**). За ее решение указывалось вознаграждение 100 долларов США. Конечно, сразу появляется очень много желающих эту задачу решить, но попробуйте выиграть тендер.

Следующая задача является частью упомянутой задачи, ее сформулируем таким образом: после запуска программы должна отображаться форма с элементом управления **ListBox**, а через две секунды в список добавляется запись "Прошло две секунды", далее через каждые две секунды в список добавляется аналогичная запись. На этой задаче мы освоим технологию работы с таймером.

Для реализации программы в форму из панели элементов **Toolbox** перетащим элемент управления **Timer** и список **ListBox**. Программный код приведен в листинге 6.5.

Листинг 6.5. Использование таймера

```
// Демонстрация использования таймера Timer. После запуска программы
// показываются форма и элемент управления список элементов ListBox.
// Через 2 секунды в списке элементов появляется запись "Прошло две секунды",
// и через каждые последующие 2 секунды в список добавляется аналогичная
// запись
using System;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace ПростоТаймер
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
         this.Text = "Timer";
```

```
            timer1.Interval = 2000; // - 2 секунды
            // Старт отчета времени
            timer1.Enabled = true;  // - время пошло
        }
        private void timer1_Tick(object sender, EventArgs e)
        {
            listBox1.Items.Add("Прошло две секунды");
        }
    }
}
```

Экземпляр класса **Timer** — это невидимый во время работы программы элемент управления, предназначенный для периодического генерирования события `Tick`. В программном коде сразу после инициализации конструктором компонентов программы задаем интервал времени (`Interval`), равный 2000 миллисекунд (две секунды). Каждые две секунды будет возникать событие `Tick`, при этом в список элементов `listBox1` будет добавляться запись "Прошло две секунды ". Этот текст будет выводиться каждые две секунды до тех пор, пока пользователь программы не щелкнет на кнопке формы **Закрыть**. На рис. 6.4 показан фрагмент работы программы.

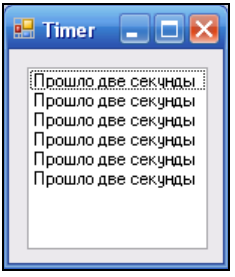


Рис. 6.4. Вывод сообщения

Убедиться в работоспособности программы можно, открыв решение ПростоТаймер.sln в папке ПростоТаймер.

Пример 43. Запись в файлы текущих состояний экрана каждые пять секунд

Как уже говорилось в предыдущем разделе, 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 данной задачи мы имеем уже весь инструментарий, он был рассмотрен в предыдущих задачах. А именно, задание интервала времени в пять се-

кунд с помощью 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 перетаскиваем в форму элементы управления **Timer** и **Button**. Текст программы приведен в листинге 6.6.

Листинг 6.6. Запись в файлы текущих состояний экрана с интервалом 5 секунд

```
// Программа после запуска каждые пять секунд делает снимок текущего
// состояния экрана и записывает эти снимки в файлы Pic1.BMP, Pic2.BMP
// и т. д. Количество таких записей в файл — пять
using System;
using System.Drawing;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace SaveСкриншотКаждые5сек
{
   public partial class Form1 : Form
   {
      int i = 0; // счет секунд
      public Form1()
      {
         InitializeComponent();
         this.Text = "Запись каждые 5 секунд в файл";
         button1.Text = "Пуск";
      }
      private void timer1_Tick(object sender, EventArgs e)
      {
         i = i + 1;
         this.Text = string.Format("Прошло {0} секунд", i);
         if (i >= 28) { timer1.Enabled = false; this.Close(); }
         if (i % 5 != 0) return;
         // Имитируем нажатие клавиш <Alt>+<PrintScreen>
         SendKeys.Send("%{PRTSC}");
         // Объявление объекта-получателя из буфера обмена
         IDataObject Получатель = Clipboard.GetDataObject();
         Bitmap Растр;
         // Если данные в буфере обмена представлены в формате Bitmap,
         // то записать
         if (Получатель.GetDataPresent(DataFormats.Bitmap) == true)
         {  // эти данные из буфера обмена в переменную Растр в формате Bitmap
```

```
            Растр = (Bitmap)Получатель.GetData(DataFormats.Bitmap);
            // Сохранить изображение из переменной Растр
            // в файл C:\Pic1, C:\Pic2, C:\Pic3, ...
            string ИмяФайла = string.Format(@"C:\Pic{0}.BMP", i / 5);
            Растр.Save(ИмяФайла);
         }
      }
      private void button1_Click(object sender, EventArgs e)
      {
         this.Text = string.Format("Прошло 0 секунд");
         timer1.Interval = 1000; // равно 1 секунде
         timer1.Enabled = true;  // время пошло
      }
   }
}
```

Как видно, 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, сколько раз программа сделает запись в буфер обмена, а из буфера обмена — в файл. 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.

Убедиться в работоспособности программы можно, открыв соответствующее решение в папке SaveСкриншотКаждые5сек.

Глава 7

Ввод и вывод табличных данных. Решение системы уравнений

Пример 44. Формирование таблицы. Функция *String.Format*

При создании инженерных, экономических и других приложений часто задача сводится к вводу данных, расчету (обработке введенных данных), а затем — выводу результатов вычислений в таблицу. В этом разделе мы обсудим такую типичную задачу: как оптимально сформировать таблицу, а затем вывести ее на экран с возможностью печати на принтере.

Чтобы выразительно показать именно процесс формирования таблицы, абстрагируемся от ввода данных и расчетов и сосредоточимся только на сути. Например, пусть мы имеем информацию о телефонах наших знакомых, и нам хотелось бы представить эту информацию в виде наглядной таблицы. Предположим, что результаты обработки записаны в два массива: массив имен знакомых `Imena` и массив телефонов `Tel`. Наша программа формирует таблицу из этих двух массивов в текстовом поле **TextBox**. Кроме того, в программе участвует элемент управления **MenuStrip** для организации раскрывающегося меню, с помощью которого пользователь выводит сформированную таблицу в Блокнот (notepad.exe) с целью последующей корректировки (редактирования) и вывода на печать.

Таким образом, новый проект будет содержать форму, текстовое поле **TextBox** со свойством `Multiline = true` и меню **MenuStrip**. Текст программы представлен в листинге 7.1.

Листинг 7.1. Формирование таблицы

```
// Программа формирует таблицу из двух строковых массивов в текстовом поле,
// используя функцию String.Format. Кроме того, в программе участвует
// элемент управления MenuStrip для организации выпадающего меню,
// с помощью которого пользователь выводит сформированную таблицу
// в Блокнот с целью последующего редактирования и вывода на печать
```

```
using System;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace ТаблTxt
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
         this.textBox1.Multiline = true;
         this.textBox1.Size = new System.Drawing.Size(320, 216);
         this.Text = "Формирование таблицы";
         String[] Imena = {"Андрей - раб", "Света-Х", "ЖЭК",
                           "Справка по тел", "Александр Степанович",
                           "Мама - дом", "Карапузова Таня",
                           "Погода сегодня", "Театр Браво"};
         String[] Tel = {"274-88-17","+38(067)7030356",
                         "22-345-72", "009", "223-67-67 доп 32-67",
                         "570-38-76", "201-72-23-прямой моб",
                         "001", "216-40-22"};
         textBox1.ScrollBars = ScrollBars.Vertical;
         textBox1.Font = new System.Drawing.Font("Courier New", 9.0F);
         textBox1.Text = "ТАБЛИЦА ТЕЛЕФОНОВ\r\n\r\n";
         for (int i = 0; i <= 8; i++)
            textBox1.Text += String.Format(
                        "{0, -21} {1, -21}", Imena[i], Tel[i]) + "\r\n";
         textBox1.Text += "\r\nПРИМЕЧАНИЕ:" +
                        "\r\nдля корректного отображения таблицы" +
                        "\r\nв Блокноте укажите шрифт Courier New";
         // Запись таблицы в текстовый файл C:\Table.txt.
         // Создание экземпляра StreamWriter для записи в файл
         var Писатель = new System.IO.StreamWriter(@"C:\Table_CS.txt", false,
                              System.Text.Encoding.GetEncoding(1251));
         // - здесь заказ кодовой страницы Win1251 для русских букв
         Писатель.Write(textBox1.Text);
         Писатель.Close();
      }

      private void показатьТаблицуВБлокнотеToolStripMenuItem_Click(
                                    object sender, EventArgs e)
```

```
      {
         try
         {
            System.Diagnostics.Process.Start("Notepad", @"C:\Table.txt");
         }
         catch (Exception Ситуация)
         {  // Отчет об ошибках
            MessageBox.Show(Ситуация.Message, "Ошибка",
            MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
         }
      }

      private void выходToolStripMenuItem_Click(object sender, EventArgs e)
      {  // Выход из программы:
         this.Close();
      }
   }
}
```

Чтобы в текстовом поле **TextBox** таблица отображалась корректно, мы заказали шрифт Courier New. Особенность этого шрифта заключается в том, что каждый символ (буква, точка, запятая и др.) этого шрифта имеет одну и ту же ширину, как это было на печатающей машинке. Поэтому, пользуясь шрифтом Courier New, удобно строить таблицы. Таким же замечательным свойством обладает, например, шрифт Consolas.

Далее в пошаговом цикле `for` мы использовали оператор `+=`, он означает: сцепить текущее содержание текстовой переменной `textBox1.Text` с текстом, представленным справа. Функция `String.Format` возвращает строку, сформированную по формату. Формат заключен в кавычки. Ноль в первых фигурных скобках означает: вставить вместо нуля переменную `Imena[i]`, а единица во вторых фигурных скобках — вставить вместо единицы строку `Tel[i]`. Число 21 означает, что длина строки в любом случае будет состоять из 21 символа (недостающими символами будут пробелы), причем знак "минус" заставляет прижимать текст влево. Символы `"\r\n"` означают, что следует начать текст с новой строки. Внешний вид таблицы в текстовом поле формы показан на рис. 7.1.

После формирования всех строк `textBox1.Text` записываем их в текстовый файл C:\Table.txt через `StreamWriter` с кодовой таблицей Windows 1251. Подробное обсуждение этого фрагмента программы читатель может посмотреть в примере 24 *(см. главу 4)*.

При выборе пользователем пункта меню **Показать таблицу в Блокноте** система создает событие, которое обрабатывается в соответствующей процедуре. Здесь вызываем программу операционной системы Блокнот (notepad.exe) для открытия файла C:\Table.txt (см. рис. 7.2).

Убедиться в работоспособности программы можно, открыв решение ТаблТхt.sln в папке ТаблТхt.

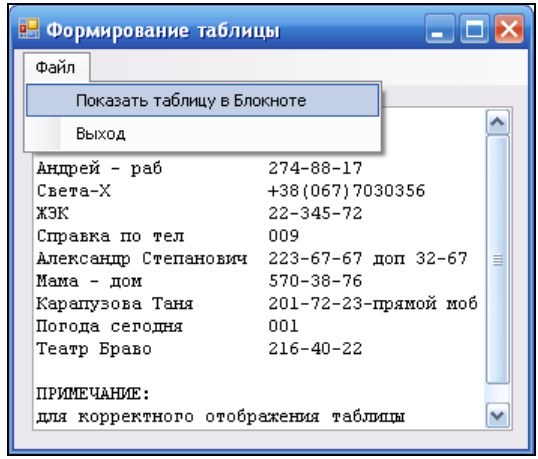


Рис. 7.1. Таблица из двух массивов в текстовом поле

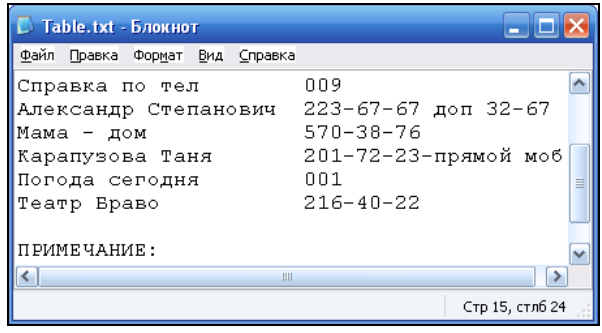


Рис. 7.2. Просмотр созданной таблицы в Блокноте

Пример 45. Форматирование *Double*-переменных в виде таблицы. Вывод таблицы на печать. Поток *StringReader*

Данная программа решает похожую задачу, однако в результате вычислений имеем не строковые переменные `String`, а два массива переменных с двойной точностью `Double`. Например, пусть в результате расчетов получены координаты точек на местности *X* и *Y*. Эти координаты необходимо оформить в виде таблицы. Табли-

цу следует назвать "Каталог координат". Координаты в таблице должны быть округлены до 2-х знаков после запятой. Сформированную таблицу следует показать пользователю в текстовом поле **TextBox**. Далее надо организовать возможность распечатать таблицу на принтере. Заметьте, что в этой задаче мы решаем проблемы без использования Блокнота.

Для решения этой задачи, также как и в предыдущем разделе, в форме с помощью панели элементов управления **Toolbox** создадим текстовое поле **TextBox**, выберем элементы управления **MenuStip** и **PrintDocument**. На вкладке **Design** подготовим пункты меню **Печать** и **Выход** так, как показано на рис. 7.3. Непосредственно текст программы представлен в листинге 7.2.

Листинг 7.2. Формирование таблицы и вывод ее на печать

```
// Программа формирует таблицу на основании двух массивов переменных
// с двойной точностью. Данную таблицу программа демонстрирует пользователю
// в текстовом поле TextBox. Есть возможность распечатать таблицу на принтере
using System;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace TablTxtPrint
{
   public partial class Form1 : Form
   {
      System.IO.StringReader Читатель;
      public Form1()
      {
         InitializeComponent();
         this.Text = "Формирование таблицы";
         Double[] X = {5342736.17653, 2345.3333, 234683.853749,
                       2438454.825368, 3425.72564, 5243.25,
                       537407.6236, 6354328.9876, 5342.243};
         Double[] Y = {27488.17, 3806703.356, 22345.72,
                       54285.34, 2236767.3267, 57038.76,
                       201722.3, 26434.001, 2164.022};
         textBox1.Multiline = true;
         textBox1.ScrollBars = ScrollBars.Vertical;
         textBox1.Font = new System.Drawing.Font("Courier New", 9.0F);
         textBox1.Text = "КАТАЛОГ КООРДИНАТ\r\n";
         textBox1.Text += "---------------------------------\r\n";
         textBox1.Text += "|Пункт|     X     |     Y     |\r\n";
         textBox1.Text += "---------------------------------\r\n";
```

```
            for (int i = 0; i <= 8; i++)
               textBox1.Text += String.Format(
                 "| {0,3:D} | {1,10:F2} | {2,10:F2} |", i, X[i], Y[i]) + "\r\n";
            textBox1.Text += "---------------------------------\r\n";
         }
         private void печатьToolStripMenuItem_Click(object sender, EventArgs e)
         { // Пункт меню "Печать"
            try
            {  // Создание потока Читатель для чтения из строки:
               Читатель = new System.IO.StringReader(textBox1.Text);
               try
                  { printDocument1.Print(); }
               finally
                  { Читатель.Close(); }
            }
            catch (Exception ex)
               { MessageBox.Show(ex.Message); }
         }
         private void выходToolStripMenuItem_Click(object sender, EventArgs e)
         {  // Выход из программы
            this.Close();
         }
      }
}
```

Как видно, формирование таблицы также происходит в цикле `for` с помощью функции `String.Format`. В фигурных скобках числа 0, 1 и 2 означают, что вместо фигурных скобок следует вставлять переменные `i`, `X[i]`, `Y[i]`. Выражение `"3:D"` означает, что переменную `i` следует размещать в трех символах по формату целых переменных `"D"`. Выражение `"10:F2"` означает, что переменную `X(i)` следует размещать в десяти символах по фиксированному формату с двумя знаками после запятой.

При обработке события "щелчок на пункте меню" **Печать** (см. рис. 7.3) в блоках `try...finaly...catch` создаем поток `Читатель`, однако не для чтения из файла, а для чтения из текстовой переменной `textBox1.Text`. В этом случае мы обращаемся с потоком `Читатель` так же, как при операциях с файлами, но совершенно не обращаясь к внешней памяти (диску). Поэтому организация многостраничной печати остается абсолютно такой же, как в примере 29 *(см. главу 4)*.

Как видно из приведенной программы, для того чтобы просмотреть, откорректировать и распечатать на принтере таблицу (инженерных или экономических вычислений), совершенно необязательно записывать эту таблицу в текстовый файл и читать его Блокнотом.

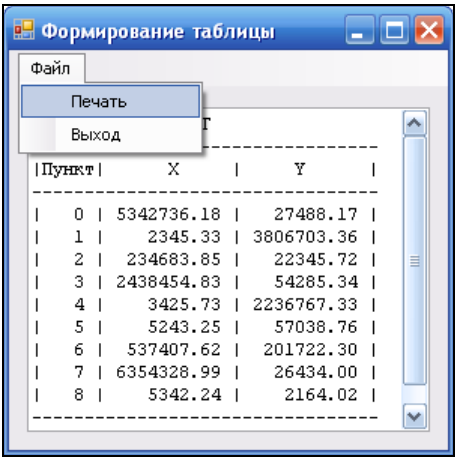


Рис. 7.3. Вывод таблицы в текстовое поле

Убедиться в работоспособности программы можно, открыв соответствующее решение в папке ТаблTxtPrint.

Пример 46. Вывод таблицы в Internet Explorer

Приведем несколько необычный подход к выводу таблицы для целей ее просмотра и распечатывания на принтере. Запишем таблицу в текстовый файл в формате HTML, затем прочитаем ее с помощью обозревателя (браузера) Web-страниц Internet Explorer. HTML (HyperText Markup Language, язык гипертекстовой разметки) — специальные инструкции браузеру, с помощью которых создаются Web-страницы. То есть Web-страницы — это документы в формате HTML, содержащие текст и специальные теги (дескрипторы) HTML. По большому счету теги HTML необходимы для форматирования текста (т. е. придания ему нужного вида), который "понимает" браузер. Документы HTML хранятся в виде файлов с расширением htm или html. Теги HTML сообщают браузеру информацию о структуре и особенностях форматирования Web-страницы. Каждый тег содержит определенную инструкцию и заключается в угловые скобки (<>).

Приведем пример простейшей таблицы, записанный на языке HTML (листинг 7.3).

Листинг 7.3. Представление таблицы на языке HTML

```
<title>Пример таблицы</title>
<table border>
<caption>Таблица телефонов</caption>
```

```
<tr><td>Андрей — раб<td>274-88-17
<tr><td>Света-Х<td>+38(067)7030356
<tr><td>ЖЭК<td>22-345-72
<tr><td>Справка по тел<td>009
</table>
```

Вообще говоря, если строго придерживаться правил языка HTML, то сначала следует написать теги <HTML>, <HEAD>, <TITLE> и т. д. Однако современные браузеры понимают и такую разметку, которая приведена в листинге 7.3. В нашем примере даже не указан ни один закрывающий тег для <tr> (тег, задающий строку в таблице) и для <td> (тег, задающий ячейку в таблице). Очень полезно будет читателю набрать приведенный нами пример в Блокноте, как-нибудь назвать этот файл с расширением htm и открыть его в каком-либо браузере Internet Explorer, Mozilla Firefox или другом обозревателе. Также поучительно открыть этот файл в редакторе HTML Microsoft Office FrontPage и проследить, как меняется HTML-форматирование при тех или иных изменениях, сделанных в таблице.

Наша программа имеет данные, уже знакомые читателю из примера 44. Эти данные находятся в двух массивах: Imena и Tel. На основании этих двух массивов программа формирует таблицу в формате HTML, т. е. создает текстовый файл (в нашей программе он называется C:\Tabl_tel.htm), а затем открывает этот файл браузером Internet Explorer. Текст программы приведен в листинге 7.4.

Листинг 7.4. Вывод таблицы в Internet Explorer

```
// Вывод таблицы в Internet Explorer. Здесь реализован несколько необычный
// подход к выводу таблицы для ее просмотра и печати на принтере.
// Программа записывает таблицу в текстовый файл в формате HTML.
// Теперь у пользователя появляется возможность прочитать эту таблицу
// с помощью обозревателя Web-страниц Internet Explorer или другого браузера
using System;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace Табл_HTM
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
         this.Text = "Таблица в формате HTML";

         string[] Imena = {"Андрей - раб", "Света-Х", "ЖЭК",
                   "Справка по тел", "Александр Степанович", "Мама - дом",
                   "Карапузова Таня", "Погода сегодня", "Театр Браво"};
```

```
        string[] Tel = {"274-88-17", "+38(067)7030356",
                  "22-345-72", "009", "223-67-67 доп 32-67", "570-38-76",
                  "201-72-23-прямой моб", "001", "216-40-22"};
        string text = "<title>Пример таблицы</title>" +
              "<table border><caption>" + "Таблица телефонов</caption>\r\n";
        for (int i = 0; i <= 8; i++)
        text += string.Format("<tr><td>{0}<td>{1}", Imena[i],
                              Tel[i]) + "\r\n";
        text += "</table>";
        // Запись таблицы в текстовый файл C:\Tabl_tel.htm.
        // Создание экземпляра StreamWriter для записи в файл
        var Писатель = new System.IO.StreamWriter(@"C:\Tabl_tel.htm", false,
                               System.Text.Encoding.GetEncoding(1251));
        // - здесь заказ кодовой страницы Win1251 для русских букв
        Писатель.Write(text); Писатель.Close();
        try
        {
           // System.Diagnostics.Process.Start("Iexplore",
                                                @"C:\Tabl_tel.htm");
           // Файл HTM можно открывать также с пом MS_WORD:
           System.Diagnostics.Process.Start("WinWord", @"C:\Tabl_tel.htm");
        }
        catch (Exception Ситуация)
        {  // Отчет об ошибках
           MessageBox.Show(Ситуация.Message, "Ошибка",
                MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
        }
      }
   }
}
```

Как видно, HTML-файл формируется с помощью строковой переменной `text`. Между тегами `<title>` и `</title>` указано название страницы. Это текст, который браузер покажет в заголовке окна.

Тег `<table>` указывает на начало таблицы, между тегами `<caption>` и `</caption>` расположено название таблицы. Далее в пошаговом цикле `for` формируется каждая строка таблицы. Используется уже известный читателю `String.Format`, т. е. вместо фигурной скобки `{0}` подставляется элемент массива `Imena[i]`, а вместо `{1}` — `Tel[i]`.

Затем, создавая поток `Писатель`, сохраняем на диск текстовый файл C:\Tabl_tel.htm. Далее в блоке `try...catch` открываем этот файл с помощью браузера Internet Explorer.

Результат работы программы приведен на рис. 7.4.

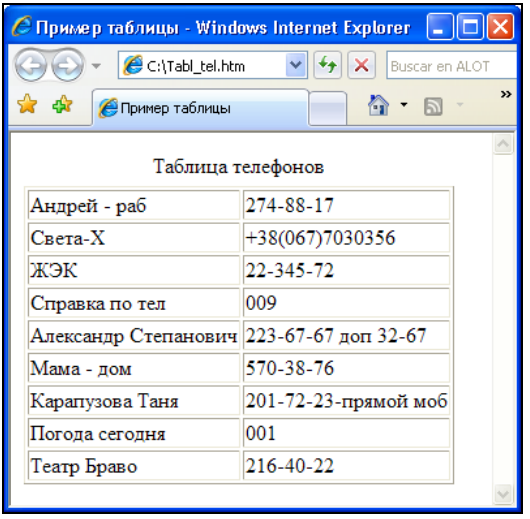


Рис. 7.4. Вывод таблицы в браузер

Очень технологично открыть созданный HTM-файл не Web-браузером, а текстовым редактором MS Word, т. е. в программном коде написать:

```
System.Diagnostics.Process.Start("WinWord", @"C:\Tabl_tel.htm")
```

В этом случае пользователь вашей программы будет иметь возможность редактировать полученную таблицу. Убедиться в работоспособности программы можно, открыв решение Табл_HTM.sln в папке Табл_HTM.

Пример 47. Формирование таблицы с помощью элемента управления *DataGridView*

Создадим приложение, которое заполняет два строковых массива и выводит эти массивы на экран в виде таблицы, используя элемент управления **DataGridView** (просмотр *сетки данных*). Элемент управления **DataGridView** предназначен для просмотра таблиц с возможностью их редактирования.

Запускаем Visual Studio 2010, далее — новый проект и выбираем шаблон **Windows Forms Application C#**, при этом получаем стандартную форму. Перенесем в форму элемент управления **DataGridView** (Сетка данных) из панели **Toolbox**. В данной программе два уже знакомых читателю массива `Imena[]` и `Tel[]` выводятся на сетку данных **DataGridView**. Для максимального упрощения программы формируем таблицу сразу после вызова процедуры `InitializeComponent`. Текст программы приведен в листинге 7.5.

Листинг 7.5. Формирование таблицы с помощью элемента управления *DataGridView*

```
// Программа заполняет два строковых массива и выводит эти массивы на экран
// в виде таблицы, используя элемент управления DataGridView (Сетка данных).
// Элемент управления DataGridView предназначен для просмотра таблиц
// с возможностью их редактирования
using System;
using System.Data;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace ТаблGrid
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
         base.Text = "Формирование таблицы";
         String[] Imena = {"Андрей - раб", "Света-Х", "ЖЭК",
                "Справка по тел", "Александр Степанович", "Мама - дом",
                "Карапузова Таня", "Погода сегодня", "Театр Браво"};
         String[] Tel = {"274-88-17", "+38(067)7030356",
                "22-345-72", "009", "223-67-67 доп 32-67", "570-38-76",
                "201-72-23-прямой моб", "001", "216-40-22"};
         // Создание объекта "таблица данных"
         DataTable Таблица = new DataTable();
         // Заполнение "шапки" таблицы
         Таблица.Columns.Add("Имена");
         Таблица.Columns.Add("Номера телефонов");
         // Заполнение клеток (ячеек) таблицы данных
         for (int i = 0; i <= 8; i++)
            Таблица.Rows.Add(new String[] { Imena[i], Tel[i] });
         // Для сетки данных указываем источник данных
         dataGridView1.DataSource = Таблица;
      }
   }
}
```

Как видно, в программе используется объект таблица данных `DataTable`. С его помощью сначала заполняем "шапку" таблицы данных, используя метод `Columns.Add`, а затем непосредственно ячейки таблицы, используя метод `Rows.Add`.

Чтобы передать построенную таблицу в элемент управления **DataGridView**, указываем в качестве источника данных `DataSource` объект `Таблица` класса `DataTable`.

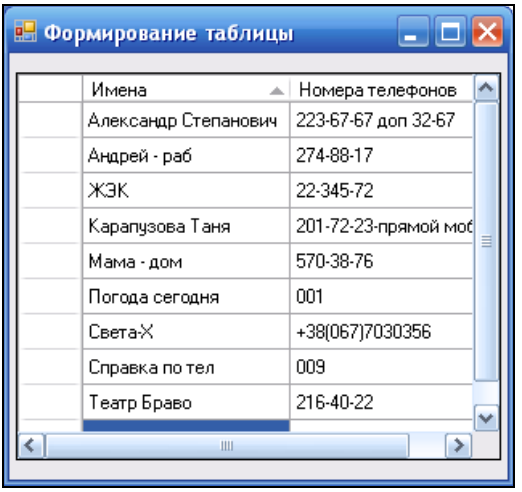


Рис. 7.5. Форматирование таблицы

На рис. 7.5 приведен результат работы программы. Заметьте, щелкая мышью на заголовках колонок, получаем сортировку данных в алфавитном порядке полей таблицы.

Убедиться в работоспособности программы можно, открыв решение ТаблGrid.sln в папке ТаблGrid.

Пример 48. Табличный ввод данных. *DataGridView*. *DataTable*. *DataSet*. Инструмент для создания файла XML

Существует множество задач, предполагающих ввод данных в виде таблиц. Конечно, можно эту таблицу программировать как совокупность текстовых полей **TextBox**, но часто заранее неизвестно, сколько рядов данных будет вводить пользователь, необходимо предусмотреть скроллинг этой таблицы и т. д. То есть проблем в организации ввода табличных данных достаточно много.

Мы предлагаем для цели ввода табличных данных использовать элемент управления **DataGridView** (Сетка данных). Прежде всего, этот элемент управления предназначен для отображения данных, которые удобно представить в виде таблицы, чаще всего источником этих данных является база данных. Однако кроме отображения **DataGridView** позволяет также редактировать табличные данные. Эле-

мент управления **DataGridView** поддерживает выделение, изменение, удаление, разбиение на страницы и сортировку.

Программа, обсуждаемая в данном разделе, предлагает пользователю заполнить таблицу телефонов его знакомых, сотрудников, родственников, любимых и т. д. После щелчка на кнопке **Запись** данная таблица записывается на диск в файл в формате XML. Для упрощения текста программы предусмотрена запись в один и тот же файл C:\tabl.xml. При последующих запусках данной программы таблица будет считываться из файла, и пользователь может продолжать редактирование таблицы. Поэтому эту программу можно громко назвать табличным редактором. Щелкая мышью на заголовках колонок, можно расположить записи в колонках в алфавитном порядке для удобного поиска необходимого телефона.

Для написания программы требуется из панели управления **Toolbox** перенести мышью элементы управления: сетка данных **DataGridView** и кнопка **Button**. Текст программы приведен в листинге 7.6.

Листинг 7.6. Заполнение телефонной книги

```
// Программа предлагает пользователю заполнить таблицу телефонов его знакомых,
// сотрудников, родственников, любимых и т. д. После щелчка на кнопке Запись
// данная таблица записывается на диск в файл в формате XML. Для упрощения
// текста программы предусмотрена запись в один и тот же файл C:\tabl.xml.
// При последующих запусках данной программы таблица будет считываться
// из этого файла, и пользователь может продолжать редактирование таблицы
using System;
using System.Data;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace ТаблВвод
{
   public partial class Form1 : Form
   {
      DataTable Таблица = new DataTable(); // Создание объекта "таблица данных"
      DataSet НаборДанных = new DataSet(); // Создание объекта "набор данных"
      public Form1()
      {
         InitializeComponent();
         base.Text = "Почти табличный редактор";
         button1.Text = "Запись";
         if (System.IO.File.Exists(@"C:\tabl.xml") == false)
         {  // Если XML-файла НЕТ:
            dataGridView1.DataSource = Таблица;
```

```
            // Заполнение "шапки" таблицы
            Таблица.Columns.Add("Имена");
            Таблица.Columns.Add("Номера телефонов");
            // Добавить объект Таблица в DataSet
            НаборДанных.Tables.Add(Таблица);
         }
         else // Если XML-файл ЕСТЬ:
         {
            НаборДанных.ReadXml(@"C:\tabl.xml");
            // Содержимое DataSet в виде строки XML для отладки:
            string СтрокаXML = НаборДанных.GetXml();
            dataGridView1.DataMember = "Название таблицы";
            dataGridView1.DataSource = НаборДанных;
         }
      }
      private void button1_Click(object sender, EventArgs e)
      {  // Сохранить файл tabl.xml:
         Таблица.TableName = "Название таблицы";
         НаборДанных.WriteXml(@"C:\tabl.xml");
      }
   }
}
```

Как видно из текста программы, потребовалось всего лишь несколько строк программного кода для создания такой многофункциональной программы. Это стало возможным благодаря использованию мощной современной технологии ADO.NET. В начале класса объявлены два объекта этой технологии: набор данных `DataSet` и таблица данных `DataTable`. Объект класса `DataSet` является основным компонентом архитектуры ADO.NET. `DataSet` представляет кэш данных, расположенный в оперативной памяти. `DataSet` состоит из коллекции объектов класса `DataTable`. То есть в один объект класса `DataSet` может входить несколько таблиц, а информацию о них мы можем записывать в файл на диск одним оператором `WriteXml`, соответственно читать — `ReadXML`. Таким образом, в этой программе мы имеем дело преимущественно с тремя объектами: `DataSet` — кэш данных, `DataTable` — представляет одну таблицу с данными и `DataGridView` — элемент управления для отображения данных.

Сразу после инициализации компонентов формы мы обработали две ситуации. Если файла, в который мы сохраняем информацию о таблице, не существует `Exists("C:\\tabl.xml") == false`, то назначаем в качестве источника данных `DataSource` для `DataGrid` объект класса `DataTable` и заполняем "шапку" таблицы, т. е. указываем названия колонок: "Имена" и "Номера телефонов", а затем добавляем объект `DataTable` в объект `DataSet`. Теперь пользователь видит пустую таблицу с двумя колонками и может ее заполнять. Если файл существует (ветвь `else`), то

данные в объект `DataSet` отправляем из XML-файла (`ReadXML`). Здесь уже в качестве источника данных для сетки данных `DataGrid` указываем объект `DataSet`.

При щелчке мышью на кнопке **Запись** (рис. 7.6) — событие `button1.Click` — происходит запись XML-файла на диск (`WriteXml`).

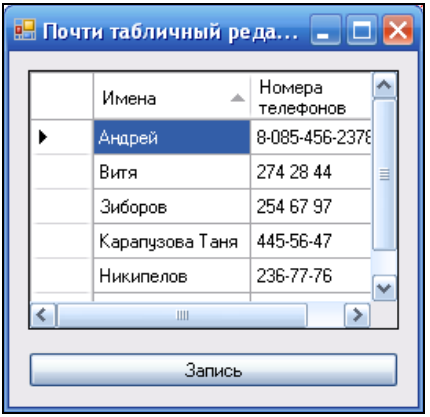


Рис. 7.6. Почти табличный редактор

Здесь используются так называемые XML-файлы. Формат этих файлов позволяет легко и надежно передавать данные посредством Интернета даже на компьютеры другой платформы (например, Macintosh). В нашей задаче мы не ставили перед собой цель работать в Интернете, а всего лишь воспользовались данной технологией. Файл формата XML можно просмотреть Блокнотом или с помощью MS Word, поскольку это текстовый файл. Однако следует учесть, что этот файл записан в кодировке UTF-8, поэтому другими текстовыми редакторами, например edit.com или Rpad32.exe (русский Блокнот), его прочитать затруднительно. XML-документ открывается Web-браузером, XML-editor (входит в состав Visual Studio 2010), MS Office SharePoint Designer, MS Front Page и другими программами. При этом отступы и разные цвета позволяют выразительно увидеть структуру данного файла. XML-файл можно открыть табличным редактором MS Excel, и при этом он может отобразиться в виде таблицы. На рис. 7.7 приведен образец представления XML-файла в MS Word.

При использовании нами XML-файлов для данной задачи программирования простейшего табличного редактора совсем необязательно вникать в его структуру, тем более что при выборе имени файла для сохранения совершенно необязательно устанавливать расширение файла xml, файл с любым расширением будет читаться методом `ReadXml`, как XML-файл.

Программист может иметь доступ к полям таблицы. Например, доступ к левой верхней ячейке (полю) таблицы можно получить, используя свойство объекта класса `DataTable`: `Таблица.Rows.Item(0).Item(0)`. Однако запись этого поля, например, в последовательный файл будет некорректной даже при использовании дополнительной переменной из-за того, что технология ADO.NET предусматривает

кэширование данных. Таким образом, чтение и запись данных для подобных таблиц следует организовывать только через методы объекта `DataSet`.

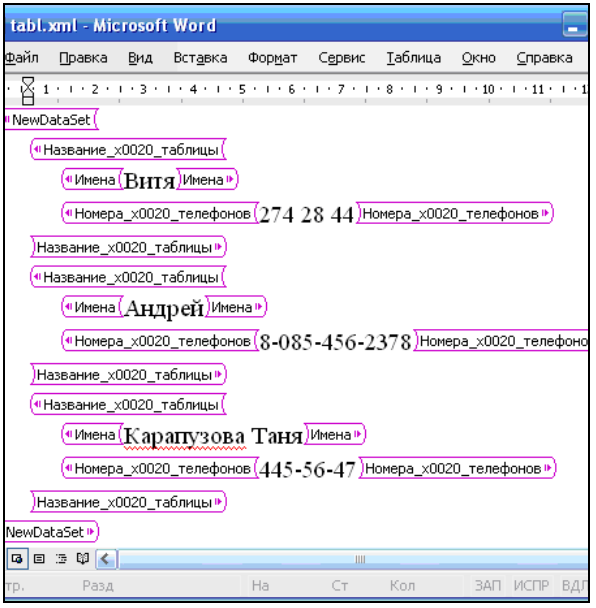


Рис. 7.7. Образец представления XML-файла в MS Word

Замечу, что данная программа может также являться инструментом для создания XML-файлов. Убедиться в работоспособности программы можно, открыв решение ТаблВвод.sln в папке ТаблВвод.

Пример 49. Решение системы линейных уравнений. Ввод коэффициентов через *DataGridView*

В инженерном деле, в экономических, научных расчетах часто встречается задача по решению системы линейных алгебраических уравнений (СЛАУ). Между тем, решение данной задачи последнее время редко приводится в книгах по языкам программирования. Данная программа приглашает пользователя ввести в текстовое поле количество неизвестных (т. е. размерность системы). Если пользователь справился с этим заданием, то это текстовое поле становится недоступным и появляется элемент управления сетка данных **DataGridView**, куда пользователь имеет возможность ввести коэффициенты линейных уравнений и свободные члены. При щелчке на кнопке **Решить** программа проверяет корректность введенных данных (не должно быть нечисловых символов, количество строк в матрице коэффициен-

тов должно быть равно количеству неизвестных). Далее происходит непосредственное решение введенной системы методом Гаусса и вывод результатов вычислений с помощью `MessageBox`.

Таким образом, пользовательский интерфейс будет состоять (см. рис. 7.8) из формы, метки **Label**, текстового поля **TextBox**, элемента управления **DataGridView** и кнопки **Button**. В листинге 7.7 приведен программный код решения задачи.

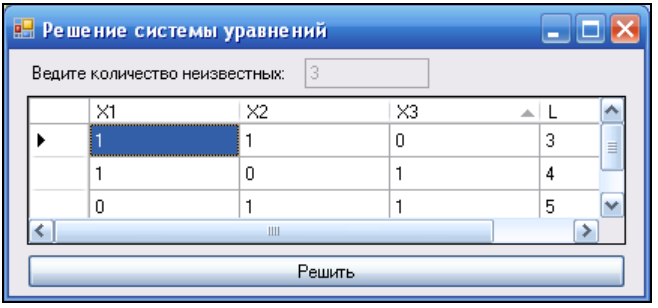


Рис. 7.8. Фрагмент табличного ввода данных

Листинг 7.7. Решение системы линейных уравнений

```
// Программа для решения системы линейных уравнений. Ввод коэффициентов
// предусмотрен через DataGridView
using System;
using System.Data;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace ГауссGrid
{
   public partial class Form1 : Form
   {
      int n; // - размерность СЛАУ
      DataTable dt = new DataTable();
      public Form1()
      {
         InitializeComponent();
         base.Text = "Решение системы уравнений";
         // Чтобы при старте программы фокус находился в текстовом поле:
         textBox1.TabIndex = 0;
         dataGridView1.Visible = false; // сетку данных пока не видно
         label1.Text = "Ведите количество неизвестных:";
```

```
         button1.Text = "Ввести";       // первоначальная надпись на кнопке
      }
      private void button1_Click(object sender, EventArgs e)
      {
         double[,] A = new double[n, n]; // - матрица коэффициентов
         double[] L = new double[n];     // - вектор свободных членов
         int i, j; bool Число_ли = false;
         string tmp;   // - временная рабочая переменная
         if (button1.Text == "Ввести")
         {
            for (; ; )
            {  // Бесконечный цикл, пока пользователь не введет именно число:
               Число_ли = int.TryParse(textBox1.Text,
                     System.Globalization.NumberStyles.Integer,
                     System.Globalization.NumberFormatInfo.CurrentInfo, out
n);
               if (Число_ли == false) return;
               button1.Text = "Решить";
               textBox1.Enabled = false;  // теперь текстовое поле недоступно
               dataGridView1.Visible = true; // сетку данных теперь уже видно
               dataGridView1.DataSource = dt;
               // Создаем "шапку" таблицы
               for (i = 1; i <= n; i++)
               {
                  tmp = "X" + Convert.ToString(i);
                  dt.Columns.Add(new DataColumn(tmp));
               }
               // Колонка правой части системы:
               dt.Columns.Add(new DataColumn("L"));
               return;
            }
         }
         else // button1.Text == "Решить")
        {   // Нажали кнопку "Решить"
            // dt.Rows.Count - количество рядов
            if (dt.Rows.Count != n)
            {
               MessageBox.Show("Количество строк не равно количеству колонок");
               return;
            }
            // Заполнение матрицы коэффициентов системы A[j, i]
            for (j = 0; j <= n - 1; j++)
```

```
      {
         for (i = 0; i <= n - 1; i++)
         {
            A[j, i] = ВернутьЧисло(j, i, ref Число_ли);
            if (Число_ли == false) return;
         }
         // Правая часть системы B(j, 0)
         L[j] = ВернутьЧисло(j, i, ref Число_ли);
         if (Число_ли == false) return;
      } // j
   }
   gauss(n, A, ref L); // Решение системы A*x = L методом Гаусса
   // L - вектор свободных членов системы, сюда же возвращается
   // решение x
   string s = "Неизвестные равны:\n";
   for (j = 1; j <= n; j++)
   {
      tmp = L[j - 1].ToString();
      s = s + "X" + j.ToString() + " = " + tmp + ";\n";
   }
   MessageBox.Show(s);
}
private void gauss(int n, double[,] A, ref double[] LL)
{  // n  - размер матрицы
   // A  - матрица коэффициентов линейных уравнений
   // LL - правая часть, сюда возвращаются значения неизвестных
   int i, j, l = 0;
   Double c1, c2, c3;
   for (i = 0; i <= n - 1; i++) // цикл по элементам строки
   {
      c1 = 0;
      for (j = i; j <= n - 1; j++)
      {
         c2 = A[j, i];
         if (Math.Abs(c2) > Math.Abs(c1))
         {
            l = j; c1 = c2;
         }
      }

      for (j = i; j <= n - 1; j++)
      {
```

```
            c3 = A[l, j] / c1;
            A[l, j] = A[i, j]; A[i, j] = c3;
         } // j

         c3 = LL[l] / c1; LL[l] = LL[i]; LL[i] = c3;

         for (j = 0; j <= n - 1; j++)
         {
            if (j == i) continue;
            for (l = i + 1; l <= n - 1; l++)
            {
               A[j, l] = A[j, l] - A[i, l] * A[j, i];
            } // l
            LL[j] = LL[j] - LL[i] * A[j, i];
         } // j
      } // i
   }
   private double ВернутьЧисло(int j, int i, ref bool Число_ли)
   {  // j - номер строки, i - номер столбца
      double rab; // - рабочая переменная
      string tmp = dt.Rows[j][i].ToString();
      Число_ли = Double.TryParse(tmp,
          System.Globalization.NumberStyles.Number,
          System.Globalization.NumberFormatInfo.CurrentInfo, out rab);
      if (Число_ли == false)
      {
         tmp = string.Format("Номер строки {0}, номер столбца " +
                  "{1},\n в данном поле - не число", j + 1, i + 1);
         MessageBox.Show(tmp);
      }
      return rab;
   }
}
}
```

Как видно их программного кода, при начальной загрузке программы пользователь не видит (`Visible = false`) сетку данных, а первоначальная надпись на кнопке — "Ввести". При щелчке на кнопке, если пользователь корректно ввел количество неизвестных, меняем надпись на кнопке (она теперь будет — "Решить"), по количеству неизвестных подготавливаем "шапку" таблицы и размерность сетки данных, куда пользователь будет вводить коэффициенты линейных уравнений и свободные члены.

После ввода коэффициентов и щелчка на кнопке **Решить** происходит проверка количества введенных рядов коэффициентов и проверка на нечисловые символы. После этого вызывается *процедура решения СЛАУ* методом Гаусса `gauss`, т. е. методом *последовательного исключения* неизвестных. В основе процедуры `gauss` — цикл `for` по элементам строки матрицы системы. В этот внешний цикл вложены три внутренних цикла по строкам матрицы. После вызова процедуры `gauss` формируется строковая переменная `s` для вывода значений неизвестных, которая выводится посредством диалогового окна `MessageBox` (рис. 7.9).

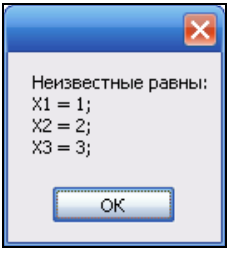


Рис. 7.9. Вывод неизвестных значений в диалоговое окно

Данная программа не предусматривает проверку на вырожденность СЛАУ, однако в этом случае в качестве значений неизвестных пользователь получает либо "бесконечность", либо константу `NaN`, значение которой является результатом деления на ноль. Программа не предусматривает ограничение сверху на размерность решаемой системы и должна работать при любой разумной размерности. Работа программы должна быть ограничена лишь размером оперативной памяти. Однако автор не тестировал эту программу на решение систем большой размерности. Если количество неизвестных равно одному, то программа также нормально функционирует.

Убедиться в работоспособности программы можно, открыв решение GauccGrid.sln в папке GauccGrid.

Глава 8

Элемент управления *WebBrowser*

Пример 50. Отображение HTML-таблиц

В этом примере воспользуемся элементом управления **WebBrowser** для отображения таблицы, записанной на языке HTML с помощью элементарных тегов: `<tr>`, задающего строку в таблице, и `<td>`, задающего ячейку в таблице. Понятно, что, сформировав такой HTML-файл, содержащий таблицу, подлежащую выводу на экран, можно вывести этот файл в поле элемента управления **WebBrowser** с помощью метода `Navigate` объекта.

Покажем, как, задав на языке HTML какую-нибудь таблицу в строковой переменной, можно вывести эту таблицу в поле элемента управления **WebBrowser**, при этом не записывая на винчестер HTML-файл. Для этой цели воспользуемся простейшей таблицей из примера о формировании таблицы.

Приступим к программированию поставленной задачи. Запустим Visual Studio 2010 и выберем новый проект. Панель элементов **Toolbox** содержит в себе элемент **WebBrowser**, который перетаскиваем в форму. Текст программы приведен в листинге 8.1.

Листинг 8.1. Отображение таблиц с помощью элемента управления *WebBrowser*

```
// В программе для отображения таблицы используется элемент управления
// WebBrowser. Таблица записана на языке HTML с помощью элементарных
// тегов <tr> (строка в таблице) и <td> (ячейка в таблице)
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace ТаблWebHTM
{
   public partial class Form1 : Form
   {
      public Form1()
      {
```

```
        InitializeComponent();
        // или webBrowser1.Navigate("c:\\table.htm");
        string Str_html = "Какой-либо текст до таблицы" +
                          "<table border> " +
                          "<caption>Таблица телефонов</caption> " +
                          "<tr><td>Андрей — раб<td>274-88-17 " +
                          "<tr><td>Света-Х<td>+38(067)7030356 " +
                          "<tr><td>ЖЭК<td>22-345-72 " +
                          "<tr><td>Справка по тел<td>009 " +
                          "</table> " +
                          "Какой-либо текст после таблицы";
        webBrowser1.Navigate("about:" + Str_html);
      }
   }
}
```

Как видно из текста программы, в строковой переменной `Str_html` задаем HTML-представление простейшей таблицы, состоящей из двух столбцов и четырех строк. Эту строку подаем на вход метода `Navigate` объекта `WebBrowser1`, сцепляя ее с ключевым словом `"about:"`. В комментарии показано, как можно подать на вход метода `Navigate` таблицу, если она уже записана в файл table.htm.

В результате работы этой программы получаем отображение таблицы в поле элемента управления **WebBrowser**, как показано на рис. 8.1.

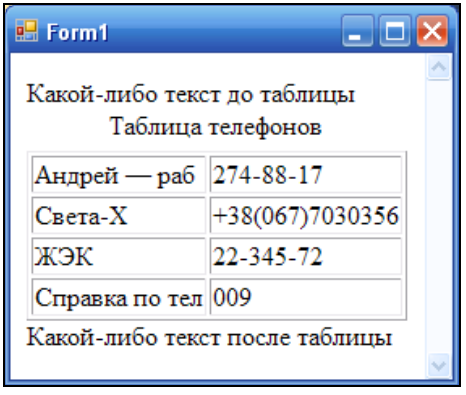


Рис. 8.1. Отображение таблицы в поле элемента управления **WebBrowser**

Теперь очень легко можно распечатать данную таблицу на принтере, для этого необходимо попасть в контекстное меню, щелкая правой кнопкой мыши в пределах таблицы.

Убедиться в работоспособности программы можно, открыв решение ТаблWebHTM.sln в папке ТаблWebHTM.

Пример 51. Отображение Flash-файлов

Ранее мы уже обсуждали элемент управления **PictureBox**, способный отображать растровые файлы различных форматов. Однако **PictureBox** не умеет отображать Flash-файлы формата SWF. Этот формат очень распространен в Интернете благодаря тому, что удачно сочетает в себе векторную графику, растровую графику, воспроизведение звука и при этом предусматривает компактный код, что позволяет быстро загружать SWF-файлы в браузер.

В данном разделе мы создадим программу, которая будет отображать в поле элемента управления **WebBrowser** файл SWF. Запустим Visual Studio 2010, затем выберем новый проект — **Windows Forms Application C#**. Перетащим с панели **Toolbox** элемент управления **WebBrowser** в форму. Теперь, например, через контекстное меню мы попадем на вкладку программного кода, где введем текст из листинга 8.2.

Листинг 8.2. Отображение Flash-файлов

```
// Программа использует элемент управления WebBrowser для отображения
// Flash-файлов
using System.Windows.Forms;
// using System.IO;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace FlashWeb
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
         // webBrowser1.Navigate("www.mail.ru");
         webBrowser1.Navigate(System.IO.Directory.GetCurrentDirectory() +
                              "\\Shar.swf");
      }
   }
}
```

Здесь в тексте программы вызываем метод `Navigate` объекта `webBrowser1`. На вход метода подаем полный путь к Flash-файлу Shar.swf. Полный путь к текущей папке получаем с помощью функции `GetCurrentDirectory()`. Обращение к этой функции можно было немного сократить, импортируя пространство имен с помощью директивы `using System.IO`, как показано в комментарии. Также в комментарии показано, как можно вывести Web-страницу из Интернета в поле элемента управления **WebBrowser**.

В результате работы этой программы получаем отображение файла Shar.swf (рис. 8.2).

К сожалению, из рисунка не видно, что шар вращается. Убедиться в работоспособности программы можно, открыв решение FlashWeb.sln в папке FlashWeb.

Рис. 8.2. Отображение Flash-файла в поле элемента управления **WebBrowser**

Пример 52. Отображение Web-страницы и ее HTML-кода

Напишем на C#-программу, способную в одном поле отображать Web-страницу, а в другом поле, ниже, показывать HTML-код загружаемой в браузер страницы. Ядром этой программы будет все тот же элемент управления обозревателя **WebBrowser**. Таким образом, экранная форма будет содержать на себе элемент управления **WebBrowser**, два текстовых поля и кнопку (рис. 8.3).

Запустим Visual Studio 2010, выполним команду **New | Project** из меню **File**, далее выберем шаблон **Windows Forms Application C#** и нажмем кнопку **OK**. Используя панель элементов, добавим элемент управления **WebBrowser** в экранную форму, также добавим два текстовых поля и кнопку. Первое текстовое поле `textBox1` предназначено для ввода URL-адреса желаемой Web-страницы (например, **www.latino.ho.ua**), а второе поле `textBox2` растянем побольше, поскольку сюда будет выводиться HTML-код загружаемой в браузер страницы. Чтобы была возможность растянуть второе поле по вертикали, укажем явно в свойствах этого поля (**Properties**) `Multiline = True`. Текст программы приведен в листинге 8.3.

Листинг 8.3. Отображение Web-страницы и ее HTML-кода

```
// Эта программа использует элемент управления WebBrowser
// для отображения Web-страницы и ее HTML-кода
using System;
using System.Windows.Forms;
```

```
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace Split
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
         base.Text = "WEB-страница и ее HTML-код";
         textBox1.Text = string.Empty; textBox2.Text = string.Empty;
         textBox2.Multiline = true;
         textBox2.ScrollBars = ScrollBars.Vertical;
         button1.Text = "ПУСК";
      }
      private void button1_Click(object sender, EventArgs e)
      {  // Щелчок на кнопке ПУСК:
         webBrowser1.Navigate(textBox1.Text);
         // webBrowser1.Navigate("www.latino.ho.ua")
         // webBrowser1.GoBack()     // Назад
         // webBrowser1.GoForward()  // Вперед
         // webBrowser1.GoHome()     // На домашнюю страницу
      }
      private void webBrowser1_DocumentCompleted(object sender,
                              WebBrowserDocumentCompletedEventArgs e)
      {  // Событие документ обозревателя "укомплектован":
         //                                "заполнен"
         // Получить HTML-код из документа WebBrowser
         textBox2.Text = webBrowser1.Document.Body.InnerHtml;
      }
   }
}
```

Как видно из программного кода, сразу после вызова процедуры `InitializeComponent` задаем некоторые уже хорошо известные читателю свойства. Свойство `ScrollBars`, определяющее скроллинг текстового поля, приведено в состояние `Vertical`, означающее разрешение вертикальной прокрутки.

При обработке события "щелчок на кнопке" **Пуск** метод `Navigate` объекта `webBrowser1` вызывает в поле элемента управления обозревателя Web-страницу с адресом, указанным пользователем данной программы в текстовом поле `textBox1`. В комментарии приведены варианты дальнейшего совершенствования данной программы. Так, метод `GoBack` возвращает пользователя на предыдущую Web-страницу, метод `GoForward` — на последующую, а метод `GoHome` загружает в поле

браузера домашнюю Web-страницу. Все эти методы можно было бы развести по разным командным кнопкам, но автор не стал этого делать, чтобы не загромождать программу, а наоборот выразительно показать главное.

Событие `DocumentCompleted` подразумевает, что документ обозревателя "укомплектован", т. е. загрузка страницы в браузер завершена. При обработке этого события функция `Body.innerHTML` возвращает в свойство `Text` текстового поля `textBox2` HTML-код, соответствующий загруженной в браузер Web-страницы. На рис. 8.3 показан пример работы данной программы.

Убедиться в работоспособности программы можно, открыв решение Split.sln в папке Split.

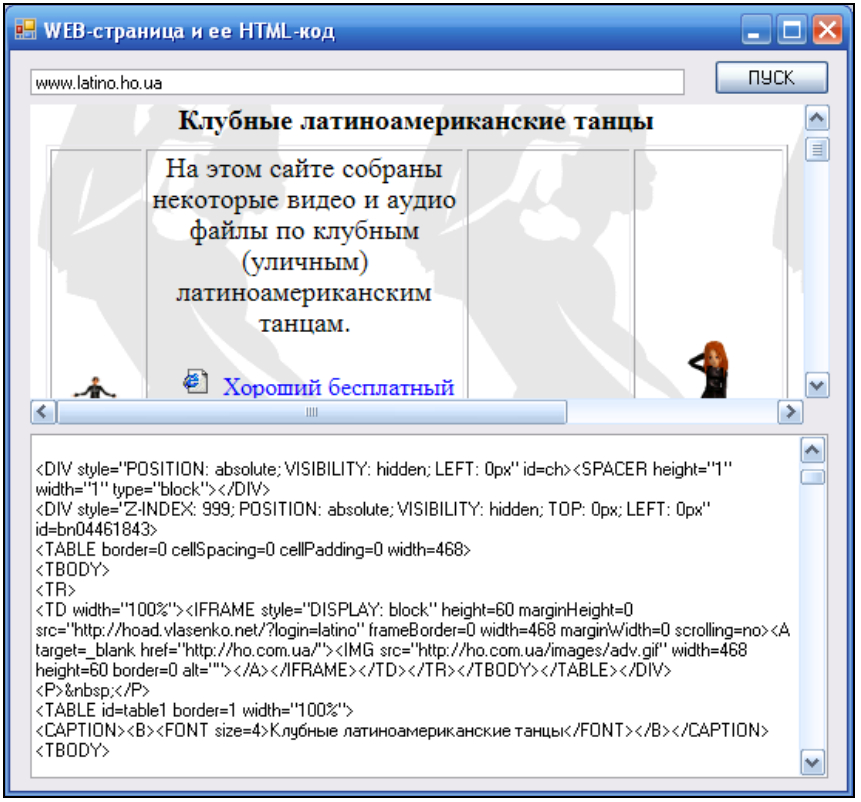


Рис. 8.3. Загрузка Web-страницы и ее HTML-кода

Пример 53. Программное заполнение Web-формы

Задача состоит в том, чтобы, воспользовавшись элементом управления **WebBrowser**, отобразить в этом элементе какую-либо Web-страницу, содержащую

Web-форму. Затем программно заполнить поля формы (например, логин и пароль на почтовом сервере, или строку поиска в поисковой системе, или персональные данные для регистрации в нужных нам ресурсах, при оформлении подписки на рассылку и проч.). А при завершении заполнения программно щелкнуть на кнопке **Submit** (она может называться по-разному: **OK**, **Отправить**, **Send** и т. д.) для отправки заполненной Web-формы для обработки на сервер. В итоге в элементе управления **WebBrowser** будет отображен уже результат, полученный с сервера после обработки. Подобным средством можно воспользоваться для подбора паролей, однако мы не хотели бы популяризировать эту возможность из соображений морали. В данном примере мы будем заполнять строку поиска в наиболее используемых поисковых системах, таких как Yandex, Rambler, Google и др.

Для решения этой задачи запустим Visual Studio 2010, далее выберем новый проект шаблона **Windows Forms Application C#**. Из панели элементов **Toolbox** в экранную форму перенесем элемент управления **WebBrowser**. Чтобы получить удобный доступ к конкретным дескрипторам (тегам) HTML, составляющим Web-страницу, к текущему проекту добавим ссылку на так называемую "неуправляемую" библиотеку Microsoft.mshtml.dll. Не стоит пугаться слова "неуправляемая", так компания Microsoft называет библиотеки, содержащие "неуправляемый" код, который выполняется не общеязыковой средой, а непосредственно операционной системой. Как видите, имя файла библиотеки содержит не совсем привычные две точки. Этот файл, скорее всего, расположен на вашем компьютере в папке: C:\Program Files\Microsoft Visual Studio 10.0\Visual Studio Tools for Office\PIA\Common.

Для добавления ссылки на эту библиотеку в пункте меню **Project** выберем команду **Add Reference**, а на вкладке **.NET** дважды щелкнем на ссылке Microsoft.mshtml. Теперь в окне **Solution Explorer**, раскрывая пункт **References**, увидим добавленную в наш проект искомую ссылку.

Далее перейдем на вкладку программного кода и введем текст, представленный в листинге 8.4.

Листинг 8.4. Программное заполнение формы некоторых поисковых систем

```
// Программа загружает в элемент WebBrowser начальную страницу поисковой
// системы http://yahoo.com. Далее, используя указатель на неуправляемый
// интерфейс DomDocument (свойство объекта класса WebBrowser),
// приводим его к указателю IHTMLDocument2. В этом случае мы получаем
// доступ к формам и полям Web-страницы по их именам. Заполняем поле
// поиска ключевыми словами для нахождения соответствующих Web-страниц,
// а затем для отправки заполненной формы на сервер "программно" нажимаем
// кнопку Submit. В итоге получим в элементе WebBrowser результат работы
// поисковой системы, а именно множество ссылок на страницы, содержащие
// указанные ключевые слова.
using System;
using System.Windows.Forms;
```

```
// Другие директивы using удалены, поскольку они не используются
// в данной программе.

// В данное приложение необходимо импортировать неуправляемую библиотеку
// Microsoft.mshtml.dll, для этого в пункте меню Project выбираем команду
// Add Reference, а на вкладке .NET дважды щелкнем на ссылке Microsoft.mshtml
namespace ЗаполнениеWeb_формы
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
         this.Text = "Программное заполнение формы";
         // *** Для сайта "http://google.com":
         //string АдресСайта = "http://google.com"
         //string ИмяФормы = "f";
         //string ИмяПоляФормы = "q";
         // Для сайта "http://meta.ua":
         //string АдресСайта = "http://meta.ua";
         //string ИмяФормы = "sForm";
         //string ИмяПоляФормы = "q";
         // *** Для сайта "http://yandex.ru":
         //string АдресСайта = "http://yandex.ru";
         //string ИмяФормы = "form";
         //string ИмяПоляФормы = "text";
         // *** Для сайта "http://rambler.ru":
         //string АдресСайта = "http://rambler.ru";
         //string ИмяФормы = "rSearch";
         //string ИмяПоляФормы = "query";
         // *** Для сайта "http://aport.ru":
         //string АдресСайта = "http://aport.ru";
         //string ИмяФормы = "aport_search";
         //string ИмяПоляФормы = "r";
         // *** Для сайта "http://bing.com":
         //string АдресСайта = "http://bing.com";
         //string ИмяФормы = "sb_form";
         // - в HTML-коде нет name формы, но есть id = "sb_form"
         //string ИмяПоляФормы = "q";
         // *** Для сайта "http://yahoo.com":
         string АдресСайта = "http://yahoo.com";
         string ИмяФормы = "sf1";    // или "p_13838465-searchform"
```

```
        string ИмяПоляФормы = "p";  // или "p_13838465-p"
        // Загружаем Web-документ в элемент WebBrowser:
        webBrowser1.Navigate(АдресСайта);
        while (webBrowser1.ReadyState != WebBrowserReadyState.Complete)
        {
           Application.DoEvents();
           System.Threading.Thread.Sleep(50);
        }
        if (webBrowser1.Document == null)
        {
           MessageBox.Show("ERROR: Возможно, вы не подключены к Интернету");
           return;
        }
        // Свойство DomDocument приводим к указателю IHTMLDocument2:
        mshtml.IHTMLDocument2 Док = (mshtml.
                          IHTMLDocument2)webBrowser1.Document.DomDocument;
        // В этом случае мы получаем доступ к формам Web-страницы
        // по их именам:
        mshtml.HTMLFormElement Форма = Док.forms.item(ИмяФормы, null);
        if (Форма == null)
        {
           MessageBox.Show(String.Format("ERROR: Форма с " +
                           "именем \"{0}\" не найдена", ИмяФормы));
           return;
        }
        // В форме находим нужное поле по его (полю) имени:
        mshtml.IHTMLInputElement ТекстовоеПоле =
                     Форма.namedItem(ИмяПоляФормы);
        if (ТекстовоеПоле == null)
        {
           MessageBox.Show(String.Format("ERROR: Поле формы с " +
                           "именем \"{0}\" не найдено ", ИмяПоляФормы));
           return;
        }
        // Заполняем текстовое поле:
        string query = "Зиборов Visual";
        ТекстовоеПоле.value = query;
        // "Программно" нажимаем кнопку "Submit":
        Форма.submit();
     }
  }
}
```

Как видно из программного кода, сразу после завершения процедуры `InitializeComponent` мы подготовили для поисковой системы Yahoo ее Web-адрес, имя формы и имя поля формы, куда пользователь вводит ключевые слова для поиска. Также в комментариях приведены эти параметры и для нескольких других поисковых систем: Google, Meta, Yandex, Rambler, Aport и Bing. Как мы определили эти параметры?

Для этого в браузере, например, Internet Explorer зададим в адресной строке имя поисковой системы Yahoo, затем в контекстном меню выберем команду **Просмотр HTML-кода**. В этом случае мы увидим в Блокноте HTML-разметку начальной страницы системы. Используя контекстный поиск программы Блокнот, в разметке поисковой системы Yahoo найдем тег формы `"<form"`. В этом теге следует найти либо атрибут `name`, либо атрибут `id`. Любой из этих атрибутов можно указать в качестве имени формы в нашей программе. Далее между открывающимся тегом `<form>` и закрывающимся тегом `</form>` ищем тег `<input>`, представляющий поле формы. Этот тег должен иметь тип (`type`) `"text"`, но не `"hidden"` (скрытый). В этом теге также найдем или атрибут `name`, или атрибут `id`. Любой из них можно указать в качестве имени поля формы в нашей программе. Аналогичным образом мы нашли имена форм и полей этих форм и в других Web-страницах поисковых систем, которые представлены в комментариях.

После первичной загрузки Web-документа в элемент управления **WebBrowser**, используя указатель на неуправляемый интерфейс `DomDocument` (свойство объекта класса `WebBrowser`), приводим его к указателю `IHTMLDocument2`. В этом случае мы получаем доступ к формам и полям Web-страницы по их именам. Далее заполняем поле поиска ключевыми словами для нахождения ссылок на соответствующие Web-страницы, а затем для отправки заполненной формы на сервер "программно" нажимаем кнопку **Submit**. В результате получим в элементе **WebBrowser** результат работы поисковой системы, а именно множество ссылок на страницы, содержащие указанные ключевые слова (рис. 8.4).

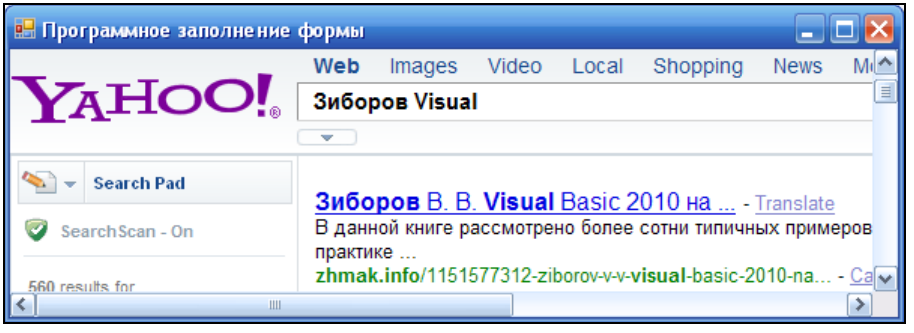


Рис. 8.4. Результат работы поисковой системы

Убедиться в работоспособности программы можно, открыв соответствующее решение в папке ЗаполнениеWeb_формы.

Глава 9

Использование функций MS Word, MS Excel, AutoCAD и MATLAB

Пример 54. Проверка правописания в текстовом поле с помощью обращения к MS Word

Пакет приложений Microsoft Office может являться сервером OLE-объектов, и его функции могут использоваться другими приложениями. Продемонстрируем такое использование. Для этого создадим программу, которая позволяет пользователю ввести какие-либо слова, предложения в текстовое поле и после нажатия соответствующей кнопки проверить орфографию введенного текста. Для непосредственной проверки орфографии воспользуемся функцией `CheckSpelling` объектной библиотеки MS Word.

Запустим Visual Studio 2010, выберем новый проект **Windows Forms Application C#**. Перетащим из панели элементов **Toolbox** в форму текстовое поле. Чтобы растянуть его на всю форму в свойстве `Multiline` текстового поля укажем `True` (разрешим введение множества строк). Также с панели элементов перетащим кнопку **Button**. Мы должны получить примерно такой дизайн, который представлен на рис. 9.1.

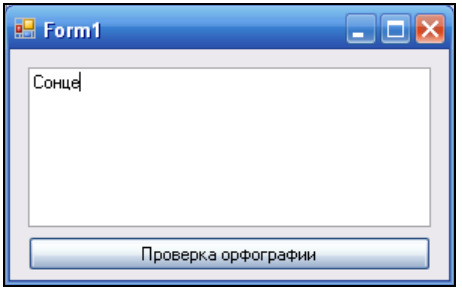


Рис. 9.1. Фрагмент работы программы проверки орфографии

Далее к текущему проекту добавим объектную библиотеку MS Word (библиотеку компонентов). Для этого в пункте меню **Project** выберем команду **Add Reference.** Затем, если на вашем компьютере установлен MS Office 2003, то на вкладке **COM** дважды щелкнем по ссылке на библиотеку **Microsoft Word 11.0 Object Libary**. Если же установлен MS Office 2007, то дважды щелкнем на ссылке **Microsoft Word 12.0 Object Library**. Эта объектная библиотека соответствует файлу, расположенному по адресу: C:\Program Files\Microsoft Office\OFFICE11\MSWORD.OLB (или ...\OFFICE12\MSWORD.OLB для MS Office 2007).

Теперь убедимся в том, что данная ссылка благополучно установлена. Для этого в обозревателе решений раскроем узел **References**, здесь среди прочих ссылок мы видим строку **Microsoft.Office.Interop.Word**. Кроме того, в папке проекта obj\x86\Debug появился файл Interop.Office.dll.

Таким образом, мы подключили библиотеку объектов MS Word. Далее введем программный код, представленный в листинге 9.1.

Листинг 9.1. Проверка орфографии

```
// Добавляем эту директиву для более коротких выражений
using Ворд = Microsoft.Office.Interop;
using System;
using System.Windows.Forms;
// Программа позволяет пользователю ввести какие-либо слова, предложения
// в текстовое поле и после нажатия соответствующей кнопки проверить
// орфографию введенного текста. Для непосредственной проверки орфографии
// воспользуемся функцией CheckSpelling объектной библиотеки MS Word.
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace Орфография
{  // В пункте меню Project выберем команду Add Reference.
   // Затем, если на вашем компьютере установлен MS Office 2007,
   // то на вкладке COM дважды щелкнем по ссылке
   // на библиотеку Microsoft Word 12.0 Object Libary.
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
         textBox1.Clear(); button1.Text = "Проверка орфографии";
      }
      private void button1_Click(object sender, EventArgs e)
      {
         var Ворд1 = new Ворд.Word.Application();
```

```
            Ворд1.Visible = false;
            Ворд1.Documents.Add(); // Открываем новый документ
            // Копируем содержимое текстового окна в документ
            Ворд1.Selection.Text = textBox1.Text;
            // Проверка орфографии:
            Ворд1.ActiveDocument.CheckSpelling();
            // Копируем результат назад в текстовое поле
            textBox1.Text = Ворд1.Selection.Text;
            Ворд1.Documents.Close(Ворд.Word.WdSaveOptions.wdDoNotSaveChanges);
            // или Ворд.ActiveDocument.Close
            //       (Ворд.Word.WdSaveOptions.wdDoNotSaveChanges);
            // Это важно:
            Ворд1.Quit();
            Ворд1 = null;
        }
    }
}
```

Как видно из текста программы, сразу после `InitializeComponent` очищается текстовое поле и инициализируется название кнопки **Проверка орфографии**. При обработке события "щелчок по кнопке" `Button1.Click` создается новый объект класса `Word.Application` и командой `Documents.Add` открывается новый документ. Далее весь введенный пользователем текст копируется в этот документ. Затем происходит непосредственная проверка орфографии командой `CheckSpelling`. Далее документ закрываем без сохранения изменений `wdDoNotSaveChanges`. После нажатия кнопки **Проверка орфографии** получим диалоговое окно, подобное представленному на рис. 9.2.

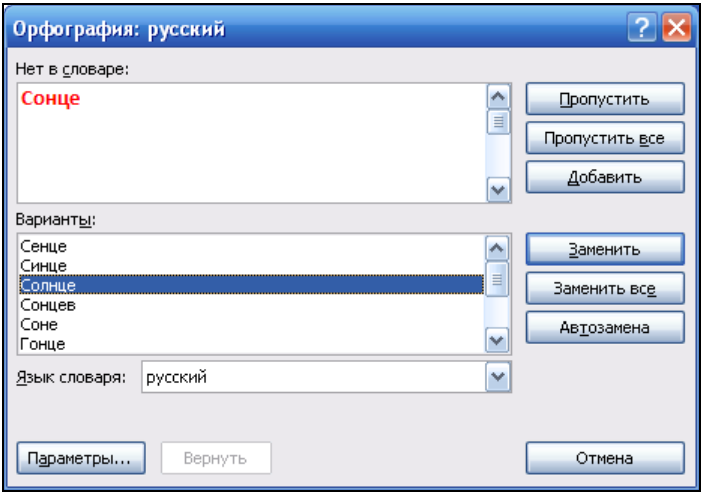


Рис. 9.2. Проверка правописания в текстовом поле

Выберем правильный вариант написания слова и щелкнем на кнопке **Заменить**, при этом диалоговое окно закроется, а в нашем текстовом поле на форме окажется исправленное слово.

Убедиться в работоспособности программы можно, открыв решение Орфография.sln в папке Орфография.

Пример 55. Вывод таблицы средствами MS Word

В данной книге мы уже рассматривали способы формирования таблицы. Здесь мы обсудим способ создания таблицы, используя функции MS Word. Вообще говоря, программировать взаимодействие программы на Visual C# 2010 с различными офисными приложениями (Word, Excel, Access, PowerPoint и т. д.), а также с AutoCAD и CorelDRAW удобно, поскольку во все эти приложения встроен язык VBA (Visual Basic for Applications), в арсенале которого используются программные объекты, названия и назначения которых в многом схожи с объектами, используемыми в Visual C#. Причем есть возможность записи макроса с последующим просмотром соответствующей VBA-программы. Например, мы хотим посмотреть, как организована вставка таблицы в редакторе MS Word. Для этого запускаем MS Word, затем в меню **Сервис** выбираем команду **Макрос | Начать запись**, далее в диалоговом окне **Запись макроса** указываем имя макроса и щелкаем на кнопке **OK**. Теперь в текст MS Word вставляем таблицу, используя пункты меню **Таблица | Вставить | Таблица** и т. д. После заполнения таблицы нажимаем кнопку **Остановить запись**. Далее с помощью комбинации клавиш <Alt>+<F11> откроем окно Microsoft Visual Basic, здесь мы увидим текст макроса на языке VBA. Из этого текста мы можем понять основной принцип, имена используемых объектов, функций, свойств и проч.

А теперь рассмотрим уже конечный результат — программу на Visual C# 2010, которая, используя функции MS Word, строит таблицу. Итак, запускаем Visual Studio 2010, выбираем новый проект, указываем шаблон **Windows Forms Application C#**. Далее к текущему проекту добавим объектную библиотеку MS Word. Для этого в меню **Project** укажем команду **Add Reference** и на появившейся вкладке **COM** дважды щелкнем по ссылке на библиотеку **Microsoft Word 11.0 Object Library** (или другая версия MS Word). На экранную форму перенесем командную кнопку **Button**, чтобы работа программы выглядела более выразительно. То есть именно после щелчка на кнопке будет формироваться таблица и вызываться MS Word для ее отображения. Далее введем программный код, представленный в листинге 9.2.

Листинг 9.2. Вывод таблицы средствами MS Word

```
// Программа вывода таблицы средствами MS Word: запускается программа,
// пользователь наблюдает, как запускается редактор MS Word
// и автоматически происходит построение таблицы
```

```
using System;
using System.Windows.Forms;
// Добавляем эту директиву для более коротких выражений
using Ворд = Microsoft.Office.Interop;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace ТаблицаWord
{
   public partial class Form1 : Form
   {
      public Form1()
      {  // В меню Project укажем команду Add Reference и на появившейся
         // вкладке COM дважды щелкнем по ссылке на библиотеку
         // Microsoft Word 12.0 Object Library
         InitializeComponent();
         button1.Text = "Пуск"; base.Text = "Построение таблицы";
      }
      private void button1_Click(object sender, EventArgs e)
      {
         string[] Imena = {"Андрей - раб", "Света-Х", "ЖЭК",
                           "Справка по тел", "Александр Степанович",
                           "Мама - дом", "Карапузова Таня",
                           "Погода сегодня", "Театр Браво"};
         string[] Tel = {"274-88-17", "+38(067)7030356",
                         "22-345-72", "009", "223-67-67 доп 32-67",
                         "570-38-76", "201-72-23-прямой моб",
                         "001", "216-40-22"};
         var Ворд1 = new Ворд.Word.Application();
         Ворд1.Visible = true;
         Ворд1.Documents.Add();   // Открываем новый документ
         Ворд1.Selection.TypeText("ТАБЛИЦА ТЕЛЕФОНОВ");
         // Создаем таблицу из 9 строк и 2 столбцов;
         // автоподбор ширины столбцов -
         //   по содержимому ячеек (wdAutoFitContent)
         Ворд1.ActiveDocument.Tables.Add(Ворд1.Selection.Range, 9, 2,
         Ворд.Word.WdDefaultTableBehavior.wdWord9TableBehavior,
         Ворд.Word.WdAutoFitBehavior.wdAutoFitContent);
         // Заполнять ячейки таблицы можно так:
         for (int i = 1; i <= 9; i++)
         {
            Ворд1.ActiveDocument.Tables[1].Cell(i, 1).
                                    Range.InsertAfter(Imena[i - 1]);
```

```
            Ворд1.ActiveDocument.Tables[1].Cell(i, 2).
                                    Range.InsertAfter(Tel[i - 1]);
        }
        // Перевести курсор (Selection) за пределы таблицы
        Ворд1.Selection.MoveDown(Ворд.Word.WdUnits.wdLine, 9);
        Ворд1.Selection.TypeText("Какой-либо текст после таблицы");
        // Сохранять нет смысла, но это решит пользователь
        //       W.ActiveDocument.SaveAs("C:\a.doc")
    }
  }
}
```

Заметим, что содержимое текстовой таблицы такое же, как и в примере 44 *(см. главу 7)*. То есть наш сюжет меняется, а действующие персонажи остаются прежними. Данные находятся в двух массивах: `Imena()` и `Tel()`.

Далее создаем экземпляр объекта `Word.Application` и открываем новый документ `Document.Add`. Затем демонстрируем, как можно добавлять какие-либо тексты в новый документ из C#-программы, например, мы вводим в активный документ текст "ТАБЛИЦА ТЕЛЕФОНОВ". Далее создаем таблицу, состоящую из девяти строк (рядов) и двух столбцов, причем ширина столбцов будет регулироваться в зависимости от содержимого ячеек (`wdAutoFitContent`). Затем в цикле заполняем ячейки таблицы и выводим курсор (`Selection`) за пределы таблицы, чтобы написать какой-либо текст.

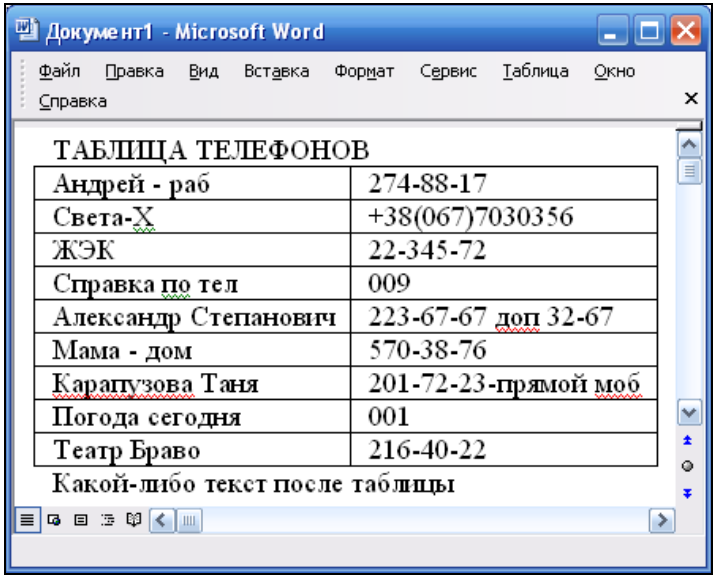


Рис. 9.3. Программно сформированная таблица в редакторе MS Word

После запуска этой программы очень красиво, прямо на наших глазах в редакторе MS Word сформируется таблица (рис. 9.3), которую при желании можно редактировать, сохранять и распечатывать на принтере.

Убедиться в работоспособности программы можно, открыв решение ТаблицаWord.sln в папке ТаблицаWord.

Пример 56. Обращение к функциям MS Excel из Visual C# 2010

Очень заманчиво обратиться из какой-нибудь вашей C#-программы к функциям Microsoft Excel. Табличный редактор MS Excel содержит очень мощные средства для сложных вычислений и анализа данных, которые могут значительно расширить возможности ваших программ.

В данной программе мы продемонстрируем буквально в трех строчках программного кода обращение к одной простой функции MS Excel, а именно получение значения числа $\pi = 3{,}14$. Число π представлено в классе `Math` языка C#. Но цель данной программы — показать легкость доступа к функциям MS Excel.

Как обычно, запустим Visual Studio 2010, далее в окне **New Project** выберем шаблон **Windows Forms Application C#** и щелкнем на кнопке **OK**. Проектируемую экранную форму сделаем совсем маленькой, поскольку число π будем выводить в строку заголовка формы.

Чтобы добавить в текущий проект возможности MS Excel, следует подключить библиотеку объектов MS Excel. Для этого в пункте меню **Project** выберем команду **Add Reference**. Затем, если на вашем компьютере установлен пакет MS Office 2003, то на вкладке **COM** дважды щелкнем по ссылке на библиотеку **Microsoft Excel 11.0 Object Library**. Если же установлен MS Office 2007, то дважды щелкнем на ссылке **Microsoft Excel 12.0 Object Library**. То есть процедура добавления новой библиотеки объектов такая же, как и в примерах об использовании возможностей MS Word, а названия пунктов меню сохранились почти такими же, как в предыдущих версиях Visual Studio. Подключить новую библиотеку объектов в текущий проект можно также через контекстное меню окна **Solution Explorer** (Обозреватель решений), щелкнув на пункте **Add Reference**.

Таким образом, мы подключили библиотеку объектов MS Excel. В окне **Solution Explorer** (чтобы его добавить, следует щелкнуть на значке **Solution Explorer**) щелкните на значке **Show All Files**, а затем раскройте дерево **References**. Одной из ветвей дерева будет ветвь **Excel**. Щелкнув правой кнопкой мыши на значке этой ветви и выбрав пункт **View in Object Browser**, мы увидим объекты данной ветви в окне **Object Browser** (Обозреватель объектов). Найдем объект `WorksheetFunction`, при этом в окне **Members of 'WorksheetFunction'** увидим доступные нам функции MS Excel для объекта `WorksheetFunction`. Теперь в программном коде обратимся к одной из этих функций, а именно функции `Pi()`.

Для этого перейдем на вкладку программного кода **Form1.cs** и напишем программный код, приведенный в листинге 9.3.

Листинг 9.3. Обращение к одной из функций MS Excel

```
// Программа обращается к одной простой функции объектной библиотеки
// MS Excel для получения значения числа Пи = 3,14
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace ExcelПи
{ // Чтобы добавить ссылку на объектную библиотеку Excel, в пункте меню
  // Project выберем команду Add Reference. Затем, если на вашем компьютере
  // установлен MS Office 2007, то на вкладке COM дважды щелкнем по ссылке
  // на библиотеку Microsoft Excel 12.0 Object Library.
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
         // Создание экземпляра класса Excel.Application
         var xl = new Microsoft.Office.Interop.Excel.Application();
         double PI = xl.WorksheetFunction.Pi();
         // Выводим значение ПИ в строку заголовка формы
         this.Text = "PI = " + PI;
      }
   }
}
```

Как видно, сразу после вызова метода `InitializeComponent` при обработке события загрузки формы создается объект `Excel.Application`, с помощью которого имеем доступ для одной из функций MS Excel, возвращающей число $\pi = 3{,}14$.

Результат работы программы показан на рис. 9.4.

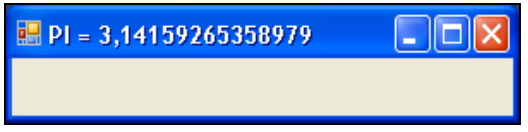


Рис. 9.4. Вывод числа π в заголовок формы

Убедиться в работоспособности программы можно, открыв решение ExcelПи.sln в папке ExcelПи.

Пользоваться функциями MS Excel в C#-программе очень перспективно. Например, оцените возможность решать сложнейшие в математическом смысле оптимизационные задачи (т. е. задачи нахождения максимума/минимума с набором ограничений), доступные в MS Excel через **Сервис | Поиск решения**.

Пример 57. Использование финансовой функции MS Excel

Рассмотрим еще один пример обращения к функциям MS Excel из программы на C# 2010. Допустим, вы взяли кредит на покупку квартиры 100 тыс. долларов под 15% годовых, срок погашения кредита — 10 лет. Требуется узнать сумму, которую вы вынуждены будете платить ежемесячно. В русскоязычном MS Excel для подобных расчетов есть функция `ПЛТ()`, на вход которой следует подать месячную процентную ставку (т. е. в нашем случае `0.15/12`), срок погашения кредита в месяцах (`120` месяцев) и размер кредита ($100 тыс.). Аналогом функции `ПЛТ()` является функция (метод) `Pmt()` класса `WorksheetFunction`, которая имеет такие же аргументы. Таким образом, мы можем написать C#-программу с обращением к функции `Pmt()` и проверить результат в русскоязычной версии MS Excel. Список всех методов (функций) объекта `WorksheetFunction` с описанием аргументов можно найти по адресу: **http://msdn.microsoft.com/en-us/library/bb225774.aspx**.

Для программирования обращений к этим функциям из программы, созданной в Visual Studio 2010, важно найти соответствие русскоязычных функций MS Excel и их аналогов в объекте `WorksheetFunction` для отладки на тестовых примерах.

Запустим Visual Studio 2010, далее закажем новый проект из шаблона **Windows Forms Application C#**. В проектируемую экранную форму из панели **Toolbox** перенесем три метки, три текстовых поля (для ввода трех вышеперечисленных аргументов функции `Pmt()`) и кнопку. В текущий проект подключаем библиотеку объектов MS Excel. Для этого в меню **Project** выберем команду **Add Reference**, затем на вкладке **COM** дважды щелкнем на ссылке **Microsoft Excel 12.0 Object Library**. Теперь можно перейти к программному коду, приведенному в листинге 9.4.

Листинг 9.4. Использование финансовой функции MS Excel

```
// Программа использует финансовую функцию Pmt() объектной библиотеки
// MS Excel для вычисления суммы периодического платежа на основе
// постоянства сумм платежей и постоянства процентной ставки
using System;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace ExcelПлт
```

```
{  // Для подключения библиотеки объектов MS Excel в пункте меню Project
   // выберем команду Add Reference. Затем, если на вашем компьютере
   // установлен MS Office 2007, то на вкладке COM дважды щелкнем по ссылке
   // на библиотеку Microsoft Excel 12.0 Object Library
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();           label1.Text = "Год. ставка в %";
         label2.Text = "Срок в месяцах";  label3.Text = "Размер кредита";
         textBox1.Clear(); textBox2.Clear(); textBox3.Clear();
         this.Text = "Расчет ежемесячных платежей"; button1.Text = "Расчет";
      }
      private void button1_Click(object sender, EventArgs e)
      {
         try
         {
            var XL = new Microsoft.Office.Interop.Excel.Application();
            double pay = XL.WorksheetFunction.Pmt(
                 (Convert.ToDouble(textBox1.Text)) / 1200,
                  Convert.ToDouble(textBox2.Text),
                  Convert.ToDouble(textBox3.Text));
            // ИЛИ, если использовать функцию Pmt() из Microsoft.VisualBasic:
            //double pay = Microsoft.VisualBasic.Financial.Pmt(
            //     (Convert.ToDouble(textBox1.Text)) / 1200,
            //      Convert.ToDouble(textBox2.Text),
            //      Convert.ToDouble(textBox3.Text));
            string ss = string.Format(
                 "Каждый месяц следует платить {0:$#.##} долларов",
                 Math.Abs(pay));
            MessageBox.Show(ss);
         }
         catch (Exception Ситуация)
         {
            MessageBox.Show(Ситуация.Message, "Ошибка",
               MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
         }
      }
   }
}
```

Как видно из текста программы, сразу после вызова метода `InitializeComponent` очищаются (`Clear`) текстовые поля, а также подписываются названия этих полей с помощью меток `label1`—`label3` и присваивается название кнопки `Button1`.

При обработке события "щелчок на кнопке" **Расчет** создается объект `Excel.Application`. Объект `Excel.Application` обеспечивает доступ к функциям MS Excel, в частности к функции `Pmt()`. На вход функции `Pmt()` подаем значения текстовых полей, конвертированных из строкового типа в тип `Double`. При этом первый аргумент функции переводим из годовой процентной ставки в месячную ставку в сотых долях единицы, поэтому делим на 1200. На выходе функции `Pmt()` получаем размер месячного платежа, который выводим, используя функцию `MessageBox.Show`.

Обращение к функциям MS Excel оформляем в блоке `try...catch` для обработки исключительных ситуаций (`Exception`). Замечу, что в данной программе мы не предусмотрели диагностику обязательного заполнения всех полей, а также диагностику ввода только числовых данных, чтобы не перегружать текст программы многочисленными очевидными подробностями.

Интерфейс программы показан на рис. 9.5.

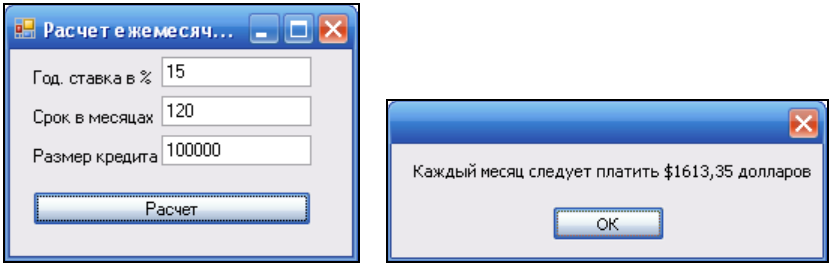


Рис. 9.5. Расчет ежемесячных платежей

Убедиться в работоспособности программы можно, открыв решение ExcelПлт.sln в папке ExcelПлт.

В данном примере на простом примере мы рассмотрели, как легко подключиться к библиотеке объектов MS Excel и пользоваться ее функциями. Однако функция `Pmt()` имеется также в среде Visual Studio 2010 в пространстве имен `Microsoft.VisualBasic.Financial` точно с такими же параметрами. (Более того, эта функция была еще в Visual Basic 6.) Для обращения к этой функции потребовалось бы подключение к Visual Basic, как мы это делали в примере 14 *(см. главу 2)*. То есть следовало бы в проект *добавить ссылку на библиотеку* Microsoft.VisualBasic.dll. Для этого в пункте меню **Project** надо выбрать команду **Add Reference**, а на вкладке **.NET** дважды щелкнуть на ссылке **Microsoft.VisualBasic**. В этом случае можно было бы обращаться к функции `Pmt()`, как это представлено в комментарии. Однако в данном примере показана принципиальная возможность работы с функциями MS Excel из C#-программы.

Пример 58. Решение системы уравнений с помощью функций MS Excel

Используя функции MS Excel, в своей программе, созданной в Visual C# 2010, можно решать и более серьезные задачи. Например, рассмотрим, как решить *систему линейных алгебраических уравнений*:

$$X_1 + X_2 + X_3 = 6$$
$$X_1 + X_2 = 3$$
$$X_2 + X_3 = 5$$

через *обратную матрицу*. Исходную систему уравнений запишем в матричном виде:

$$A \cdot X = L.$$

Здесь A — матрица коэффициентов при неизвестных;

X — вектор неизвестных X_1, X_2, X_3;

L — вектор свободных членов 6, 3, 5.

Тогда *решением системы* будет выражение

$$X = A^{-1} \cdot L,$$

где X^{-1} — обратная матрица.

Для нахождения обратной матрицы в русскоязычной версии MS Excel есть функция `МОБР()`, а объект `WorksheetFunction` в библиотеке объектов Microsoft Excel имеет функцию `MInverse()`. Для умножения обратной матрицы на вектор свободных членов есть соответственно функции `МПРОИЗ()` и `MMult()`. Таких функций нет в Visual Studio 2010, и в данном случае мы получаем реальный положительный эффект от подключения к функциям MS Excel.

Для программной реализации решения поставленной задачи запустим Visual Studio 2010. Далее выберем новый проект из шаблона **Windows Forms Application C#** и щелнем на кнопке **OK**. В проектируемую форму из панели **Toolbox** добавим метку и растянем ее побольше и симметрично относительно формы. На ней будем формировать ответ задачи. Кроме того, добавим библиотеку объектов MS Excel. Для этого в пункте меню **Project** выберем команду **Add Reference** и на вкладке **COM** отметим библиотеку **Microsoft Excel 12.0 Object Library**, а затем щелкнем на кнопке **OK**.

Программу построим следующим образом (листинг 9.5): сразу после выполнения процедуры `InitializeComponent` прямо в тексте программы зададим (инициализируем) прямую матрицу в виде двумерного массива и вектор свободных членов в виде одномерного массива. Затем после решения системы выведем ответ на метку `label1`.

Листинг 9.5. Решение системы линейных алгебраических уравнений

```
// Программа решает систему уравнений с помощью функций объектной
// библиотеки MS Excel
```

```
using System;
using System.Windows.Forms;
using XL = Microsoft.Office.Interop.Excel.Application;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace ExcelСЛАУ
{  // Для подключения библиотеки объектов MS Excel в пункте меню Project
   // выберем команду Add Reference. Затем, если на вашем компьютере
   // установлен MS Office 2007, то на вкладке COM дважды щелкнем по ссылке
   // на библиотеку Microsoft Excel 12.0 Object Library
   public partial class Form1 : Form
   {
      public Form1()
      {
          InitializeComponent();
         // Матричное уравнение AX = L решаем через
         // обратную матрицу: X = A(-1)L.
         // Здесь (-1) - "знак" обратной матрицы.
         // Решаем систему
         // X1 + X2 + X3 = 6
         // X1 + X2      = 3
         //      X2 + X3 = 5
         // Для этой системы прямая матрица будет иметь вид
         // double[,] A = new double[n, n]; // - матрица коэффициентов
         double[,] A = {{1, 1, 1},
                        {1, 1, 0},
                        {0, 1, 1}};
         // double[] L = new double[n]; // - вектор свободных членов
         // Свободные члены
         double[] L = { 6, 3, 5 };
         XL XL1 = new XL();
         // Вычисление детерминанта матрицы A
         double det_A = XL1.Application.WorksheetFunction.MDeterm(A);
         // Если det_A != 0, то выход из процедуры:
         if (Math.Abs(det_A) < 0.01)
         {
            label1.Text = "Система не имеет решения, поскольку\n\n" +
                          "определитель равен нулю";
            return;
         }
         // Получение обратной матрицы oA:
         Object oA = XL1.Application.WorksheetFunction.MInverse(A);
```

```
            // Умножение обратной матрицы на вектор свободных членов
            Object[,] X = XL1.Application.WorksheetFunction.MMult(
                        oA, XL1.Application.WorksheetFunction.Transpose(L));
            // ТАК НАПИСАТЬ ПРАВОМОЧНО
            // double X1 = (double)X[1, 1];
            // Формирование ответа:
            label1.Text = string.Format(
               "Неизвестные равны:\n\nX1 = {0};  X2 = {1};  X3 = {2}.",
                                                X[1, 1], X[2, 1], X[3, 1]);
        }
    }
}
```

Как видно из текста программы, задаем прямую матрицу, причем присваиваем значения коэффициентов сразу при объявлении двумерного массива. Аналогично поступаем с вектором свободных членов. Согласно требованию объекта `WorksheetFunction` возвращаемые обратная матрица и вектор неизвестных должны быть объявлены как объектные переменные. Вначале вычисляем детерминант (определитель) прямой матрицы, используя функцию MS Excel `Mdeterm()`. Если прямая матрица *плохо обусловлена*, т. е. определитель по абсолютному значению меньше `0.01`, то выходим из процедуры и сообщаем пользователю в метке `label1`, что система не имеет решения. Если определитель матрицы больше `0.01`, то с помощью функции MS Excel `MInverse()` находим обратную матрицу. Далее обратную матрицу с помощью функции MS Excel `MMult()` умножаем на вектор неизвестных, но прежде его следует сделать вертикальным массивом с помощью функции MS Excel `Transpose`, т. е. транспонировать массив `L`. Следующим оператором форматируем ответ в метке `label1`.

Результат работы программы приведен на рис. 9.6. Убедиться в работоспособности программы можно, открыв решение ExcelСЛАУ.sln в папке ExcelСЛАУ.

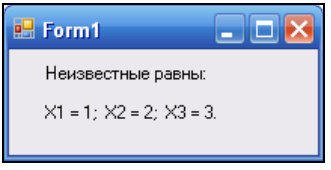


Рис. 9.6. Решение системы линейных алгебраических уравнений

Как видим, довольно сложные задачи можно решать в коротенькой программе благодаря обращению к функциям MS Excel. Причем на компьютере, где будет работать данная программа, вовсе не обязательно должен быть инсталлирован MS Excel. Однако инсталляция вашей программы должна содержать соответствующую dll-библиотеку.

Пример 59. Построение диаграммы средствами MS Excel

Очень часто необходимо изобразить на графике (диаграмме) какие-либо экономические показатели или технические измерения (геодезические, метрологические, астрономические), например, с целью принятия каких-либо решений. Часто сделать это надо очень оперативно. Для этих целей в ячейки рабочего листа MS Excel можно ввести измеренные данные, а далее чтобы получить график, построенный по этим данным, воспользоваться пунктами меню **Вставка | Диаграмма**. В данном разделе я покажу, как можно очень быстро получить график (диаграмму) из программы Visual C# 2010, используя средства (объекты компонентной библиотеки) MS Excel.

Запустим Visual Studio 2010, далее, поскольку экранная форма нам не нужна, выбираем новый проект из шаблона **Console Application**. Теперь к текущему проекту добавим библиотеку объектов MS Excel. Для этого в пункте меню **Project** выберем команду **Add Reference**, далее на вкладке **COM** отметим библиотеку **Microsoft Excel 12.0 Object Library** и щелкнем на кнопке **OK**. Затем на вкладке программного кода введем текст, приведенный в листинге 9.6.

Листинг 9.6. Построение диаграммы средствами MS Excel

```
// Программа строит график (диаграмму) средствами объектов
// компонентной библиотеки MS Excel
using XL = Microsoft.Office.Interop.Excel;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace ExcelГрафик
{  // Для подключения библиотеки объектов MS Excel в пункте меню Project
   // выберем команду Add Reference. Затем на вкладке COM дважды щелкнем
   // по ссылке на библиотеку Microsoft Excel 12.0 Object Library
   class Program
   {
      static void Main(string[] args)
      {
         XL.Application XL1 = new XL.Application();
         XL1.Workbooks.Add();
         //XL1.ActiveSheet.Range["A1"].Value = "Месяц";
         XL1.ActiveSheet.Range["A2"].Value = "Март";
         XL1.ActiveSheet.Range("A3").Value = "Апр";
         XL1.ActiveSheet.Range("A4").Value = "Май";
         XL1.ActiveSheet.Range("A5").Value = "Июнь";
         XL1.ActiveSheet.Range("A6").Value = "Июль";
```

```
            //XL1.ActiveSheet.Range("B1").Value = "Продажи:";
            XL1.ActiveSheet.Range("B2").Value = 138;
            XL1.ActiveSheet.Range("B3").Value = 85;
            XL1.ActiveSheet.Range("B4").Value = 107;
            XL1.ActiveSheet.Range("B5").Value = 56;
            XL1.ActiveSheet.Range("B6").Value = 34;
            XL1.Charts.Add();
            // Задаем тип графика "столбчатая диаграмма" (гистограмма):
            XL1.ActiveChart.ChartType = XL.XlChartType.xlColumnClustered;
            // Отключаем легенду графика:
            XL1.ActiveChart.HasLegend = false;
            XL1.ActiveChart.HasTitle = true;
            XL1.ActiveChart.ChartTitle.Characters.Text =
                "ПРОДАЖИ ЗА ПЯТЬ МЕСЯЦЕВ";
            // Подпись оси x
            XL1.ActiveChart.Axes(XL.XlAxisType.xlValue).HasTitle = true;
            XL1.ActiveChart.Axes(XL.XlAxisType.xlValue).AxisTitle.
                                        Characters.Text = "Уровни продаж";
            // Подпись оси y
            XL1.ActiveChart.Axes(XL.XlAxisType.xlCategory).HasTitle = true;
            XL1.ActiveChart.Axes(XL.XlAxisType.xlCategory).AxisTitle.
                                        Characters.Text = "Месяцы";
            // Сохранение графика в растровом файле:
            XL1.ActiveChart.Export("C:\\ExcelГрафик.jpg");
            XL1.Visible = true;
        }
    }
}
```

Вначале программного кода создаем объект `Excel.Application` и рабочую книгу. Далее заполняем ячейки первого листа: вначале в ячейки `Ai` пишем подписи ординат гистограммы, а в ячейки `Bi` — значения ординат. Затем задаем тип диаграммы — `xlColumnClustered`, что соответствует гистограмме (столбиковой диаграмме). Далее указываем название гистограммы и подписываем горизонтальную и вертикальную оси. Затем с помощью функции `Export()` сохраняем полученную диаграмму на диске в виде jpg-файла. Графическое отображение этого файла можно посмотреть на рис 9.7.

Далее делаем видимой (`Visible = true`) построенную диаграмму на рабочем листе MS Excel.

Убедиться в работоспособности программы можно, открыв решение ExcelГрафик.sln в папке ExcelГрафик.

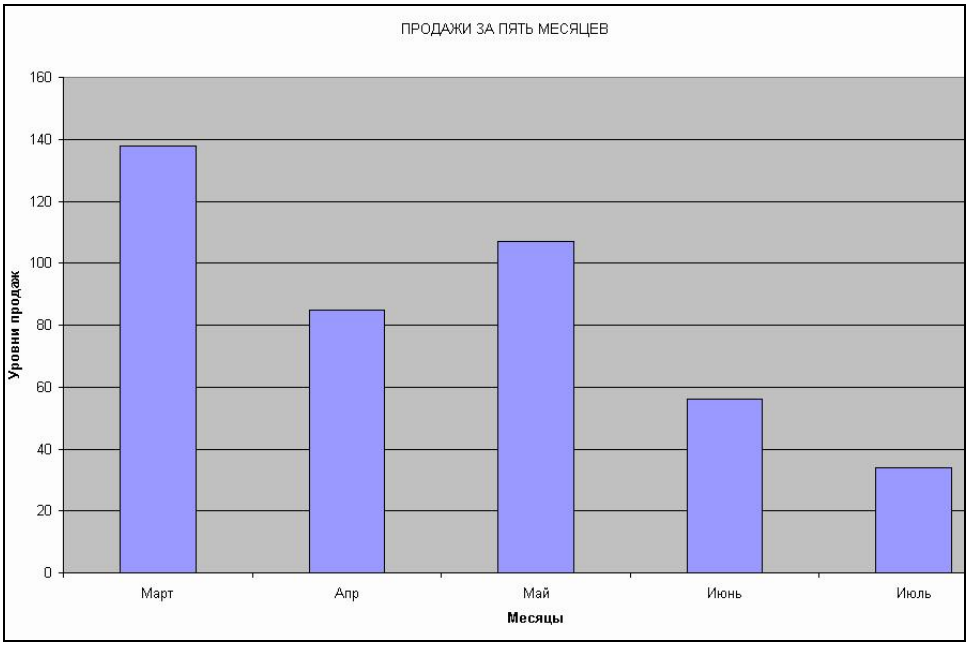


Рис. 9.7. Графическое отображение полученного jpg-файла

Пример 60. Управление функциями AutoCAD из программы на Visual C# 2010

Если результатом работы вашей программы должен быть какой-либо векторный чертеж (техническая документация, строительный чертеж, географическая карта и проч.), то самый быстрый путь создания такого приложения — это обращение к функциям AutoCAD из вашей C#-программы. AutoCAD (Computer-Aided Design) — это 2- и 3-мерная система автоматизированного проектирования и черчения. Эта система, также как и пакет приложений Microsoft Office, может являться сервером OLE-объектов, и его функции могут использоваться другими приложениями.

Графическими примитивами векторной графики являются отрезки, дуги, окружности, тексты, которые можно выводить под различными углами к горизонту, и, может быть, еще некоторые простейшие геометрические фигуры. Чертеж, подлежащий построению, состоит из совокупности таких элементов. Программа на C# 2010 путем обращения к соответствующим функциям AutoCAD формирует такой чертеж и записывает его в dwg-файл. Пользователь может просмотреть этот файл в среде AutoCAD, отредактировать его и вывести на печать.

Приступаем к программированию поставленной задачи. Как обычно после запуска Visual Studio 2010, выбираем шаблон **Windows Forms Application C#**. Далее

следует подключить библиотеку объектов AutoCAD, для этого в пункте меню **Project** выбираем команду **Add Reference**, затем на вкладке **COM** дважды щелкаем на ссылке **AutoCAD 2000 Object Library**. Эта ссылка соответствует файлу ACAD.TLD в соответствующей папке Program Files, где расположен результат инсталляции AutoCAD. Теперь после компиляции проекта в папке obj\x86\Debug появится файл Interop.AutoCAD.dll.

Таким образом, мы подключили библиотеку объектов AutoCAD версии 2000. Если на вашем компьютере инсталлирована другая версия AutoCAD, то действуйте аналогично. Программа, выводящая в dwg-файл два отрезка, а также две горизонтально и вертикально ориентированных строки текста, представлена в листинге 9.7.

Листинг 9.7. Построение отрезков и двух строк текста в AutoCAD

```
// Программа строит средствами объектов библиотеки AutoCAD элементарный
// чертеж из отрезков и некоторого текста. Этот чертеж сохраняется в файле
// формата DWG. Конкретнее: эта программа запускает AutoCAD 2000i, рисует
// два отрезка, два текстовых объекта, сохраняет чертеж в файле C:\Чертеж.dwg
// и завершает работу AutoCAD
using System;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace ACADЧертеж
{ // Следует подключить библиотеку объектов AutoCAD. Для этого надо выбрать
  // Project | Add Reference — вкладка COM — AutoCAD 2000 Object Library — OK
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
         AutoCAD.AcadApplication ACAD1 = new AutoCAD.AcadApplication();
         AutoCAD.AcadDocuments Docs1 = ACAD1.Documents;
         AutoCAD.AcadDocument Doc1 = Docs1.Add();
         // Видимость:
         ACAD1.Visible = true;
         // Точки:
         double[] T1 = { 10, 10, 0 };
         double[] T2 = { 200, 200, 0 };
         double[] T3 = { 200, 10, 0 };
         double[] T4 = { 15, 200, 0 };
         // Нарисовать отрезок от точки T1 до точки T2:
         Doc1.ModelSpace.AddLine(T1, T2);
         // Нарисовать отрезок красным цветом:
         Doc1.ModelSpace.AddLine(T2, T3).Color = AutoCAD.ACAD_COLOR.acRed;
```

```
            // Горизонтальный текст (с разворотом 0 градусов)
            Doc1.ModelSpace.AddText("Горизонтальный", T4, 22);
            // Вертикальный текст с разворотом на 90 град = ПИ/2
            Doc1.ModelSpace.AddText("Вертикальный", T1, 22).Rotation =
               Math.PI / 2;
            // Сохраняем чертеж на диске:
            Doc1.SaveAs("C:\\Чертеж.dwg");
            ACAD1.Quit();
        }
    }
}
```

Как видно из программного кода, весь процесс работы программы происходит сразу после выполнения процедуры `InitializeComponent`. Вначале создается объект класса `AutoCAD.Application`. Далее создаем коллекцию документов `AcadDocuments` и открываем (делаем активным) один документ предложением `Docs.Add`.

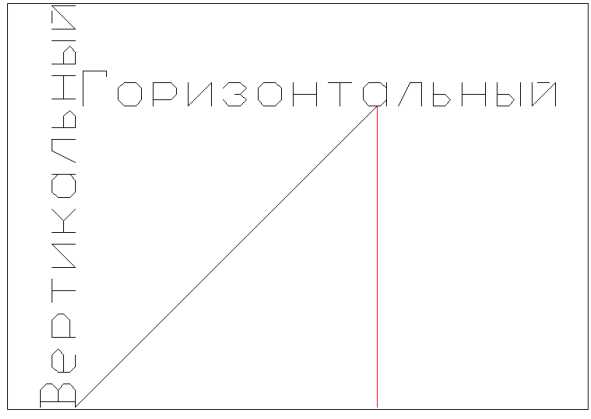


Рис. 9.8. Отображение полученного dwg-файла в системе AutoCAD

Затем задаем видимость работы AutoCAD `Visible = True`, при этом AutoCAD только мелькнет на экране. (Заметьте, что для конечной цели, т. е. для получения dwg-файла, видимость не обязательна.) Далее задаем четыре точки, которые будут участвовать в построении чертежа. Обратите внимание, что каждая точка имеет три координаты, хотя мы собираемся строить плоский чертеж. Третью координату мы будем воспринимать, как напоминание того, что AutoCAD способен строить трехмерные чертежи.

Затем рисуем два отрезка `AddLine` через точки `T1`, `T2` и `T2`, `T3`. Причем мы показали, как второй отрезок задать красным цветом. Далее подаем команду выводить текст *горизонтально*, затем другой текст — *вертикально* с разворотом на 90°, т. е. $\pi/2$. Затем, используя метод `SaveAs`, записываем построенный в документе чертеж в dwg-файл на логический диск C:.

В результате работы этой программы получаем чертеж в системе AutoCAD, подобный представленному на рис. 9.8.

Убедиться в работоспособности программы можно, открыв решение АСАДЧертеж.sln в папке АСАДЧертеж.

Пример 61. Вызов MATLAB из вашей программы на Visual C# 2010

Можно вызывать MATLAB из вашей C#-программы. Среда MATLAB является стандартным мощным инструментом для работы в различных отраслях математики. При подготовке этого примера автор пользовался наиболее распространенной версией MATLAB 6.5. В данном примере продемонстрируем подготовку вводных данных для MATLAB, создание экземпляра объекта типа MATLAB и непосредственный его вызов на выполнение. В результате мы увидим построение графика функции $y = \sin(x) \cdot e^{-x}$. Для программирования этой задачи запустим Visual Studio 2010 и закажем новый проект шаблона **Windows Forms Application C#**. Из панели элементов **Toolbox** перенесем командную кнопку **Button**, чтобы обращение к среде MATLAB происходило при щелчке на этой кнопке и выглядело бы наиболее выразительно. Далее на вкладке программного кода введем текст из листинга 9.8.

Листинг 9.8. Использование возможностей среды MATLAB

```
// Программа, подготовив соответствующие команды для MATLAB, вызывает
// его на выполнение этих команд. В результате строится затухающая
// синусоида y = sin(x)*exp(-x)
using System;
using System.Windows.Forms;
// Добавляем директиву System.Reflection:
using System.Reflection;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace MatlabВызов
{  // Для успешной работы программы нет необходимости добавлять ссылку на
   // объектную библиотеку через Project | Reference. Однако на компьютере
   // MATLAB должен быть установлен
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
         button1.Text = "Вызвать MATLAB";
      }
```

```
        private void button1_Click(object sender, EventArgs e)
        {  // Получить тип приложения MATLAB:
           Type ТипМатЛаб = Type.GetTypeFromProgID("Matlab.Application");
           // Создать экземпляр объекта типа MATLAB:
           object МатЛаб = Activator.CreateInstance(ТипМатЛаб);
           // Подготавливаем команды для MATLAB:
           object[] Команды = new Object[]
           // { "surf(peaks)" };
              { "x = 0:0.1:6.28; y = sin(x).*exp(-x); plot(x,y)" };
           // { "s = sin(0.5); c = cos(0.5); y = s*s+c*c; y" };
           // Вызываем MATLAB, подавая ему на вход подготовленные команды:
           object Результат =
              ТипМатЛаб.InvokeMember("Execute", BindingFlags.InvokeMethod,
                                     null, МатЛаб, Команды);
           // MessageBox.Show(Результат.ToString());
        }
    }
}
```

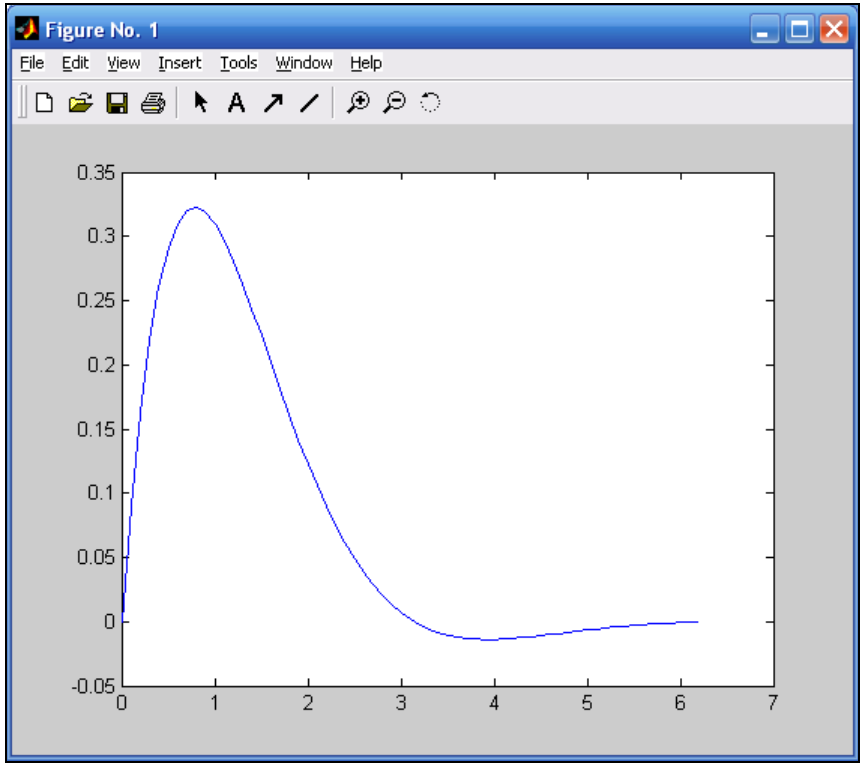


Рис. 9.9. График функции, построенный в среде MATLAB

Как видно из текста программы, при обработке события "щелчок на кнопке" в переменную `ТипМатЛаб` получаем тип приложения MATLAB. Далее создаем экземпляр объекта этого типа. Затем подготавливаем три команды для MATLAB, разделенные точкой с запятой. Первая команда `"x = 0:0.1:6.28;"` задает вектор **x** (набор чисел) от нуля до 2π (6,28) с шагом 0,1. Вторая команда `"y = sin(x).*exp(-x);"` вычисляет второй вектор по значениям первого вектора. Третья команда `plot` создает график зависимости `y` от `x`. Метод `Execute` выполняет в среде MATLAB подготовленные команды. В результате обращения к MATLAB получим построенный график заданной функции (рис. 9.9).

В комментариях приведены и другие команды, которые можно выполнить, подключаясь к среде MATLAB.

Убедиться в работоспособности программы можно, открыв решение MatlabВызов.sln в папке MatlabВызов.

Пример 62. Решение системы уравнений путем обращения к MATLAB

Основной особенностью языка MATLAB являются его широкие возможности по работе с матрицами, которые создатели языка выразили в лозунге "Думай векторно" (от англ. *Think vectorized*). Изначально среда MATLAB эволюционизировалась с задач матричной алгебры, отсюда и слово MATLAB означает матричная лаборатория (matrix laboratory). Решить систему уравнений, глядя на предыдущий пример, очень просто, нужно всего лишь знать, как строятся команды в MATLAB. Продемонстрируем процесс решения системы линейных уравнений на следующем примере.

$$X_1 + X_2 + X_3 = 6$$
$$X_1 + X_2 = 3$$
$$X_2 + X_3 = 5$$

Данную систему решим через *обратную матрицу*.

Для программирования этой задачи запустим Visual Studio 2010 и закажем новый проект шаблона **Windows Forms Application C#**. Из панели элементов **Toolbox** перенесем командную кнопку **Button**. Далее на вкладке программного кода введем текст из листинга 9.9.

Листинг 9.9. Решение системы линейных уравнений с помощью MATLAB

```
// Программа, подготовив команды для решения системы уравнений в среде
// MATLAB, вызывает его на выполнение этих команд. В результате получаем
// решение, которое выводим его на экран с помощью MessageBox
using System;
using System.Reflection;
using System.Windows.Forms;
```

```
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace MatlabСлау
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
         button1.Text = "Решить СЛАУ";
      }
      private void button1_Click(object sender, EventArgs e)
      {
         // Матричное уравнение AX = L решаем через
         // обратную матрицу: X = A(-1)L.
         // Здесь (-1) - "знак" обратной матрицы.
         // Решаем систему
         // X1 + X2 + X3 = 6
         // X1 + X2     = 3
         //     X2 + X3 = 5
         // Для решения этой системы в MATLAB следует подать такие команды:
         // A = [1 1 1; 1 1 0; 0 1 1]; L = [6; 3; 5];
         // % здесь задание прямой матрицы A и вектора свободных членов L
         // X = inv(A)*L % умножение обратной матрицы на L" };
         // % - это признак комментария в MATLAB
         // Получить тип приложения MATLAB:
         Type ТипМатЛаб = Type.GetTypeFromProgID("Matlab.Application");
         // Создать экземпляр объекта типа MATLAB:
         object МатЛаб = Activator.CreateInstance(ТипМатЛаб);
         // Подготавливаем команды для MATLAB:
         object[] Команды = new Object[]
             { "A = [1 1 1; 1 1 0; 0 1 1]; L = [6; 3; 5]; " +
               "X = inv(A)*L % обратная матрица inv" };
         // Вызываем MATLAB, подавая ему на вход подготовленные команды:
         object Результат = ТипМатЛаб.InvokeMember("Execute",
                     BindingFlags.InvokeMethod, null, МатЛаб, Команды);
         // Таким образом мы могли бы вывести решение на экран:
         // MessageBox.Show(Результат.ToString());
         // Однако этот результат будет внутри строки, а хотелось бы
         // получить ответ в массив double для дальнейшей обработки.
         // Этот массив можно получить методом GetFullMatrix из среды
         // MATLAB, как показано ниже
```

```
            var p = new ParameterModifier(4);
            p[0] = false; p[1] = false; p[2] = true; p[3] = true;
            ParameterModifier[] mods = { p };
            double[,] X = new double[3, 1];
            object[] Аргументы = new object[] { "X", "base", X, new double[0] };
            // Здесь "X" - это название матрицы, которую мы хотим получить.
            // "base" является названием рабочей среды MATLAB, где следует
            // искать матрицу "X".
            Результат = ТипМатЛаб.InvokeMember("GetFullMatrix", BindingFlags.
                     InvokeMethod, null, МатЛаб, Аргументы, mods, null, null);
            // Решение системы получаем в матрицу X:
            X = (double[,])Аргументы[2];
            string Строка = string.Format("X1 = {0};  X2 = {1};  X3 = {2};",
                                          X[0, 0], X[1, 0], X[2, 0]);
            MessageBox.Show(Строка);
        }
    }
}
```

Как видно из программного кода, подход к решению задачи аналогичен предыдущему примеру. Мы реализовали обращение к MATLAB, используя метод `Execute`, решение системы получили в переменную `Результат`. И, как показано в комментарии, можем вывести результат вычислений на экран с помощью `MessageBox`. Однако для дальнейшей работы с полученным вектором неизвестных желательно иметь его в виде массива `Double`, а не в виде строки. Конечно, можно выделить из строки решения каждое значение неизвестного с помощью опереаций со строками, используя функцию `Split`, которая возвращает строковый массив, содержащий подстроки данного экземпляра.

Однако существует более красивое решение. Оно заключается в использовании метода `GetFullMatrix`. Технологию этого использования мы привели в данной программе. Здесь наиболее важным параметром является объектная переменная `Аргументы`. Ее первым компонентом является компонент `"X"`, содержащий название матрицы, которую мы хотим получить из среды MATLAB, второй компонент — `"base"` является названием рабочей среды (workspace) MATLAB, где следует искать матрицу `"X"`. Согласно документации, в среде MATLAB мы имеем две основные рабочие среды: "base" и "global". Третьим компонентом является массив `X`, куда получаем результат решения из среды MATLAB, а четвертым компонентом — массив мнимой части решения, которой в нашей задаче нет, но для общности технологии требуется ее формальное присутствие.

О других способах подключения к MATLAB можно узнать на сайте компании The MathWorks, производителя MATLAB, **www.mathworks.com**. Убедиться в работоспособности обсуждаемой программы можно, открыв решение MatlabClay.sln в папке MatlabClay.

Глава 10

Обработка баз данных с использованием технологии ADO.NET

Пример 63. Создание базы данных SQL Server

Покажем, как можно создать базу данных SQL Server в среде Visual Studio 2010. В этой простейшей базе данных будет всего одна таблица, содержащая сведения о телефонах ваших знакомых, т. е. в этой таблице будем иметь всего три колонки: **Имя**, **Фамилия** и **Номер телефона**.

Запустим Visual Studio 2010, закажем новый проект шаблона **Windows Forms Application C#**, зададим имя — БД_SQL_Server. Далее непосредственно создадим новую базу данных SQL Server. Для этого в пункте меню **Project** выберем команду **Add New Item** (т. е. добавить новый элемент) и в появившемся одноименном окне укажем шаблон **Local Database** (локальная база данных), введем имя файла (в поле **Name**) Vic.sdf и последовательно щелкнем на кнопках **Add**, **Next**, **Finish**.

Теперь добавим таблицу в базу данных, для этого в пункте меню **View** выберем команду **Server Explorer** (Обозреватель баз данных). В обозревателе баз данных развернем узел (щелчком мыши) **Vic.sdf** и выберем **Tables** (Таблицы). Щелкнем правой кнопкой мыши на пункте **Tables**, а затем выберем пункт **Create Table** (Создать таблицу). Откроется окно **New Table**. Назовем новую таблицу (поле **Name**) **БД телефонов**. Заполним структуру таблицы, как показано на рис. 10.1.

Нажмем кнопку **OK**, чтобы создать таблицу и закрыть окно **New Table**.

Чтобы исключить повторяющиеся записи (т. е. строки в таблице), следует назначить первичные ключи (или ключевые столбцы). *Ключевым столбцом* назначают столбец в таблице, который всегда содержит уникальные (неповторяющиеся в данной таблице) значения. Однако в нашей таблице могут быть люди с одинаковыми именами или одинаковыми фамилиями, т. е. в нашей таблице в качестве первичных ключей следует использовать одновременно два столбца: столбец **Имя** и столбец **Фамилия**. Представьте себе, что у нас уже есть сотни записей в таблице, и

при попытке ввести вторую строку, содержащую то же самое значение, появляется сообщение об ошибке. Это очень технологично, удобно!

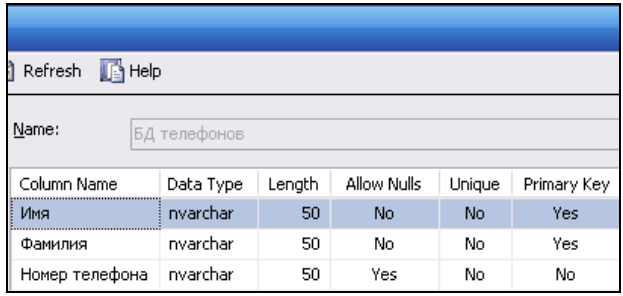


Рис. 10.1. Формирование структуры новой таблицы

Чтобы добавить первичные ключи в таблицу, в **Server Explorer** развернем узел **Tables**. Далее щелкнем правой кнопкой мыши на нашей только что созданной таблице и выберем пункт **Edit Table Schema**, затем для полей **Имя** и **Фамилия** укажем для параметра **Allow Nulls** значение **No**, т. е. обяжем нашего пользователя всегда заполнять эти поля (ячейки таблицы). Далее для параметра **Unique** (Являются ли эти поля уникальными?) ответим **No**, поскольку и имена, и фамилии повторяются. И, наконец, назначим колонки **Имя** и **Фамилия** *первичными ключами* (**Primary Key** — **Yes**). Нажмем кнопку **OK** для сохранения этих настроек и закрытия окна **Edit Table - БД телефонов**.

Теперь добавим данные в таблицу. Для этого в окне **Server Explorer** щелкнем правой кнопкой мыши на пункте **БД телефонов** и выберем команду **Show Table Data**. Откроется окно данных таблицы, как показано на рис. 10.2, но только пока пустое.

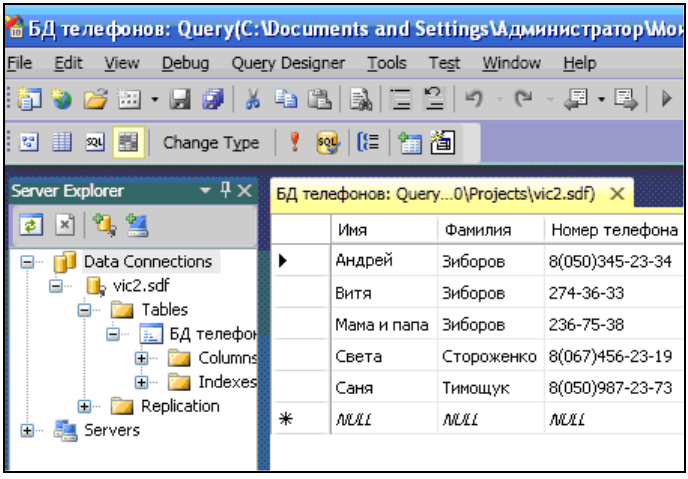


Рис. 10.2. Заполнение ячеек таблицы

Далее заполним данную таблицу. Для нашей демонстрационной цели введем пять строчек в эту таблицу. После ввода в меню **File** выберите команду **Save All** для сохранения проекта и базы данных. Теперь убедитесь, что в папке проекта появился файл Vic.sdf. Его можно открыть вне проекта с помощью Microsoft Visual Studio 2010 для редактирования базы данных.

Пример базы данных можно найти в папке БД_SQL_Server.

Пример 64. Отображение таблицы базы данных SQL Server в экранной форме

Имея базу данных, например базу данных SQL Server, в виде файла Vic.sdf, созданного в предыдущем разделе, покажем, как очень легко можно вывести таблицу из этой базы в экранную форму.

Для этой цели запустим Visual Studio 2010, закажем новый проект шаблона **Windows Application C#** и укажем имя проекта — БД_SQL_Server2. Далее в меню **View** выберем команду **Server Explorer**. Щелкнув правой кнопкой мыши на значке **Data Connections**, добавим соединение — **Add Connection**, а в качестве источника данных **Data source** укажем **Microsoft SQL Server Compact 3.5** и щелкнем на кнопке **OK**. Далее, нажмем кнопку **Browse**, найдем созданный в прошлом проекте файл базы данных Vic.sdf и нажмем кнопку **OK**. Теперь в окне **Server Explorer** мы видим значок базы данных Vic.sdf.

Далее в меню **Data** выберем команду **Add New Data Source**, тем самым мы запустим мастер **Data Source Configuration Wizard**. Здесь последовательно нажимаем кнопки **Next**, **Dataset**, **Next**. На запрос мастера копировать ли файл Vic.sdf в наш текущий проект нажмем кнопку **Да**.

В результате в окне **Solution Explorer** появится значок базы данных Vic.sdf и значок объекта vicDataSet.xsd, но база данных в текущем проекте будет пустой. Для заполнения базы данных в меню **Data** укажем **Show Data Source** и в появившемся окне **Data Source**, щелкнув правой кнопкой мыши на значке **vicDataSet**, выберем пункт **Configure Data Source With Wizard — Finish**. Теперь из окна **Server Explorer** перетащим узел **БД телефонов** на появившуюся вкладку **vicDataSet.xsd**. Затем из окна **Data Source** перетащим узел **БД телефонов** на вкладку конструктора формы **Form1.cs[Design]**. Как видно из рис. 10.3, автоматически появилось несколько программных объектов, необходимых для отображения данных из таблицы базы данных.

Заметьте, мы пока не написали *ни одной строчки* программного кода. Теперь запускаем отладку программы — нажимаем клавишу <F5>.

Как видим (рис. 10.4), в форме благодаря элементам управления **DataGridView** и **BindingNavigator** появилось отображение таблицы из базы данных. Мы имеем возможность модифицировать существующие записи (строки), добавлять новые записи (**New**), удалять записи из таблицы (**Delete**), сохранять данные (**Save Data**) с

помощью соответствующих кнопок. То есть *имеем полный набор* основных четырех функций по редактированию таблицы базы данных.

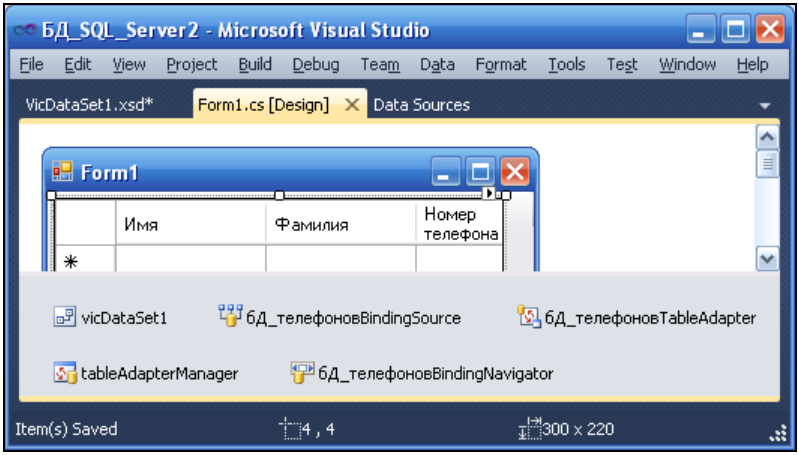


Рис. 10.3. Проект формы с объектами для отображения таблицы базы данных

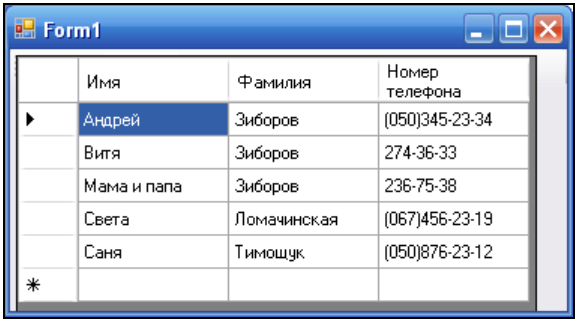


Рис. 10.4. Фрагмент работы программы

Убедиться в работоспособности данной программы можно, открыв файл решения БД_SQL_Server2.sln в папке БД_SQL_Server2. Обратите внимание на то, что копия исходной базы данных Vic.sdf находится в папке проекта bin\Debug, именно этот файл претерпевает изменения при редактировании таблицы в данном проекте.

Создание базы данных в среде MS Access

Вначале создадим базу данных (БД) vic.mdb средствами Access пакета MS Office. Поясню сразу, что реальный положительный эффект от решения какой-либо задачи информатизации с использованием баз данных можно ощутить, если количество записей (т. е. количество строк в таблице) превышает 100 тысяч. В этом

случае очень важным (решающим) фактором оказывается скорость выборки. Однако для примера работы мы будем оперировать совсем маленькой БД.

Чтобы читатель данной книги смог реально повторить наши действия, создадим базу данных телефонов ваших контактов. Структура (поля) таблицы будет следующей: **Номер п/п**, **ФИО** и **Номер телефона**. Для этого запустим на компьютере офисное приложение Microsoft Access, далее в меню **Создать** выберем команду **Новая база данных** (или нажимаем комбинацию клавиш <Ctrl>+<N>), зададим папку для размещения БД и имя файла — vic.mdb. Затем в появившемся окне **vic: база данных** выберем команду **Создание таблицы в режиме конструктора**. Далее зададим три поля (т. е. три колонки в будущей таблице): имя первого поля — **Номер п/п**, тип данных — **Счетчик**; следующее имя поля — **ФИО**, тип данных — **Текстовый**; третье имя поля — **Номер телефона**, тип данных — **Текстовый**.

При сохранении структуры таблицы появится запрос на имя таблицы, укажем: **БД телефонов**. В БД может быть несколько таблиц, а данную таблицу мы назвали именно так. Заметьте, что при работе в обычных приложениях, если вы решили отказаться от внесенных изменений в документе, то его просто закрывают без сохранения. Однако при работе с БД все изменения сохраняются на диске без нашего ведома. Запись на диск происходит напрямую, минуя операционную систему.

Далее двойной щелчок в пределах созданной таблицы — и приступаем к ее заполнению. В нашем примере в таблице всего семь записей (рис. 10.5).

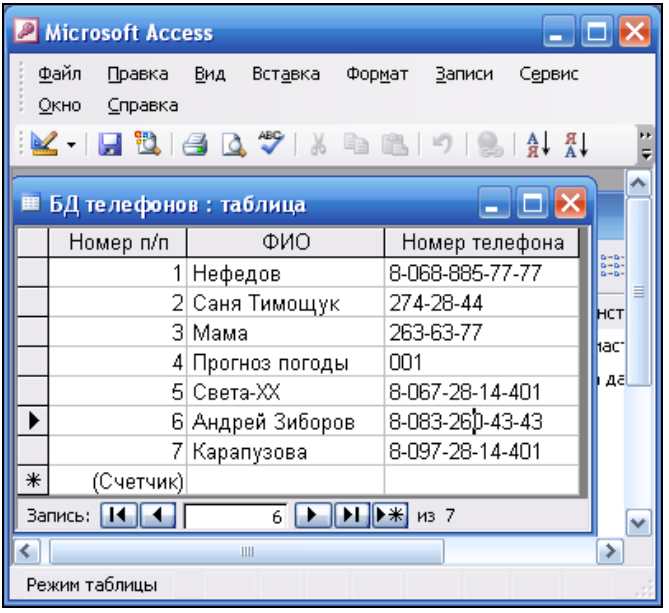


Рис. 10.5. Заполнение таблицы базы данных в среде MS Access

Теперь закроем БД Access и откроем созданную нами таблицу БД vic.mdb в среде Visual Studio 2010.

Пример 65. Редактирование таблицы базы данных MS Access в среде Visual Studio без написания программного кода

Запускаем Visual Studio 2010, однако далее мы не заказываем новый проект. Сейчас наша цель — открыть созданную нами базу данных в среде Visual Studio. Для этого выберем пункт меню **View | Server Explorer**. Здесь, щелкнув правой кнопкой мыши на значке **Data Connections**, выберем пункт **Add Connection** и укажем в качестве источника данных (**Data source**), нажав кнопку **Change**, **Microsoft Access Database File** (OLE DB). Далее с помощью кнопки **Browse** зададим путь и имя БД, например C:\vic.mdb. Теперь проверим подключение — кнопка **Test Connection**. Успешное подключение выдаст сообщение, представленное на рис. 10.6.

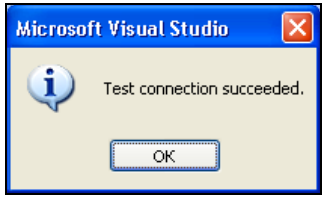


Рис. 10.6. Тестирование подключения

Проверка подключения выполнена, щелкнем на кнопке **OK**. Теперь в окне **Server Explorer** раскроем узлы, символизирующие базу данных, таблицы, поля в таблице.

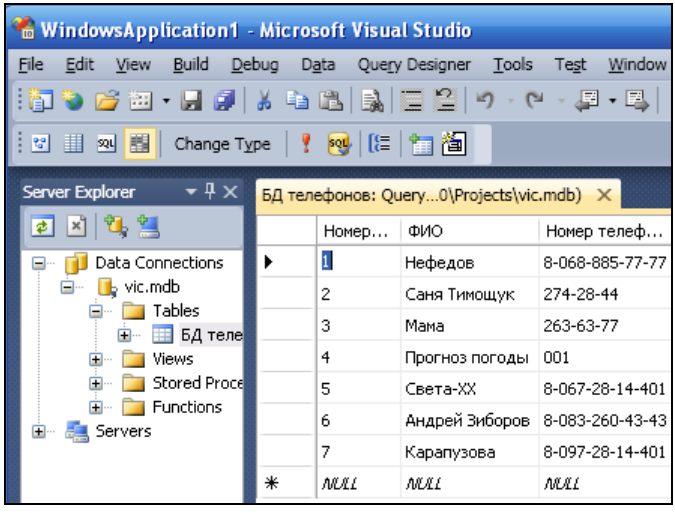


Рис. 10.7. Редактирование таблицы базы данных в среде Visual Studio

Далее щелкнем правой кнопкой мыши на узле **БД телефонов** и выберем команду **Retrieve Data**. В результате в правом окне получим содержимое этой таблицы, как показано на рис. 10.7. Здесь данную таблицу мы можем редактировать, т. е. изменять содержимое любой записи (Update), добавлять новые записи (Insert), т. е. новые строки в таблицу, удалять записи (Delete). Кроме того, щелкая правой кнопкой мыши в пределах таблицы и выбирая в контекстном меню пункты **Pane | SQL**, можно осуществлять SQL-запросы к базе данных, в том числе наиболее часто используемый запрос `Select`.

Пример 66. Отображение таблицы базы данных MS Access в экранной форме

Поставим задачу вывода таблицы базы данных MS Access в экранную форму с возможностью редактирования базы данных, как мы это сделали для базы данных SQL Server. Разница будет состоять лишь в том, что мы не будем копировать базу данных в наш проект, а подключимся к существующей базе данных. Сразу скажу, что данная задача решается аналогично.

Запустим Visual Studio 2010 и закажем новый проект шаблона **Windows Forms Application C#**, укажем имя проекта — БД_mdb. Откроется новый проект **Windows Forms**. В пункте меню **Data** (Данные) щелкнем на пункте **Add New Data Source**, при этом запустится **Data Source Configuration Wizard** (Мастер настройки источников данных). Выберем **Database** (База данных) и нажмем кнопку **Next**. Далее после выбора модели **Dataset** и нажатия кнопки **Next** нажмем кнопку **New Connection** (Новое подключение).

Откроется диалоговое окно **Add Connection** (Добавить подключение). В этом диалоговом окне проверим, является ли источник данных (**Data source**) **Microsoft Access Database File** (OLE DB). Если нет, то нажмем кнопку **Change** и выберем **Microsoft Access Database File** в диалоговом окне **Change Data Source**, а затем нажмем кнопку **OK**.

Щелкнув на кнопке **Browse**, перейдем в место сохранения базы данных vic.mdb, далее щелкнем на значке файла базы данных и на кнопке **Открыть**. Нажмем кнопку **OK**, чтобы закрыть диалоговое окно **Add Connection**, затем в мастере настройки источников данных нажмем кнопку **Next**. Если будет предложено копировать файл данных в проект, нажмем кнопку **Нет**.

Опять нажмем кнопку **Next**. На следующей странице мастера необходимо установить флажок возле всех объектов базы данных, при этом имя объекта **DataSet** будет **vicDataSet**. Нажмем кнопку **Finish** для продолжения. Перейдя в меню **Data | Show Data Sources**, получим в окне **Data Sources** узлы объектов базы, показанные на рис. 10.8.

Теперь перетащим узел **БД телефонов** из окна **Data Source** в проект экранной формы, при этом можно использовать метод копирования — вставки (<Ctrl>+<C>, <Ctrl>+<V>). Заметьте, что некоторые элементы управления автоматически доба-

вятся в форму, также будут созданы некоторые компоненты и добавлены в область компонентов под формой. Появится элемент управления **DataGridView**, который будет отображать строки и столбцы таблицы, и элемент управления для переходов (**AddressesBindingNavigator**). Кроме того, система создает компоненты, которые подключаются к базе данных, управляют извлечением и обновлением данных и хранят данные в локальном объекте **DataSet** (**AddressesBindingSource**, **AddressesTableAdapter** и **FirstDatabaseDataSet** соответственно).

Нажмем клавишу <F5> для запуска приложения. Данные из таблицы телефонов отображаются в элементе управления **DataGridView** в форме, как показано на рис. 10.9.

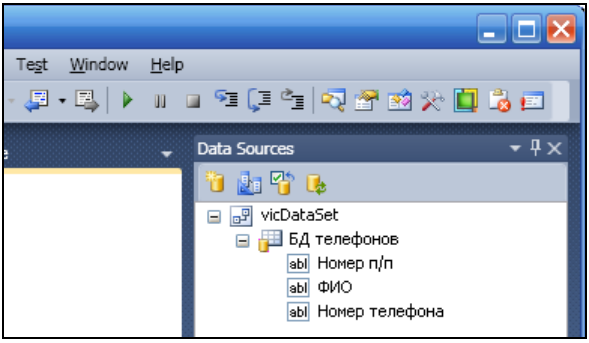


Рис. 10.8. Узлы объектов базы данных

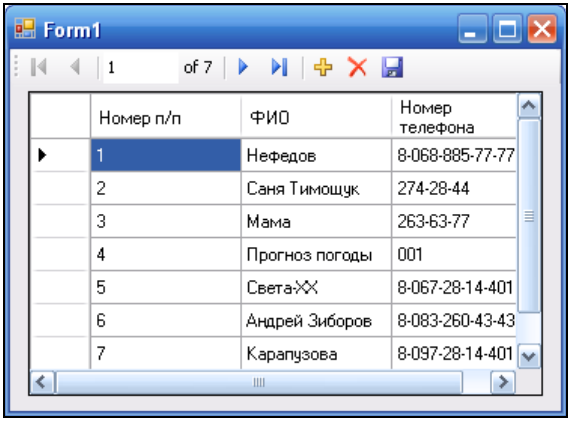


Рис. 10.9. Отображение таблицы на элементе управления **DataGridView**

Можно использовать элементы управления в **BindingNavigator** в верхней части формы для перехода между строками, можно даже добавлять или удалять записи. А также изменять записи, изменяя данные, отображаемые в сетке, но эти изменения не будут сохранены до тех пор, пока не будет нажата кнопка сохранения (**Save Data**).

Как видите, мы не написали ни одной строчки программного кода. Среда Visual Studio 2010 автоматически сгенерировала программный код для управления представляемой программой. Этот программный код можно просмотреть, открыв файл проекта Form1.Designer.cs.

Убедиться в работоспособности программы можно, открыв файл решения БД_mdb.sln в папке БД_mdb.

Пример 67. Чтение всех записей из таблицы БД MS Access на консоль с помощью объектов классов *Command* и *DataReader*

Напишем программу, которая минимальным количеством строк программного кода выводит на экран все записи (т. е. все строки) таблицы базы данных. При этом воспользуемся наиболее современной технологией ADO.NET. Технология ADO.NET — это преимущественно 4 объекта. Объект `Connection` обеспечивает соединение с базой данных. Объект `Command` обеспечивает привязку SQL-выражения к соединению с базой данных. А с помощью объектов `DataSet` и `DataReaders` можно просмотреть результаты запроса.

Здесь и далее мы рассмотрим четыре основных действия над базой данных: `Select` (выборка записей из таблицы БД), `Insert` (вставка записей), `Update` (модификация записей в таблице БД), `Delete` (удаление некоторых записей из таблицы).

Для этого запустим Visual Studio 2010 и щелкнем на пункте **New Project**. Форма нам в данном случае не нужна, поэтому выберем шаблон **Console Application C#**. Мы хотим вывести на экран простейшим способом таблицу. Если мы будем выводить ее при помощи функции `MessageBox.Show`, то ровных колонок в окне `MessageBox.Show` мы не получим, поскольку буквы, реализованные в этой функции, имеют разную ширину. Обычно в этом случае для вывода таблиц используют шрифт Courier New или Consolas, но объект `MessageBox` не содержит возможностей смены шрифта. Поэтому, идя по самому короткому пути, выводим таблицу из базы данных на консоль, т. е. на черный экран DOS. Здесь реализован моноширинный шрифт, где все символы имеют одинаковую ширину. Например, буква "Ш" и символ "." (точка) имеют одинаковую ширину, следовательно, колонки в построенных таблицах будут ровными.

Теперь на вкладке **Program.cs** напишем текст программы из листинга 10.1.

Листинг 10.1. Чтение всех записей из таблицы БД MS Access и вывод их на консоль

```
// Программа читает все записи из таблицы БД MS Access и выводит их
// на консоль с помощью объектов Command и DataReader
using System;
// Добавляем эту директиву для краткости выражений:
using ОлеДиБи = System.Data.OleDb;
```

```
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace БДDataReader1
{
   class Program
   {
      static void Main(string[] args)
      {  // Создаем объект Connection и передаем ему строку подключения
         var Подключение = new ОлеДиБи.
               OleDbConnection("Data Source=\"C:\\vic.mdb\";User " +
                     "ID=Admin;Provider=\"Microsoft.Jet.OLEDB.4.0\";");
         Подключение.Open();
         // Создаем объект класса Command, передавая ему SQL-команду
         var Команда = new ОлеДиБи.OleDbCommand(
                        "Select * From [БД телефонов]", Подключение);
         // Выбрать все записи и сортировать их по колонке "ФИО":
         // var Команда = new ОлеДиБи.OleDbCommand(
         //    "Select * From [БД телефонов] order by ФИО", Подключение);
         // Аналогично по колонке "Номер п/п":
         // var Команда = new ОлеДиБи.OleDbCommand(
         //    "Select * From [БД телефонов] ORDER BY 'Номер п/п'",
         //                                 Подключение);
         // Выполняем SQL-команду:
         ОлеДиБи.OleDbDataReader Читатель = Команда.
              ExecuteReader(System.Data.CommandBehavior.CloseConnection);
         Console.WriteLine("Таблица БД:\n");
         while (Читатель.Read() == true)
            // Цикл, пока не будут прочитаны все записи.
            // Читатель.FieldCount - кол-во полей в строке.
            // Здесь три поля: 0, 1 и 2.
            // Минус прижимает строку влево:
            Console.WriteLine("{0,-3} {1,-15} {2,-15}", Читатель.GetValue(0),
                           Читатель.GetValue(1), Читатель.GetValue(2));
         Читатель.Close(); Подключение.Close();
         // Приостановить выполнение программы до нажатия какой-нибудь клавиши:
         Console.ReadKey();
      }
   }
}
```

Как видно, вначале создаем объект `Подключение` класса `Connection` и передаем ему строку подключения. В строке подключения полный доступ к mdb-файлу за-

ключен в двойные кавычки. Это сделано для того, чтобы корректно читались длинные имена папок и файлов, содержащие пробелы.

Далее создаем объект класса `Command` и передаем ему простейшую SQL-команду:

```
Select * From [БД телефонов]
```

То есть выбрать *все записи* из таблицы `[БД телефонов]`. Название таблицы в SQL-запросе заключено в квадратные скобки из-за пробела в имени таблицы. Заметьте, что в комментарии указаны возможные варианты SQL-запроса: *сортировать* записи по колонке **ФИО** (`ORDER BY ФИО`) и по колонке **Номер п/п** (`ORDER BY 'Номер п/п'`).

Затем, используя объект класса `DataReader`, выполняем SQL-команду. Далее в цикле построчно читаем таблицу базы данных. При работе `DataReader` в памяти хранится только одна строка (запись) данных. Объект класса `DataReader` имеет булеву функцию `Read`, которая возвращает `true`, если существует следующая строка данных, и `false`, если такие строки (записи) уже исчерпались. Причем с помощью `DataReader` невозможно заранее узнать количество записей в таблице.

Результат работы программы показан на рис. 10.10.

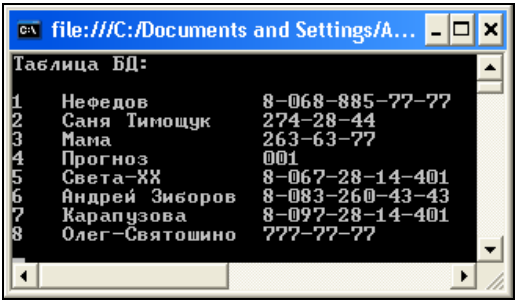


Рис. 10.10. Отображение таблицы базы данных на консоли

Таким образом, мы получили простейшую программу для просмотра таблицы базы данных. С ее помощью можно *только просматривать данные*, но нельзя их редактировать.

Убедиться в работоспособности программы можно, открыв решение БДDataReader1.sln в папке БДDataReader1.

Пример 68. Создание базы данных MS Access в программном коде

Создадим программу, которая во время своей работы создает базу данных Access, т. е. файл new_BD.mdb. Эта база данных будет пустой, т. е. она не будет содержать ни одной таблицы. Наполнять базу данных таблицами можно впослед-

ствии как из программного кода Visual C# 2010, так и используя MS Access. Заметим, что в этом примере технология ADO.NET *не использована*.

Запускаем Visual Studio 2010 (загрузочный модуль devenv.exe), щелкаем пункт меню **New Project** и выбираем шаблон **Console Application C#**. Для добавления в наш проект DLL-библиотеки ADOX выполним следующее. В пункте меню **Project** выберем команду **Add Reference** , затем на вкладке **COM** дважды щелкнем по ссылке **Microsoft ADO Ext. 2.8 for DDL and Security**, тем самым мы добавляем эту библиотеку в текущий проект. Убедиться в том, что теперь существует ссылка на эту библиотеку, можно в окне **Solution Explorer**. Здесь, щелкнув на узле **References**, увидим ветвь **ADOX**. Теперь мы можем ссылаться на это имя в программном коде. Далее вводим программный код, приведенный в листинге 10.2.

Листинг 10.2. Создание БД во время работы программы

```
// Программа создает базу данных MS Access, т. е. файл new_BD.mdb.
// Эта база данных будет пустой, т. е. не будет содержать ни одной таблицы.
// Наполнять базу данных таблицами можно будет впоследствии
// как из программного кода C# 2010, так и используя MS Access.
// В этом примере технология ADO.NET не использована
using System.Windows.Forms; // - добавили эту директиву для MessageBox
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace БДСоздание
{  // Для добавления ADOX: Project | Add Reference, на вкладке COM выбрать
   // Microsoft ADO Ext. 2.8 for DDL and Security
   class Program
   {
      static void Main(string[] args)
      {
         ADOX.Catalog Каталог = new ADOX.Catalog();
         try
         {
            Каталог.Create("Provider=Microsoft.Jet." +
                           "OLEDB.4.0;Data Source=C:\\new_BD.mdb");
            MessageBox.Show("База данных C:\\new_BD.mdb успешно создана");
         }
         catch (System.Runtime.InteropServices.COMException Ex)
         { MessageBox.Show(Ex.Message); }
         finally
         { Каталог = null; }
      }
   }
}
```

Далее чтобы во время выполнения программы не появлялось черное окно DOS, в пунктах меню **Project | ConsoleApplication1 Properties** на вкладке **Application** в раскрывающемся списке **Application type** выберем **Windows Forms Application**. Чтобы был доступен объект `MessageBox` для вывода сообщений, добавим в проект еще одну DLL-библиотеку. Для этого, как и в предыдущем случае, укажем пункты меню **Project | Add Reference** и на вкладке **.NET** дважды щелкнем по ссылке **System.Windows.Forms.dll**, а в тексте программы добавим директиву:

```
using System.Windows.Forms;
```

Ключевое слово `using` используется для импортирования пространства имен, которое содержит класс `MessageBox`.

Программа работает следующим образом: создаем экземпляр класса `ADOX.catalog`, одна из его функций `Create` способна создавать базу данных, если на ее вход подать строку подключения. Заметим, что в строку подключения входит также и *полный путь* к создаваемой БД. Функция `Create` заключена в блоки `try...catch`, которые обрабатывают *исключительные ситуации*. После запуска этого приложения получим сообщение о создании базы данных (рис. 10.11).

Если же тут же запустим наше приложение еще раз, то получим сообщение о том, что такая база данных уже существует (рис. 10.12), поскольку БД new_BD.mdb только что создана.

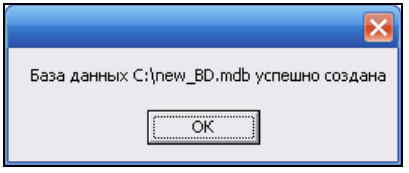


Рис. 10.11. Сообщение о создании базы данных

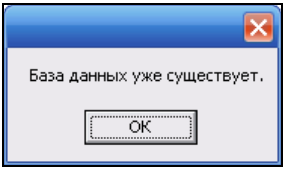


Рис. 10.12. База данных существует

Данное сообщение генерировалось обработчиком исключительной ситуации.

Программный код этой программы можно посмотреть, открыв решение БДСоздание.sln в папке БДСоздание.

Пример 69. Запись структуры таблицы в пустую базу данных MS Access. Программная реализация подключения к БД

Теперь здесь и далее мы используем только самую современную технологию ADO.NET. Создадим программу, которая записывает структуру таблицы, т. е. "шапку" таблицы, в существующую БД. В этой БД может еще не быть ни одной

таблицы, т. е. БД может быть пустой. Либо в БД могут уже быть таблицы, но название новой таблицы должно быть уникальным.

Создадим базу данных new_BD.mdb в корневом каталоге логического диска C:, используя MS Access или программным путем, как это было показано в предыдущем разделе. Никакие таблицы в базе данных создавать не станем, т. е. будем иметь пустую БД. Теперь запустим Visual Studio 2010, затем выберем пункт **New Project**. Далее поскольку для этого приложения нам не нужна экранная форма с ее элементами управления, укажем среди шаблонов **Console Application C#**. Затем напишем программный код, представленный в листинге 10.3.

Листинг 10.3. Создание таблицы в БД MS Access

```
// Программа записывает структуру таблицы в пустую базу данных MS Access.
// Программная реализация подключения к БД. В этой БД может
// еще не быть ни одной таблицы, т. е. БД может быть пустой. Либо в БД
// могут уже быть таблицы, но название новой таблицы должно быть уникальным
using System;
using System.Windows.Forms; // - добавили эту директиву для MessageBox
// Добавляем эту директиву для краткости выражений:
using ОлеДиБи = System.Data.OleDb;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace БдСоздТаблицы
{  // ЗАПИСЬ СТРУКТУРЫ ТАБЛИЦЫ В ПУСТУЮ БД:
   class Program
   {
      static void Main(string[] args)
      {  // Создание экземпляра объекта Connection с указанием строки
         // подключения:
         var Подключение = new ОлеДиБи.OleDbConnection(
            "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\\new_BD.mdb");
         // Открытие подключения:
         Подключение.Open();
         // Создание экземпляра объекта класса Command
         // с заданием SQL-запроса:
         var Команда = new ОлеДиБи.OleDbCommand("CREATE TABLE [" +
                       "БД телефонов] ([Номер п/п] counter, [ФИО] ch" +
                       "ar(20), [Номер телефона] char(20))", Подключение);
         try // Выполнение команды SQL:
         {
            Команда.ExecuteNonQuery();
            MessageBox.
```

```
                Show("Структура таблицы 'БД телефонов' записана в пустую БД");
            }
            catch (Exception Ситуация)
            { MessageBox.Show(Ситуация.Message); }
            Подключение.Close();
        }
    }
}
```

Чтобы работала функция `MessageBox.Show`, следует в текущий проект добавить ссылку на DLL-библиотеку. Для этого в пункте меню **Project** выберем команду **Add Reference** и на вкладке **.NET** дважды щелкнем на ссылке **System.Windows.Forms.dll**. Кроме того, используя пункты меню **Project | ConsoleApplication1 Properties**, на появившейся вкладке **Application** в раскрывающемся списке **Application type** выберем тип приложения **Windows Forms Application**.

Как видно из текста программы, вначале создаем экземпляр класса `Connection` с указанием строки подключения. Теперь мы имеем возможность управлять этой строкой *программно*. Далее создаем экземпляр класса `Command` с заданием SQL-запроса. В этом запросе создаем (`CREATE`) новую таблицу с именем **БД телефонов** с тремя полями: **Номер п/п** типа счетчик (`counter`), **ФИО** и **Номер телефона**. Здесь имя таблицы и имена полей заключены в квадратные скобки, поскольку они содержат пробелы.

Чтобы выполнить эту SQL-команду, вызываем метод `ExecuteNonQuery`, который заключим в блоки `try...catch` для обработки исключительных ситуаций. Если SQL-запрос благополучно выполнился, то получаем сообщение: "Структура таблицы 'БД телефонов' записана в пустую БД". А если, например, таблица с таким именем уже имеется в базе данных, то управление передается блоку `catch` (перехват исключительной ситуации), и мы получаем сообщение о том, что такая таблица базы данных уже существует (рис. 10.13).

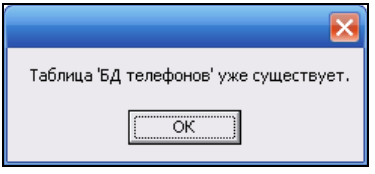


Рис. 10.13. Сообщение о существовании таблицы

Таким образом, в данной программе сначала организовано подключение `Connection` к БД через строку подключения и открытие подключения `Open`. Затем задание SQL-запроса в объекте `Command` и выполнение запроса функцией `ExecuteNonQuery`. Если связывание данных организовать программно, то добиваемся более высокой гибкости, когда, например, на стадии разработки неизвестно заранее, где (на каком диске, в какой папке) будет находиться БД.

Убедиться в работоспособности программы можно, открыв решение БдСоздТаблицы.sln в папке БдСоздТаблицы.

Пример 70. Добавление записей в таблицу базы данных MS Access

Совсем маленькую программу из предыдущего раздела можно использовать для выполнения *любого запроса* к базе данных. Например, модифицируем всего лишь одну строчку программного кода программы из предыдущего примера *для добавления новой записи* в таблицу БД. Для этого при создании экземпляра объекта `Command` зададим SQL-запрос на вставку (`INSERT`) новой записи в таблицу БД.

Заметим, что в SQL-запросе мы сознательно обратились к таблице по имени `[бд телефонов]`, т. е. со строчной буквы, хотя надо с прописной. Дело в том, что в именах таблиц следует *точно* указывать регистр символа, поскольку их поиск ведется с учетом регистра (case-sensitive search). Однако это не обязательно при наличии только одной таблицы с таким именем, поскольку при этом используется поиск без учета регистра (case-insensitive search).

Свойству `Connection` объекта класса `Command` следует дать ссылку на объект класса `Connection`:

```
Команда.Connection = Подключение;
```

Причем для добавления записи в таблицу БД такая ссылка обязательна в отличие от предыдущего примера, где мы создавали новую таблицу в существующей БД.

Программный код будет выглядеть так, как представлено в листинге 10.4.

Листинг 10.4. Добавление записей в таблицу базы данных MS Access

```
// Программа добавляет запись в таблицу базы данных MS Access. Для этого
// при создании экземпляра объекта Command задаем SQL-запрос
// на вставку (Insert) новой записи в таблицу базы данных
using System.Windows.Forms; // - добавили эту директиву для MessageBox
// Добавляем эту директиву для краткости выражений:
using ОлеДиБи = System.Data.OleDb;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace БдДобавлЗаписи
{
   class Program
   {  // ДОБАВЛЕНИЕ ЗАПИСИ В ТАБЛИЦУ БД:
      static void Main(string[] args)
      {  // Создание экземпляра объекта Connection
         // с указанием строки подключения:
         var Подключение = new ОлеДиБи.OleDbConnection(
```

```
            "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\\new_BD.mdb");
        // Открытие подключения:
        Подключение.Open();
        // Создание экземпляра объекта Command с заданием SQL-запроса:
        var Команда = new ОлеДиБи.OleDbCommand(
                "INSERT INTO [бд телефонов] (" +
                "Фио, [номер телефона]) VALUES ('Света-Х', '521-61-41')");
        // Для добавления записи в таблицу БД эта команда обязательна:
        Команда.Connection = Подключение;
        // Выполнение команды SQL:
        Команда.ExecuteNonQuery();
        MessageBox.Show("В таблицу 'БД телефонов' добавлена запись");
        Подключение.Close();
    }
  }
}
```

Часто, отлаживая программный код на Visual Studio C#, при работе с БД появляется необходимость *проверки работы* программы, например, создалась ли таблица в БД, добавилась ли запись в таблице БД, правильно ли сформирован SQL-запрос. Не обязательно запускать MS Access, чтобы выполнить SQL-запрос или проверить правильность его синтаксиса. Это можно сделать в среде Visual Studio. Для этого в пункте меню **View** выбираем команду **Server Explorer** (Обозреватель баз данных), далее в списке подключений указываем полный путь к нужной БД. Затем, щелкая правой кнопкой мыши на значке нужной таблицы, в контекстном меню выбираем пункт **Retrieve Data**. При этом в панели инструментов (**Toolbar**) появляется значок **SQL**, после щелчка по этому значку (или нажатия комбинации клавиш <Ctrl>+<3>) получим окно SQL-запроса. В этом окне мы можем задавать SQL-запрос, а затем, например, щелкая правой кнопкой мыши, либо проверять его синтаксис, либо выполнять.

Убедиться в работоспособности программы можно, открыв решение БдДобавлЗаписи.sln в папке БдДобавлЗаписи.

Пример 71. Чтение всех записей из таблицы базы данных с помощью объектов классов *Command*, *DataReader* и элемента управления *DataGridView*

Покажем, как легко, очень малой кровью, можно вывести таблицу базы данных на элемент управления **DataGridView** (сетка данных, т. е. таблица данных) с ис-

пользованием тех же объектов из предыдущей программы классов `Command` и `DataReader`.

Для решения этой задачи запустим Visual Studio 2010, закажем новый проект **New Project** шаблона **Windows Forms Application C#**. Из панели **Toolbox** добавим в проектируемую форму элемент управления **DataGridView** и растянем его на всю форму. На вкладке **Form1.cs** напишем программный код, представленный в листинге 10.5.

Листинг 10.5. Чтение всех записей из таблицы БД

```
// Программа читает все записи из таблицы базы данных с помощью объектов
// Command, DataReader и элемента управления DataGridView
using System;
using System.Data;
using System.Windows.Forms;
// Добавляем эту директиву для краткости выражений:
using ОлеДиБи = System.Data.OleDb;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace БдReaderGridView
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
         // Создаем объект Connection и передаем ему строку подключения:
         var Подключение = new ОлеДиБи.OleDbConnection(
                   "Data Source=\"C:\\vic.mdb\";User " +
                   "ID=Admin;Provider=\"Microsoft.Jet.OLEDB.4.0\";");
         Подключение.Open();
         // Создаем объект Command, передавая ему SQL-команду
         ОлеДиБи.OleDbCommand Команда = new ОлеДиБи.
               OleDbCommand("Select * From [БД телефонов]", Подключение);
         // Выполняем SQL-команду
         ОлеДиБи.OleDbDataReader Читатель = Команда.
               ExecuteReader(); // (CommandBehavior.CloseConnection)
         DataTable Таблица = new DataTable();
         // Заполнение "шапки" таблицы
         Таблица.Columns.Add(Читатель.GetName(0));
         Таблица.Columns.Add(Читатель.GetName(1));
         Таблица.Columns.Add(Читатель.GetName(2));
         while (Читатель.Read() == true)
```

```
            // Заполнение клеток (ячеек) таблицы
            Таблица.Rows.Add(new object[] {Читатель.GetValue(0),
                    Читатель.GetValue(1), Читатель.GetValue(2)});
        // Здесь три поля: 0, 1 и 2
        Читатель.Close(); Подключение.Close();
        dataGridView1.DataSource = Таблица;
    }
  }
}
```

Как видно из программы, она очень похожа на предыдущую. После выполнения SQL-команды создаем объект `DataTable`, который в конце программного кода задаем как источник (`DataSource`) для сетки данных `dataGridView1`. Заполняем "шапку" таблицы, т. е. названия колонок, методом `Add`.

Далее, как и в предыдущей программе, в цикле `While` заполняем ячейки таблицы. Фрагмент работы программы показан на рис. 10.14.

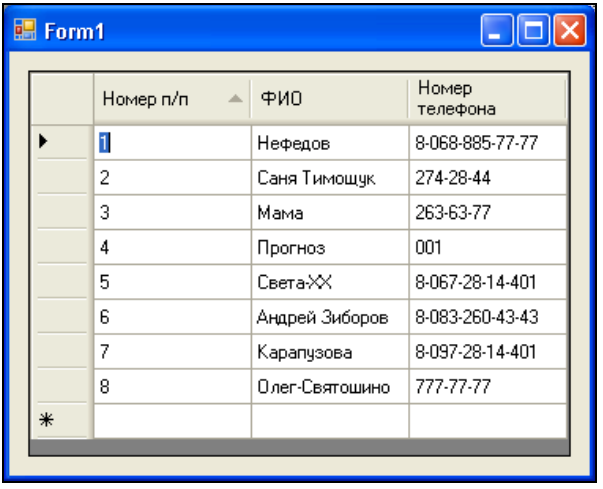


Рис. 10.14. Отображение таблицы базы данных на элементе **DataGridView**

В этой таблице мы можем сортировать записи по любой из колонок, щелкая мышью на названиях соответствующих колонок. Можем редактировать (изменять) содержание ячеек, но в базу данных эти изменения не попадут (сохранения не будет).

Одно из ключевых преимуществ использования объекта `DataReader` — это его скорость и небольшое использование памяти. Однако применение циклического считывания данных сводит эти преимущества на нет.

Убедиться в работоспособности программы можно, открыв решение БдReaderGridView.sln в папке БдReaderGridView.

Пример 72. Чтение данных из БД в сетку данных *DataGridView* с использованием объектов классов *Command*, *Adapter* и *DataSet*

Рассмотрим пример чтения таблицы с помощью объекта `Adapter` из базы данных посредством выбора нужных данных и передачи их объекту `DataSet`. Очень удобно прочитать таблицу, записанную в `DataSet`, используя элемент управления **DataGridView** (сетка данных, т. е. таблица данных), указав в качестве источника данных для сетки **DataGridView** объект класса `DataSet`.

Поскольку нам нужен элемент управления **DataGridView**, мы создаем новый проект с экранной формой. Для этого, как обычно, запускаем Visual Studio 2010, заказываем новый проект **New Project** шаблона **Windows Forms Application C#**. Из панели **Toolbox** добавляем в форму элемент управления **DataGridView** и растягиваем его на всю форму, как показано на рис. 10.15.

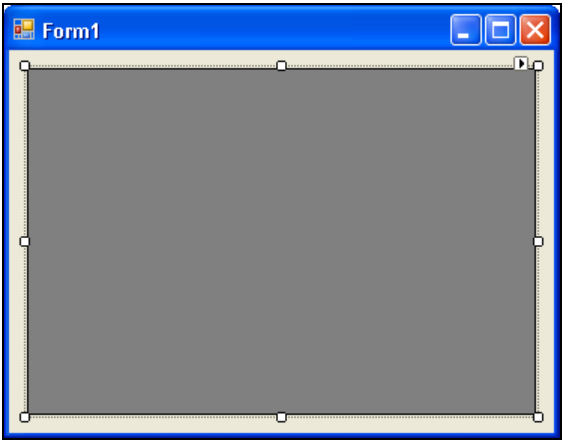


Рис. 10.15. Проектирование экранной формы

Далее пишем программный код, представленный в листинге 10.6.

Листинг 10.6. Чтение данных из БД в сетку данных *DataGridView*

```
// Программа читает из БД таблицу в сетку данных DataGridView
// с использованием объектов класса Command, Adapter и DataSet
using System.Data;
using System.Windows.Forms;
// Добавляем эту директиву для краткости выражений:
using ОлеДиБи = System.Data.OleDb;
```

```
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace БдАдаптерGridView
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
         this.Text = "Чтение таблицы из БД:";
         var Подключение = new ОлеДиБи.OleDbConnection(
                        "Data Source=\"C:\\vic.mdb\";User " +
                        "ID=Admin;Provider=\"Microsoft.Jet.OLEDB.4.0\";");
         Подключение.Open();
         var Команда = new ОлеДиБи.OleDbCommand(
                        "Select * From [БД телефонов]", Подключение);
         // var Команда = new ОлеДиБи.OleDbCommand("SELECT * FRO" +
         //          "M [БД телефонов] WHERE (фио LIKE 'м%')", Подключение);
         // Создаем объект Adapter и выполняем SQL-запрос
         var Адаптер = new ОлеДиБи.OleDbDataAdapter(Команда);
         // Создаем объект класса DataSet
         DataSet НаборДанных = new DataSet();
         // Заполняем DataSet результатом SQL-запроса
         Адаптер.Fill(НаборДанных, "БД телефонов");
         // Содержимое DataSet в виде строки XML для отладки:
         string СтрокаXML = НаборДанных.GetXml();
         // Указываем источник данных для сетки данных:
         dataGridView1.DataSource = НаборДанных;
         // Указываем имя таблицы в наборе данных:
         dataGridView1.DataMember = "БД телефонов";
         Подключение.Close();
      }
   }
}
```

Как видно из текста программы, вначале создаем объект класса `Connection`, передавая строку подключения. Затем, создавая объект класса `Command`, задаем SQL-команду выбора всех записей из таблицы **БД телефонов**. Здесь мы можем задать любую SQL-команду, в комментарии приведен пример такой команды, которая содержит `SELECT` и `LIKE`: выбрать из таблицы **БД телефонов** только записи, в которых поле **ФИО** начинается на "м". Оператор `LIKE` используется для поиска по шаблону (pattern matching) вместе с символами универсальной подстановки (мета-

символами) "звездочка" (*) и "знак вопроса" (?). Строка шаблона заключена в одинарные кавычки. Заметим также, что большинство баз данных использует символ "%" вместо значка "*" в LIKE-выражениях.

Далее при создании объекта класса Adapter выполняем SQL-команду и при выполнении метода Fill заполняем объект класса DataSet таблицей, полученной в результате SQL-запроса. Затем указываем в качестве источника данных для сетки данных dataGridView1 объект класса DataSet. Этого оказывается достаточно для вывода на экран результатов SQL-запроса (рис. 10.16).

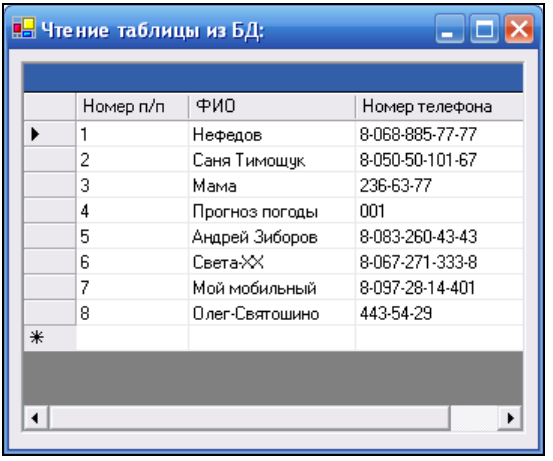


Рис. 10.16. Вывод результата SQL-запроса

Так же как и при использовании объекта класса DataReader в предыдущем примере, в полученной таблице мы можем сортировать записи по любой из колонок. Можем редактировать (изменять) содержание ячеек, но в базу данных эти изменения не попадут (сохранения не будет).

Заметим, что здесь с помощью визуального проектирования выполнено только перетаскивание в форму сетки данных **DataGridView**, остальное сделано программно, что обеспечивает большую гибкость программы.

Убедиться в работоспособности программы можно, открыв решение БдАдаптерGridView.sln в папке БдАдаптерGridView.

Пример 73. Обновление записей в таблице базы данных MS Access

Одним из основных четырех действий над данными в БД (Select, Insert, Update и Delete) является модификация (Update, обновление) данных. Автор поставил задачу написать маленькую программу для обновления записей в таблице базы данных, но с бóльшим удобством (гибкостью) управления программным кодом.

Рассматриваемая в данном примере программа имеет форму, сетку данных **DataGridView**, в которую из базы данных читается таблица при нажатии кнопки **Читать из БД**. Пользователь имеет возможность *редактировать* данные в этой таблице, после чего, нажав кнопку **Сохранить в БД**, данные в базе данных *будут модифицированы*, т. е. заменены новыми.

Для написания этой программы запускаем Visual Studio 2010, заказываем новый проект **New Project** шаблона **Windows Forms Application C#**. Из панели **Toolbox** добавляем в форму элемент управления **DataGridView** и две командные кнопки. Программный код программы представлен в листинге 10.7.

Листинг 10.7. Обновление записей в таблице базы данных MS Access

```
// Программа обновляет записи (Update) в таблице базы данных MS Access
using System;
using System.Data;
using System.Windows.Forms;
// Добавляем эту директиву для краткости выражений:
using ОлеДиБи = System.Data.OleDb;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace БдUpdate
{
   public partial class Form1 : Form
   {
      DataSet НаборДанных;
      ОлеДиБи.OleDbDataAdapter Адаптер;
      ОлеДиБи.OleDbConnection Подключение = new ОлеДиБи.
            OleDbConnection( // Строка подключения:
                     "Data Source=\"C:\\vic.mdb\";User " +
                     "ID=Admin;Provider=\"Microsoft.Jet.OLEDB.4.0\";");
      ОлеДиБи.OleDbCommand Команда = new ОлеДиБи.OleDbCommand();
      public Form1()
      {
         InitializeComponent();
         button1.Text = "Читать из БД"; button1.TabIndex = 0;
         button2.Text = "Сохранить в БД";
      }
      private void button1_Click(object sender, System.EventArgs e)
      {  // Читать из БД:
         НаборДанных = new DataSet();
         if (Подключение.State == ConnectionState.Closed) Подключение.Open();
         Адаптер = new ОлеДиБи.OleDbDataAdapter(
                        "Select * From [БД телефонов]", Подключение);
```

```
        // Заполняем DataSet результатом SQL-запроса
        Адаптер.Fill(НаборДанных, "БД телефонов");
        // Содержимое DataSet в виде строки XML для отладки:
        string СтрокаXML = НаборДанных.GetXml();
        // Указываем источник данных для сетки данных:
        dataGridView1.DataSource = НаборДанных;
        // Указываем имя таблицы в наборе данных:
        dataGridView1.DataMember = "БД телефонов";
        Подключение.Close();
     }
     private void button2_Click(object sender, System.EventArgs e)
     {  // Сохранить в базе данных
        Команда.CommandText = "UPDATE [БД телефонов] SET [Но" +
                  "мер телефона] = ?, ФИО = ?  WHERE ([Номер п/п] = ?)";
        // Имя, тип и длина параметра
        Команда.Parameters.Add("Номер телефона", ОлеДиБи.OleDbType.VarWChar,
                               50, "Номер телефона");
        Команда.Parameters.Add("ФИО", ОлеДиБи.OleDbType.VarWChar, 50, "ФИО");
        Команда.Parameters.Add
           (new ОлеДиБи.OleDbParameter("Original_Номер_п_п",
                            ОлеДиБи.OleDbType.Integer,
                            0, System.Data.ParameterDirection.
                            Input, false, (byte)0, (byte)0, "Номер п/п",
                            System.Data.DataRowVersion.Original, null));
        Адаптер.UpdateCommand = Команда;
        Команда.Connection = Подключение;
        try
        {  // Update возвращает количество измененных строк
           int kol = Адаптер.Update(НаборДанных, "БД телефонов");
           MessageBox.Show("Обновлено " + kol + " записей");
        }
        catch (Exception Ситуация)
        { MessageBox.Show(Ситуация.Message, "Недоразумение"); }
     }
  }
}
```

Как видно, мы имеем две процедуры обработки событий: щелчок на кнопке **Читать из БД** и щелчок на кнопке **Сохранить в БД**. Чтобы объекты классов `DataSet`, `DataAdapter`, `Connection` и `Command` были видны в этих трех процедурах, объявляем эти объекты внешними внутри класса `Form1`.

При программировании чтения из базы данных вначале с помощью SQL-запроса мы выбрали все записи из таблицы (`Select * From [БД телефонов]`) и с помощью объекта класса `Adapter` поместили в набор данных `DataSet`. А затем указали объект класса `DataSet` в качестве источника (`DataSource`) для сетки данных `dataGridView1`. Фрагмент работы программы после чтения из базы данных представлен на рис. 10.17.

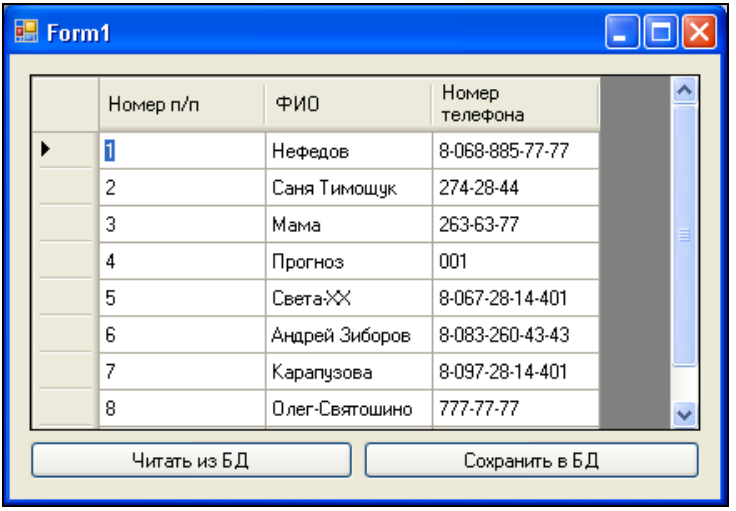


Рис. 10.17. Фрагмент работы программы обновления данных

Представляет интерес программирование *модификации* записей базы данных. Эта возможность запрограммирована при обработке события "щелчок мышью на кнопке" **Сохранить в БД**. Здесь свойству `CommandText` присвоено значение текста SQL-запроса. В качестве заменителей параметров используются вопросительные знаки. Как видно, в данном SQL-запросе имеют место три вопросительных знака. Им соответствуют три параметра, которые должны указываться именно в порядке следования вопросительных знаков. Эти параметры задаем с использованием метода `COMMAND.Parameters.Add`. Здесь указываем имя поля (например, `"Номер телефона"`), тип, длину параметра и значение по умолчанию. Заметим, что третий параметр (`"Номер п/п"`) задается как новый, поскольку он не должен подлежать редактированию со стороны пользователя, а устанавливаться автоматически, независимо от пользователя.

Далее в блоке `try...catch` вызываем непосредственно метод `Update`, который возвращает количество (`kol`) *обновленных* записей. В случае неудачного обновления обрабатывается исключительная ситуация `Exception`, объект `Exception` обеспечивает соответствующее сообщение о недоразумении.

Убедиться в работоспособности программы можно, открыв решение БдUpdate.sln в папке БдUpdate.

Пример 74. Удаление записей из таблицы базы данных с использованием SQL-запроса и объекта класса *Command*

Можно также удалять записи (строки из таблицы БД), формируя в программном коде соответствующий SQL-запрос, передаваемый в объект класса `Command`. Именно объект `Command` — обеспечивает привязку SQL-выражения к соединению с базой данных. Напишем самый простой пример такой программы.

Для этого запустим Visual Studio 2010 и закажем новый проект **New Project**. В данном случае экранная форма нам не нужна, поэтому выберем, как и в некоторых предыдущих примерах, шаблон **Console Application C#**. Для того чтобы во время выполнения данной программы не видеть черного DOS-экрана, в меню **Project** выберем команду **ConsoleApplication1 Properties** и на вкладке **Application** укажем в раскрывающемся списке **Application type** тип приложения **Windows Forms Application**. Чтобы иметь доступ к функции `MessageBox.Show`, добавим к проекту ссылку на динамическую библиотеку Forms.dll. Для этого выберем пункты меню **Project | Add Reference** и на вкладке **.NET** дважды щелкнем по ссылке на библиотеку **System.Windows.Forms.dll**. Отметим, что при этом в окне **Sulution Explorer** среди ссылок **References** появится соответствующая этой библиотеке ссылка.

Далее напишем программный код из листинга 10.8.

Листинг 10.8. Удаление записей из таблицы БД

```
// Программа удаляет запись из таблицы БД с использованием SQL-запроса и
// объекта класса Command
using System.Windows.Forms; // - добавили эту директиву для MessageBox
// Добавляем эту директиву для краткости выражений:
using ОлеДиБи = System.Data.OleDb;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace БдУдаленЗаписи
{
   class Program
   {
      static void Main(string[] args)
      {  // Создаем объект Connection и передаем ему строку подключения
         var Подключение = new ОлеДиБи.
               OleDbConnection( // Строка подключения:
                  "Data Source=\"C:\\vic.mdb\";User " +
                  "ID=Admin;Provider=\"Microsoft.Jet.OLEDB.4.0\";");
```

```
            Подключение.Open();
            // Создаем объект класса Command, передавая ему SQL-команду
            var Команда = new ОлеДиБи.OleDbCommand(
                            "Delete * From [БД телефонов] Where " +
                            "ФИО Like 'Vi%'", Подключение);
            // Выполнение команды SQL
            int i = Команда.ExecuteNonQuery();
            // i - количество удаленных записей
            if (i > 0) MessageBox.Show(
                "Записи, содержащие в поле ФИО фрагмент 'Vi*', удалены");
            if (i == 0) MessageBox.Show(
                "Запись, содержащая в поле ФИО фрагмент 'Vi*', не найдена");
            Подключение.Close();
        }
    }
}
```

Здесь при создании объекта класса `Command` задан SQL-запрос на удаление (`Delete`) всех записей, содержащий в поле **ФИО** фрагмент текста `"Vi*"`, причем строчные и прописные буквы являются равнозначными, т. е. будут удалены записи, содержащие `"Vi*"`, `"vi*"`, `"VI*"` и прочие комбинации. Таким образом, поиск записей ведется без учета регистра (case-insensitive search).

Замечу, что здесь для выполнения команды SQL использован метод `ExecuteNonQuery`. Он возвращает в переменную `i` количество удаленных записей. Если `i = 0`, значит, записи с таким контекстом не найдены и ни одна запись не удалена.

Убедиться в работоспособности программы можно, открыв решение БдУдаленЗаписи.sln в папке БдУдаленЗаписи.

Глава 11

Использование технологии LINQ

Технология LINQ (Language Integrated Query) предназначена для обработки (запросов и преобразований) практически любого типа источника данных, начиная от массивов, файлов, строк, коллекций объектов .NET Framework, баз данных SQL Server, наборов данных ADO.NET (DataSet) и XML-документов. LINQ упрощает ситуацию, предлагая единообразные стандартные шаблоны для работы с данными в различных видах источников и форматов данных. Стандартные шаблоны включают в себя основные операции запросов LINQ: фильтрация, упорядочение, группировка, соединение, выбор (проецирование), статистическая обработка. По форме синтаксис языка LINQ очень похож на язык запросов SQL. В данной главе рассмотрим типичные LINQ-запросы и преобразования к некоторым источникам данных.

Пример 75. LINQ-запрос к массиву данных

LINQ-запрос представляет собой выражение, извлекающее данные из источника данных. Все операции запроса LINQ состоят из трех различных действий: получение источника (в нашем случае — это присвоение начальных значений исходному массиву) данных, создание запроса (начинается с предложения `from`) и непосредственное выполнение запроса (обычно это цикл `foreach`). В данном разделе рассмотрим две задачи, первая — из массива имен извлекаем имена длиной шесть символов, записывая их или в список (коллекцию), или в новый массив. Другая задача — из массива целых чисел выбираем только те, значения которых больше четырех, также записывая результат запроса в список или массив.

Итак, запустим Visual Studio 2010, закажем новый проект шаблона **Windows Forms Application C#**. Затем из панели элементов перенесем в форму текстовое поле. Далее через щелчок правой кнопкой мыши перейдем к вкладке программного кода (листинг 11.1).

Листинг 11.1. Извлечение данных из массивов

```
// Решаем две задачи по выбору элементов из массива с помощью стандартных
// запросов технологии LINQ
```

```
using System;
using System.Linq;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace Linq1
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
         this.Text = "Технология LINQ"; textBox1.Multiline = true;
         // ЗАДАЧА 1.
         // Из массива имен выбрать имена с количеством букв, равным шести,
         // вывести эти имена в текстовое поле TextBox в алфавитном порядке,
         // при этом все буквы перевести в верхний регистр.
         // Решение:
         string СтрокаИмен = "Витя Лариса Лариса Лена Андрей Женя " +
                        "Александр Лариса Виктор Света Оксана Наташа";
         // Из строки имен получаем массив имен, задавая в качестве
         // разделителя подстрок символ пробела:
         string[] Имена = СтрокаИмен.Split(' ');
         // или проще: string[] Имена =
         //   { "Витя", "Лариса", "Лариса", "Лена", "Андрей", "Женя",
         //   "Александр", "Лариса", "Виктор", "Света", "Оксана", "Наташа" };
         textBox1.Text = "ЗАДАЧА 1. В списке имен:\r\n\r\n";
         foreach (string x in Имена)
            textBox1.Text = textBox1.Text + x + " ";
         // В результате LINQ-запроса получаем список имен с количеством букв,
         // равным шести:
         var Запрос = from s in Имена
                  where s.Length == 6  // - условие выбора
                  orderby s            // - сортировать в алфавитном порядке
                  select s.ToUpper();  // - перевод в верхний регистр
         // s - переменная диапазона, схожа с переменной итерации в foreach
         // Удаляем элементы-дубликаты из списка имен:
         Запрос = Запрос.Distinct();
         // Или таким образом:
         // Запрос = Запрос.Union(Запрос);
         textBox1.Text = textBox1.Text + "\r\n\r\n" +
                   "выбираем имена с количеством букв равным шести, при " +
                   "этом избавляемся от дублирования имен:\r\n\r\n";
```

```
         foreach (string x in Запрос) // x - переменная итерации
                                      // в цикле foreach
            textBox1.Text = textBox1.Text + x + " ";
         textBox1.Text = textBox1.Text + "\r\n\r\n";

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
         // ЗАДАЧА 2. Из массива целых чисел X[] требуется выбрать числа,
         //         значения которых >= 4, и записать эти числа в список Y,
         //         отсортировав выбранные числа по возрастанию.
         // Решение.
         // Инициализация массива целых чисел:
         int[] X = { -2, 5, -23, 0, 7, -10, 11, 11, 14, 3, 8, 5, -5, 27, 8 };
         textBox1.Text += "ЗАДАЧА 2. Из заданного массива X:\r\n\r\n";
         foreach (int x in X)
            textBox1.Text = textBox1.Text + x + "  ";
         textBox1.Text = textBox1.Text + "\r\n\r\nвыбираем числа, значения " +
                "которых >= 4 и записываем их в список (коллекцию) Y, " +
                "исключая элементы-дубликаты:\r\n\r\n";
         // Y - это список, в который помещаются выбранные элементы:
         var Y = from x in X
                   where x >= 4  // - условие выбора
                   orderby x     // - сортируем по возрастанию
                   select x;
         // Таким образом можно получить результат запроса в массив:
         // int[] Y = (from x in X // Здесь Y - это уже массив;
         //          where x >= 4   // - условие выбора;
         //          orderby x      // - сортируем по возрастанию;
         //          select x).ToArray(); // - преобразование списка в массив
         // Удаляем элементы-дубликаты из списка целых чисел:
         var Z = Y.Distinct();
         // Или таким образом:
         // var Z = Y.Union(Y);
         // Вывод элементов списка Y в метку textBox1:
         foreach (var z in Z)
            textBox1.Text = textBox1.Text + z.ToString() + " ";
      }
   }
}
```

Как видно из программного кода, после присвоения массиву `Имена` начальных значений создаем запрос, который предусматривает выбор (`select`) из (`from`) массива `Имена` строк длиной (`Length`) ровно шесть символов (условие `where`). Запись выбранных имен выполняется в список `Запрос` с сортировкой списка в алфавитном

порядке имен. Далее для удаления повторяющихся имен в списке используем функцию `Distinct`. В комментарии показано, как можно для этой же цели использовать функцию `Union`, которая, вообще говоря, находит объединение двух множеств, т. е. выбирает в двух списках одинаковые элементы с удалением повторяющихся элементов. Поскольку мы объединяем два одинаковых списка, то получаем просто удаление повторяющихся элементов. Цикл `foreach` выводит список `Запрос` в текстовое поле `textBox1`.

Следующая задача, решенная в данном программном коде, аналогична. Задан массив целых чисел `X`. Из этого массива выбираем в список `Y` элементы массива, значения которых больше или равны четырем. В комментарии показано, как можно записать результаты LINQ-запроса в массив. Удаление повторяющихся элементов в списке выполнено также с использованием функции `Distinct`.

На рис. 11.1 приведен фрагмент работы программы.

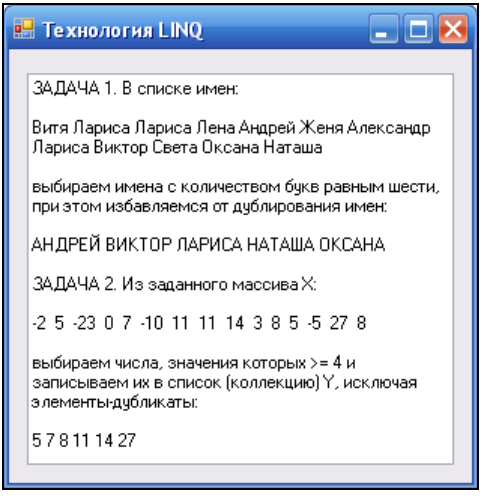


Рис. 11.1. LINQ-запросы к массивам данных

Убедиться в работоспособности программы можно, открыв решение Linq1.sln папки Linq1.

Пример 76. LINQ-запрос к коллекции (списку) данных

В некоторых случаях хранение данных в коллекции (скажем, в списке типа `List`) может оказаться более эффективным, чем в массиве. Например, если число элементов в массиве при работе изменяется часто или нельзя предсказать максимальное количество необходимых элементов, то можно получить большую производительность при использовании коллекции. Но если размер не изменяется или

изменяется довольно редко, то массив, пожалуй, более эффективен. Как всегда, производительность в большей степени зависит от конкретного приложения. Как советуют в документации MSDN, часто стоит потратить время на испытание и массива, и коллекции, чтобы выбрать наиболее практичный и эффективный вариант.

Решим три типичных задачи, их условия сформулируем при объяснении работы программы. Запустим Visual Studio 2010, закажем новый проект шаблона **Windows Forms Application C#.** Затем из панели элементов перенесем в форму текстовое поле. Далее через щелчок правой кнопкой мыши перейдем к вкладке программного кода (листинг 11.2).

Листинг 11.2. Извлечение данных из списков

```
// Решаем три различных задачи по выбору элементов (объектов) из списка
// с помощью стандартных запросов технологии LINQ
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace Linq2
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
         this.Text = "Технология LINQ"; textBox1.Multiline = true;
         // ЗАДАЧА 1. Из списка строк выбрать нужные записи,
         //           задав условие выбора
         textBox1.Text = "ЗАДАЧА 1: Из списка имен:\r\n";
         // Объявление списка строк и его заполнение:
         var Список = new List<string> { "Витя", "Света", "Андрей",
                                         "Лариса", "Маша", "Наташа" };
         // или var Список = new List<string>();   // список строк
         // Список.Add("Витя");   Список.Add("Света"); Список.Add("Андрей");
         // Список.Add("Лариса"); Список.Add("Маша");  Список.Add("Наташа");
         // Некоторые манипуляции со списком:
         int n = Список.Count;   // количество элементов в списке
         // Получение из списка третьего элемента (как в массиве):
         string A = Список.ElementAt<string>(4);
         Boolean Ответ = Список.Remove("Лариса");  // - удаление из списка
         // Преобразовать список в строковый массив:
```

```
string[] МассивСтрок = Список.ToArray();
foreach (var x in Список)
   textBox1.Text = textBox1.Text + x.ToString() + " ";
textBox1.Text += "\r\nвыбираем имена длиной четыре символа:\r\n";
// СписокВыбранныхИмен - это новый список, в который попадают
// выбранные строки в результате LINQ-запроса:
var СписокВыбранныхИмен = from Имя in Список
                  where Имя.Length == 4  // условие выбора
                  orderby Имя            // сортировать список
                  select Имя;
// Вывод списка выбранных имен в текстовое поле textBox1:
foreach (var x in СписокВыбранныхИмен)
   textBox1.Text = textBox1.Text + x.ToString() + " ";
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// ЗАДАЧА 2.
textBox1.Text +=
   "\r\n\r\nЗАДАЧА 2: Из списка сотрудников предприятия " +
            "выбираем некурящих для повышения зарплаты:\r\n\r\n";
// Заполняем список сотрудников:
List<Сотрудник> Сотрудники = new List<Сотрудник>{
  new Сотрудник {Имя="Карапузова Ирина", Возраст=27, КуритЛи=true },
  new Сотрудник {Имя="Зиборов Виктор", Возраст=47, КуритЛи=false },
  new Сотрудник {Имя="Ломачинская Светлана", Возраст=31,
                 КуритЛи=false },
  new Сотрудник {Имя="Стороженко Светлана", Возраст=34,
                 КуритЛи=false },
  new Сотрудник {Имя="Еременко Татьяна", Возраст=22, КуритЛи=true },
  new Сотрудник {Имя="Погребицкий Олег", Возраст=42, КуритЛи=true },
};
var СписокНекурящихСотрудников = from Сотрудник in Сотрудники
                        where Сотрудник.КуритЛи == false
                        orderby Сотрудник.Имя
                        select Сотрудник;
// Вывод списка некурящих сотрудников в текстовое поле textBox1:
foreach (var x in СписокНекурящихСотрудников)
   textBox1.Text = textBox1.Text + string.Format("{0} - возраст " +
                           "- {1}\r\n", x.Имя, x.Возраст);
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// ЗАДАЧА 3.
textBox1.Text += "\r\nЗАДАЧА 3: Из списка студентов факультета " +
                "выбираем двоечников:\r\n\r\n";
// Каждый элемент в списке содержит фамилию студента
// и полученные им текущие оценки:
```

```
        List<Студент> Студенты = new List<Студент> {
            new Студент {Фамилия="Зиборов",
                         Оценки= new List<int> {5, 4, 4, 5}},
            new Студент {Фамилия="Стороженко",
                         Оценки= new List<int> {3, 3, 2, 3}},
            new Студент {Фамилия="Ломачинская",
                         Оценки= new List<int> {3, 4, 4, 5}},
            new Студент {Фамилия="Погребицкий",
                         Оценки= new List<int> {2, 4, 3, 2}},
            new Студент {Фамилия="Левочкин",
                         Оценки= new List<int> {3, 3, 4, 3}}
        };
        // Для доступа к внутреннему списку оценок предложение From
        // используем два раза:
        var СписокДвоечников = from Студент in Студенты
                               from Оценка in Студент.Оценки
                               where Оценка <= 2
                               orderby Студент.Фамилия
                               select new { Студент.Фамилия, Оценка };
        foreach (var Студик in СписокДвоечников)
           textBox1.Text += string.Format("Студент {0} " +
                  "имеет оценку: {1}\r\n", Студик.Фамилия, Студик.Оценка);
        // Строка со студентом Погребицким выводится два раза,
        // поскольку он имеет две двойки
     }
  }
  // Объявляем класс, содержащий имя сотрудника, его возраст, а также
  // информацию, курит ли он:
  public class Сотрудник
  {
     public string Имя     { get; set; }
     public int    Возраст { get; set; }
     public bool   КуритЛи { get; set; }
  }
  // Объявляем класс, содержащий фамилию студента и список полученных
  // им оценок:
  public class Студент
  {
     public string    Фамилия { get; set; }
     public List<int> Оценки  { get; set; }
  }
}
```

Первая задача довольно простая и очень похожа на задачи, которые мы решали с помощью технологии LINQ в предыдущем примере. Здесь, в этой новой задаче вместо массива имен требуется создать список имен, а далее из этого списка выбрать имена длиной четыре символа. Решение этой задачи, построение LINQ-запроса аналогично решению задачи из предыдущего примера, отличие состоит лишь в применении синтаксиса манипуляций со списком типа `List`, а не с массивом. Здесь также приведены некоторые важные техники для манипуляции списком, в частности получение элемента списка по указанному индексу (аналогия с массивом), удаление элемента, преобразование списка в строковый массив.

Вторая задача заключается в том, чтобы создать список сотрудников предприятия и из этого списка выбрать некоторых сотрудников по какому-либо признаку, например тех, кто не курит. При создании списка объявлен класс `Сотрудник`, который содержит три свойства: `Имя`, `Возраст` и булеву переменную `КуритЛи`. В начале решения заполняем список сотрудников, а затем строим LINQ-запрос для заполнения списка некурящих сотрудников. Этот список выводим в текстовое поле `textBox1`, используя цикл `foreach`.

Третья задача немного сложнее. Требуется создать список студентов факультета, содержащий фамилию студента и список полученных им текущих оценок, т. е. список оценок должен быть "вложен" в список студентов. Из списка студентов необходимо выбрать тех, кто имеет в списке своих оценок хотя бы одну двойку. Для решения этой задачи вначале объявляем новый класс `Студент`, который имеет в качестве свойств класса фамилию студента и список (типа `List`) оценок. Далее в начале решения третьей задачи мы привели синтаксис заполнения списка студентов. Затем строим LINQ-запрос, где, поскольку нам необходимо искать элемент списка внутри "вложенного" списка, мы используем предложение `From` два раза. Поскольку студент Погребицкий имеет две двойки в списке своих текущих оценок, он в списке двоечников фигурирует дважды (рис. 11.2).

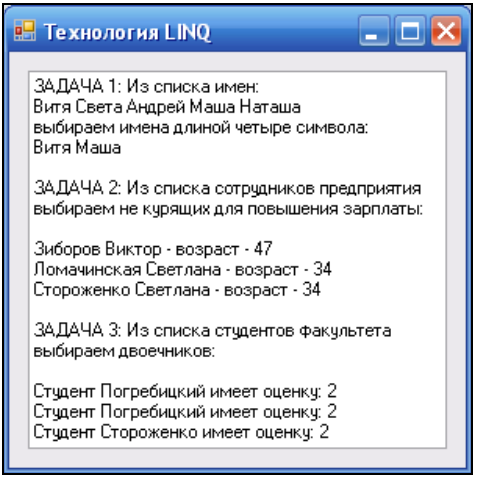


Рис. 11.2. Три LINQ-запроса к спискам данных

Убедиться в работоспособности программы можно, открыв решение Linq2.sln папки Linq2.

Пример 77. Группировка элементов списка с помощью LINQ-запроса

В данном примере используем стандартный шаблон LINQ-запроса для группировки элементов списка. Программа формирует список продуктов питания. Следует организовать такие LINQ-запросы, которые разделят искомый список на две группы по критерию цены (больше или меньше 90 руб. за единицу) и вычислят среднюю цену по каждой группе продуктов.

Для решения этой задачи запустим Visual Studio 2010 и выберем проект шаблона **Windows Forms Application**, укажем имя **Name** — LinqЦеныНаПродукты. Далее, попав в конструктор формы, из панели элементов **Toolbox** перетащим текстовое поле **TextBox** для вывода в него групп списка. В свойствах текстового поля разрешим ввод множества строк, для этого свойство `Multiline` переведем в состояние `true`. Затем на вкладке программного кода введем текст, представленный в листинге 11.3.

Листинг 11.3. Группировка элементов списка с помощью LINQ-запросов

```
// Программа формирует список некоторых продуктов питания. Первый LINQ-запрос
// группирует элементы списка по критерию цены: в первом списке оказываются
// продукты, цена за единицу которых меньше или равна 90 руб., а во втором,
// соответственно, больше 90 руб. Второй LINQ-запрос вычисляет среднюю цену
// продукта по каждой группе. Результаты запросов выводятся в текстовое поле
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace LinqЦеныНаПродукты
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
         textBox1.Multiline = true;
         base.Text = "Группировка элементов списка с помощью LINQ-запроса";
```

```
// Заполняем список продуктов:
List<Продукт> Продукты = new List<Продукт>{
         new Продукт {Наименование="Творог",  Цена=112.50F },
         new Продукт {Наименование="Хлеб",    Цена=18.75F },
         new Продукт {Наименование="Печенье", Цена=93.75F },
         new Продукт {Наименование="Чай",     Цена=76.25F },
         new Продукт {Наименование="Мясо",    Цена=150.00F },
         new Продукт {Наименование="Гречка",  Цена=62.50F },
};
var Запрос1 = from П in Продукты
              group П by new
              {
                 Критерий = П.Цена > 90,
              }
              into g
              select g;
var Запрос2 = from p in Продукты
              group p by p.Цена > 90 into g
              select new
              {
                 g.Key,
                 СредЦенаПоГруппе = g.Average(p => p.Цена)
              };
Single СредЦенаПоГруппе1 = Запрос2.ElementAt(0).СредЦенаПоГруппе;
Single СредЦенаПоГруппе2 = Запрос2.ElementAt(1).СредЦенаПоГруппе;
// Вывод результатов обоих запросов в текстовое поле:
foreach (var Группа in Запрос1)
{
   if (Группа.Key.Критерий == false)
      textBox1.Text += "\r\nЦены 90 руб или меньше:";
   else
      textBox1.Text += "\r\nЦены больше 90 руб:";
   foreach (var Прод in Группа)
   {
      textBox1.Text += String.Format("\r\n{0} - {1}",
                           Прод.Наименование, Прод.Цена);
   }
   if (Группа.Key.Критерий == false)
      textBox1.Text += String.Format(
         "\r\nСредняя цена по данной группе = {0} руб.\r\n",
                                      СредЦенаПоГруппе2);
   else
      textBox1.Text += String.Format(
```

```
                    "\r\nСредняя цена по данной группе = {0} руб.\r\n",
                                                    СредЦенаПоГруппе1);
            }
        }
    }
    public class Продукт
    {
        public String Наименование { get; set; }
        public Single Цена { get; set; }
    }
}
```

В начале программы формируем список продуктов питания. Для этого объявляем новый класс `Продукт`, в котором для каждого продукта задаем два поля: наименование продукта и его цену. После заполнения списка продуктов (для упрощения в этом списке всего шесть продуктов) задаем первый LINQ-запрос, обеспечивающий деление списка на две группы. Второй LINQ-запрос вычисляет среднюю цену продукта по каждой группе. Вывод результатов обоих запросов организуем с помощью двух вложенных циклов `foreach`.

Фрагмент работы программы показан на рис. 11.3.

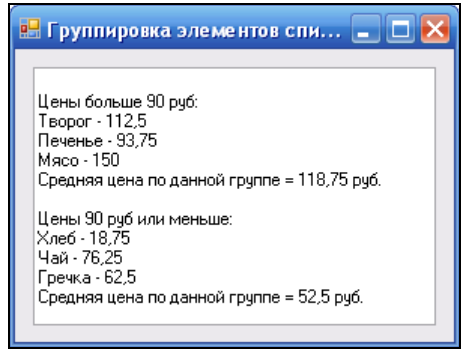


Рис. 11.3. Группировка списка с помощью LINQ-запросов

Убедиться в работоспособности программы можно, открыв решение LinqЦеныНаПродукты.sln папки LinqЦеныНаПродукты.

Пример 78. LINQ-запрос к словарю данных *Dictionary*

Замечательной структурой данных является словарь `Dictionary`. Он представляет собой совокупность (коллекцию) ключей и значений. То есть каждый элемент (запись), добавляемый в словарь, состоит из значения `Value` и связанного с ним

ключа Key. Извлечение значения по его ключу происходит очень быстро, поскольку класс Dictionary<Key, Value> реализован как хэш-таблица. Каждый ключ в словаре Dictionary<Key, Value> должен быть уникальным, т. е. единственным в своем роде, эксклюзивным. При добавлении в коллекцию Dictionary очередного элемента так называемый компаратор проверяет на равенство уникальность нового ключа. Ключ не может быть пустым (null), а значение может, если тип значения Value является ссылочным типом. Возможно создание словаря, в котором не различается регистр символов. Использование словаря Dictionary может существенно повлиять на эффективность алгоритма, на простоту его понимания и легкость программной реализации.

Задача, решаемая в данном примере, состоит в том, чтобы продемонстрировать возможность выбора элементов из словаря данных Dictionary с помощью LINQ-запроса. В начале программы зададим массив сотрудников некого учреждения, а затем массив сотрудников преобразуем в словарь Dictionary. Причем в качестве ключа к каждому элементу словаря зададим имя сотрудника, которое является уникальным. По ключу можно получить все остальные сведения о сотруднике, записанные в словарь. Кроме того, с помощью LINQ-запроса к словарю можно получать новую коллекцию сотрудников по какому-либо условию (предложение where). Здесь мы зададим выбор сотрудников, чей возраст превышает 33 года (возраст Иисуса Христа).

Для решения этой задачи запустим Visual Studio 2010 и выберем проект шаблона **Console Application**, укажем имя **Name** — LinqDictionary. Затем на вкладке программного кода введем текст, представленный в листинге 11.4.

Листинг 11.4. Организация LINQ-запроса к словарю Dictionary

```
// Задаем массив сотрудников учреждения. Из этого массива создаем словарь
// сотрудников, а в качестве ключа к этому словарю выбираем имя сотрудника.
// С помощью LINQ-запроса из массива сотрудников выбираем тех, чей возраст
// превышает 33 года. При выводе результата запроса на печать учитываем, что
// мы говорим "47 лет", но "34 года". То есть если из возраста
// вычесть число, кратное 10, то при остатке меньше 5 говорят,
// например, "34 года", а при остатке больше или равном 5 говорят "47 лет"
using System;
using System.Linq;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace LinqDictionary
{
   class Program
   {
      static void Main(string[] args)
      {
         Console.Title = "LINQ-запрос к словарю Dictionary";
```

```
// Создаем массив сотрудников учреждения:
var Сотрудники = new[] {
  new {Имя = "Карапузова Ирина",     Возраст = 27, КуритЛи = true },
  new {Имя = "Зиборов Виктор",       Возраст = 47, КуритЛи = false },
  new {Имя = "Ломачинская Светлана", Возраст = 31, КуритЛи = false },
  new {Имя = "Стороженко Светлана",  Возраст = 34, КуритЛи = false },
  new {Имя = "Еременко Татьяна",     Возраст = 22, КуритЛи = true },
  new {Имя = "Погребицкий Олег",     Возраст = 42, КуритЛи = true }
};
// Доступ к элементу массива Сотрудники можем иметь через его индекс:
var t = Сотрудники[2];
// Строим LINQ-запрос к массиву сотрудников; выбираем тех,
// чей возраст больше 33 лет:
// var СписокВзрослыхСотрудников = from Сотрудник in Сотрудники
//                          where Сотрудник.Возраст >= 33
//                          orderby Сотрудник.Имя
//                          select Сотрудник;
// Из массива сотрудников создаем словарь сотрудников, в котором
// ключом является имя сотрудника:
var СловарьСотрудников = Сотрудники.ToDictionary(Ключ => Ключ.Имя);
// В этом случае очень удобным становится доступ к сведениям
// о сотруднике по его имени:
Boolean КуритЛиЗиборов = СловарьСотрудников
["Зиборов Виктор"].КуритЛи;
int ВозрастЗиборова = СловарьСотрудников["Зиборов Виктор"].Возраст;
Console.WriteLine("Сотруднику Зиборову Виктору - {0} лет\n",
                                       ВозрастЗиборова);
// Строим LINQ-запрос к словарю сотрудников; выбираем тех,
// чей возраст больше 33 лет:
var СписокВзрослыхСотрудников =
            from Сотрудник in СловарьСотрудников
            where Сотрудник.Value.Возраст >= 33
            orderby Сотрудник.Value.Имя
            select new
            {
               Сотрудник.Key,
               Сотрудник.Value.Возраст,
               ЛетИлиГода = Сотрудник.Value.Возраст - 10F *
                 Math.Truncate(Сотрудник.Value.Возраст / 10F) >= 5
            };

// Вывод результата запроса на консоль:
Console.WriteLine("Список сотрудников, старше 33 лет:");
```

```
            foreach (var x in СписокВзрослыхСотрудников)
               Console.WriteLine("{0}, возраст - {1} {2} ", x.Key, x.Возраст,
                                 x.ЛетИлиГода ? "лет" : "года");
            Console.ReadKey();
         }
      }
}
```

Представленный текст программы очевиден. Вначале задаем массив сотрудников учреждения, в этом массиве всего 6 элементов. В комментарии показано, как можно организовать LINQ-запрос к заданному массиву. Далее массив конвертируем в словарь сотрудников, задавая в качестве ключа имя сотрудника. Затем организуем LINQ-запрос к словарю `Dictionary`, выбираем в новую коллекцию (список) тех, чей возраст превышает 33 года. Учитываем, что при выводе на консоль списка выбранных сотрудников мы говорим "47 лет", но "34 года". То есть если из возраста вычесть число, кратное 10, то при остатке меньше 5 говорят, например, "34 года", а при остатке больше или равном 5 говорят "47 лет". Число, кратное 10, мы вычисляем, используя функцию `Math.Truncate`, которая возвращает целую часть числа, поданную на ее вход. В этом алгоритме мы использовали булеву переменную `ЛетИлиГода`, которая определяет (по принципу "да" или "нет") следует писать "лет" или "года".

Результат работы программы показан на рис. 11.4.

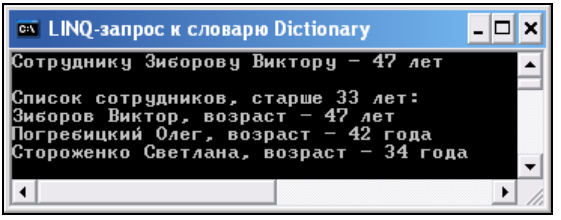


Рис. 11.4. Запрос к словарю данных на "взрослых" сотрудников

Убедиться в работоспособности программы можно, открыв решение LinqDictionary.sln папки LinqDictionary.

Пример 79. Создание XML-документа методами классов пространства имен *System.Xml.Linq*

Кроме пространства имен `System.Xml`, содержащего классы для обработки XML-документов, в Visual C# 2010 имеем пространство имен `System.Xml.Linq`,

содержащее классы, которые позволяют легко и эффективно изменять документы XML, а также организовывать LINQ-запросы. В данном примере оформим сведения о наших повседневных телефонных контактах в XML-документ. Этот документ будет иметь интуитивно понятную структуру: имя контакта, домашний и мобильный телефоны. Создав такой XML-документ и получив соответствующий XML-файл, его очень удобно просмотреть в MS Excel в виде таблицы, содержащей три столбца: имя контакта, домашний телефон и мобильный. Попутно обсудим структуру XML-документа.

Итак, запустим среду Visual Studio 2010, выберем проект шаблона **Console Application**, укажем имя **Name** — LinqСоздатьXML-документ. Затем на вкладке программного кода введем текст, представленный в листинге 11.5.

Листинг 11.5. Создание XML-документа представляющего телефонную книгу

```
// Программа создает типичный XML-документ. С ее помощью можно разобраться в
// структуре XML-документа. В комментариях приведена терминология содержимого
// XML-документа: корневой элемент, вложенные элементы, имя элемента и его
// значение, а также атрибуты элемента, их имена и значения. XML-документ
// представляет телефонную книгу, содержащую имя контакта, номер домашнего
// телефона, а также мобильного. Программа после создания XML-документа
// отображает его на консоли, а также записывает его в файл. Если этот файл
// открыть с помощью MS Excel, то мы получим таблицу из трех столбцов.
using System;
using System.Xml.Linq;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace LinqСоздатьXML_документ
{
   class Program
   {
      static void Main(string[] args)
      {
         Console.Title = "Корневой элемент XML-документа";
         // Создаем новый XML-документ:
         XDocument XMLдокумент = new XDocument(
            // Комментарий в XML-документе:
      new XComment("Телефонная_книга - это корневой элемент XML-документа:"),
      new XElement("Телефонная_книга", // - имя корневого элемента

      new XComment("Элемент СТРОКА содержит атрибут Контакт и два вложенных
элемента"),
      new XElement("СТРОКА", // - имя (Name) элемента
        new XAttribute("Контакт", "Олег"),
```

```
        new XElement("Домашний_телефон", "236-23-67"), // - имя элемента
                                                       // и его значение
        new XElement("Мобильный_телефон", "+7(495)625-31-43")),

      new XComment("Атрибут Контакт имеет значение 'Прогноз погоды':"),
      new XElement("СТРОКА",
        new XAttribute("Контакт", "Прогноз погоды"), // - атрибут элемента
                                                     // СТРОКА
        new XElement("Домашний_телефон", "001"),
        new XElement("Мобильный_телефон", "")), // - имя элемента
                                               // и его зачение (Value)

      new XComment("Поскольку каждый элемент Контакт имеет атрибут
      и два вложенных=>"),
      new XElement("СТРОКА",
        new XAttribute("Контакт", "Борис Григорьевич"), // - имя атрибута –
                                                        // Контакт
        new XElement("Домашний_телефон", "402-12-45"),
        new XElement("Мобильный_телефон", "+7(495)536-79-94")),

      new XComment("=> элемента, в MS Excel отобразится таблица с тремя
      колонками"),
      new XElement("СТРОКА",
        new XAttribute("Контакт", "Света"),    // - значение атрибута - Света
        new XElement("Домашний_телефон", ""),
        new XElement("Мобильный_телефон", "+7(495)615-24-41")))
                                      );
         // Сохранить XML-документ:
         XMLдокумент.Save(@"C:\Зиборов.XML");
         Console.WriteLine(XMLдокумент);
         Console.ReadKey();
      }
   }
}
```

Чтобы понять текст программы, рассмотрим структуру полученного XML-файла, а для этого откроем этот файл с помощью Internet Explorer (рис. 11.5).

Здесь весь XML-документ вложен в так называемый корневой элемент между начальным тегом `<Телефонная_книга>` и конечным тегом `</Телефонная_книга>`. Четыре элемента `СТРОКА` вложены в корневой элемент. В соответствующей таблице MS Excel элементы `СТРОКА` будут представлять строку в таблице. В свою очередь элемент `СТРОКА` содержит в себе атрибут `Контакт` и два вложенных в него элемента, имена (`Name`) которых — `Домашний_телефон` и `Мобильный_телефон`. Именно поэтому

в MS Excel отобразится таблица с тремя колонками (один атрибут и два элемента): "Контакт", "Домашний_телефон" и "Мобильный_телефон".

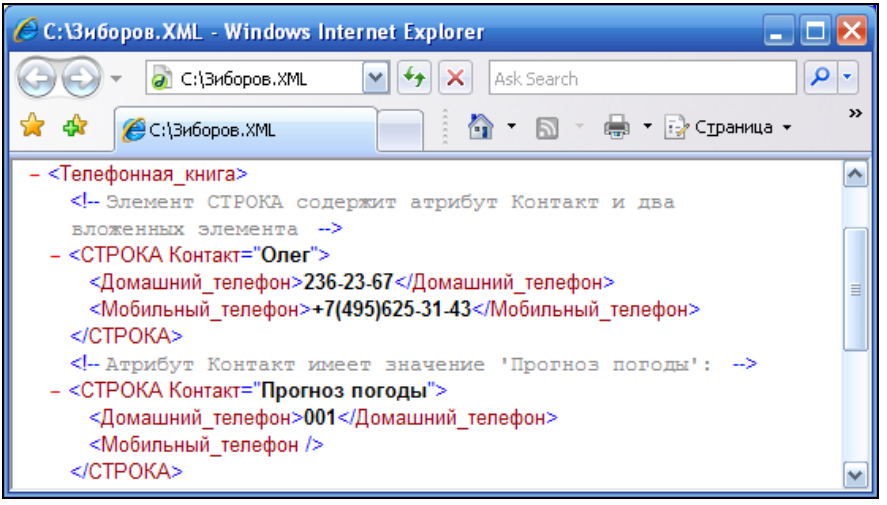


Рис. 11.5. XML-файл, открытый в Internet Explorer

Элемент может иметь один или несколько атрибутов (а может и не иметь, как, скажем, элемент `Домашний_телефон`), например, первый элемент `СТРОКА` имеет атрибут с именем (`Name`) `Контакт` и со значением атрибута (`Value`) — `001`.

После запуска данной программы будет выведено на консоль содержимое XML-документа (без XML-объявления), а также будет создан XML-файл. Открыв этот файл с помощью MS Excel, получим таблицу телефонных контактов (рис. 11.6).

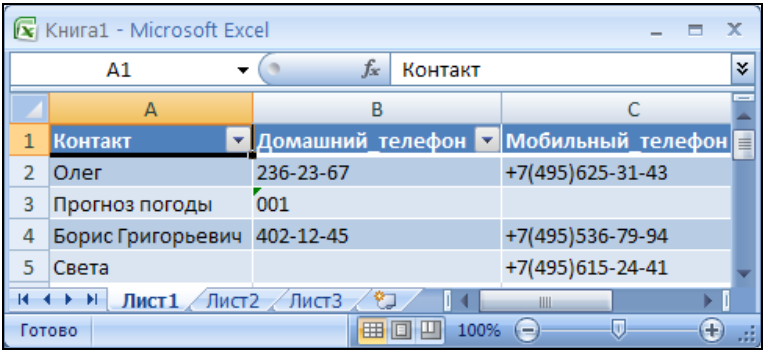


Рис. 11.6. XML-файл, открытый в MS Excel

Убедиться в работоспособности программы можно, открыв решение LinqСоздатьXML-документ.sln папки LinqСоздатьXML-документ.

Пример 80. Извлечение значения элемента из XML-документа

В данном примере, решая задачу, мы получили строку XML-данных, например от удаленной Web-службы, обеспечивающей прогнозом погоды. В этой текстовой XML-строке содержатся метеорологические показатели на заданный нами район для текущей даты. В данной задаче мы извлекаем из этих XML-данных только значение температуры.

Несколько слов о структуре XML-документа (рис. 11.7).

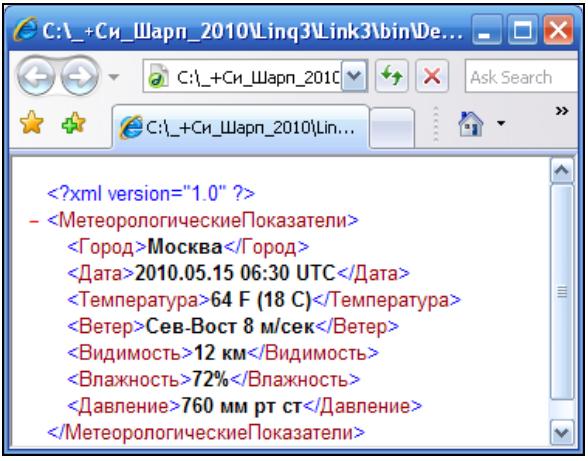


Рис. 11.7. Содержимое XML-файла с метеорологическими показателями

Как видно, XML-документ начинается с XML-объявления (XML declaration), в котором содержится информация о версии (version information parameter). А далее весь XML-документ состоит из вложенных друг в друга элементов. Элемент — это блок разметки между начальным тегом, например `<Город>`, и конечным тегом `</Город>`. Самый внешний элемент, в данном случае — это тег `<МетеорологическиеПоказатели>`, его называют *корневым элементом* (root element). Как видите, этот корневой элемент содержит в себе все показатели, и, таким образом, глубина вложенности этой иерархии равна двум. Вообще говоря, глубина вложенности такого XML-дерева может быть практически любой.

Итак, задача поставлена, сущность XML-данных понятна, приступаем к решению задачи. Для этой цели после запуска Visual Studio 2010 выберем проект шаблона **Windows Forms Application**, укажем имя **Name** — Linq3. Далее, попав в конструктор формы, из панели элементов **Toolbox** перетащим текстовое поле **TextBox** для вывода в него строки с данными XML и значения температуры из соответствующего элемента XML-дерева. Поскольку мы предполагаем вывод в текстовое поле не одной, а нескольких строчек, в свойствах объекта `textBox1` укажем `true`

напротив свойства `Multiline`. Затем на вкладке программного кода введем текст, представленный в листинге 11.6.

Листинг 11.6. Извлечение значения элемента из XML-данных

```
// Дана строка XML, содержащая прогнозные метеорологические показатели
// для Москвы на заданную дату. Программа извлекает из корневого элемента
// XML-документа значение температуры элемента "Температура"
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace Linq3
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
         this.Text = "LINQ-запрос к XML-данным"; textBox1.Multiline = true;
         string СтрокаXML =
            @"<?xml version=""1.0""?>
            <МетеорологическиеПоказатели>
               <Город>Москва</Город>
               <Дата>2010.05.15 06:30 UTC</Дата>
               <Температура> 64 F (18 C)</Температура>
               <Ветер>Сев-Вост 8 м/сек</Ветер>
               <Видимость>12 км</Видимость>
               <Влажность> 72%</Влажность>
               <Давление>760 мм рт ст</Давление>
            </МетеорологическиеПоказатели>";
         // Загрузка корневого элемента из строки, содержащей XML:
         var КорневойЭлемент = System.Xml.Linq.XElement.Parse(СтрокаXML);
         // Или корневой элемент XML-документа получаем через файл:
         // записываем строку, содержащую XML, в файл
         // System.IO.File.WriteAllText("ПоказателиПогоды.xml", СтрокаXML);
         // Загружаем корневой элемент XML:
         // var КорневойЭлемент = System.Xml.Linq.
         //                     XElement.Load("ПоказателиПогоды.xml");
         // Из корневого элемента извлекаем вложенный в него элемент
         // "Температура" и получаем соответствующее значение (Value)
         // этого элемента:
         string Температура = КорневойЭлемент.Element("Температура").Value;
```

```
            textBox1.Text = "Строка XML:\r\n\r\n" + СтрокаXML + "\r\n\r\n";
            textBox1.Text += "Значение температуры = " + Температура;
        }
    }
}
```

В начале текста программы задаем текстовую строку, содержащую XML-данные. Далее, используя метод `Parse` класса `XElement` пространства имен `Linq`, получаем корневой элемент XML-документа. В комментарии показано, как можно получить корневой элемент через запись/чтение XML-файла. Затем с помощью метода `Element` извлекаем значение (`Value`) элемента `Температура`, которое выводим в текстовое поле.

Фрагмент работы программы приведен на рис. 11.8.

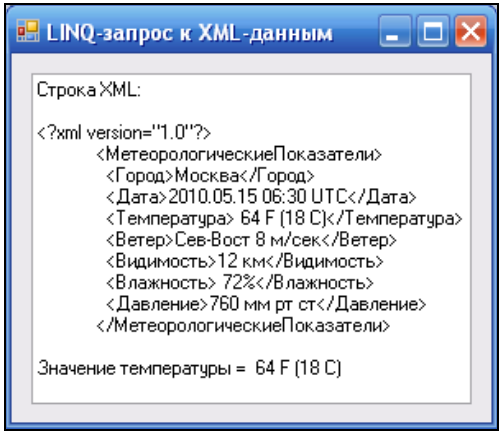


Рис. 11.8. Извлечение значения элемента из XML-документа

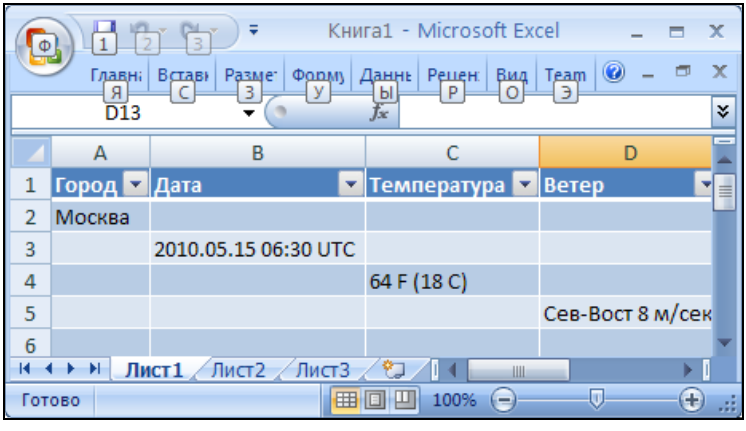


Рис. 11.9. Представление XML-данных в виде таблицы в MS Excel

Убедиться в работоспособности программы можно, открыв решение Linq3.sln папки Linq3.

Теперь решим похожую задачу по извлечению значения элемента, но пусть XML-данные представлены в другой форме, а именно каждый метеорологический показатель вложим в один и тот же элемент <Показатель> </Показатель>, в этом случае глубина вложенности элементов будет уже равна трем (см. листинг 11.7). Естественно спросить: что мы будем с этого иметь? Дело в том, что если соответствующий XML-файл открыть с помощью табличного редактора MS Excel, то мы сможем увидеть эти XML-данные в виде наглядной таблицы, даже не ссылаясь на таблицу стилей — файл XSLT (не путать с XLS-файлом), см. рис. 11.9.

Теперь для получения значения температуры удобно воспользоваться типовым LINQ-запросом (листинг 11.7).

Листинг 11.7. Извлечение значения элемента из XML-данных

```
// Дана строка XML, которая содержит прогнозные метеорологические
// показатели для Москвы на заданную дату. При этом каждый метеорологический
// показатель вложен в один и тот же элемент <Показатель> </Показатель>.
// Это обеспечивает удобный просмотр соответствующего XML-файла в MS Excel
// в виде таблицы. Программа находит в корневом элементе данного XML-документа
// элемент "Температура" и извлекает из него значение температуры.
using System.Windows.Forms;
using System.Linq;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace Linq4
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
         this.Text = "LINQ-запрос к XML-данным"; textBox1.Multiline = false;
         // Инициализация XML-строки:
         string СтрокаXML =
            @"<?xml version=""1.0""?>
            <МетеорологическиеПоказатели>
               <Показатель>
                  <Город>Москва</Город>
               </Показатель>
               <Показатель>
                  <Дата>2010.05.15 06:30 UTC</Дата>
               </Показатель>
```

```
            <Показатель>
               <Температура> 64 F (18 C)</Температура>
            </Показатель>
            <Показатель>
               <Ветер>Сев-Вост 8 м/сек</Ветер>
            </Показатель>
            <Показатель>
               <Видимость>12 км</Видимость>
            </Показатель>
            <Показатель>
               <Влажность> 72%</Влажность>
            </Показатель>
            <Показатель>
               <Давление>760 мм рт ст</Давление>
            </Показатель>
         </МетеорологическиеПоказатели>";
      var КорневойЭлемент = System.Xml.Linq.XElement.Parse(СтрокаXML);
      // Или корневой элемент получаем через файл:
      // записываем строку, содержащую XML, в файл
      // System.IO.File.WriteAllText("ПоказателиПогоды2.xml", СтрокаXML);
      // Загружаем корневой элемент:
      // var КорневойЭлемент = System.Xml.Linq.XElement.Load(
      //                               "ПоказателиПогоды2.xml");
      // Запрос - это коллекция (список) строк, в которую извлекаем
      // значение (Value) элемента "Температура":
      var Запрос = from x in КорневойЭлемент.Elements("Показатель")
                   from y in x.Elements("Температура")
                   select y.Value;
      // Таких строк в коллекции Запрос - одна
      textBox1.Text = "Значение температуры = ";
      foreach (var x in Запрос)
         textBox1.Text = textBox1.Text + x;
   }
  }
}
```

Как видно из программного кода, поиск организован в двух уровнях (два предложения `from`), сначала выбор в коллекцию всех элементов `Показатель`, а затем из этой коллекции поиск элементов `Температура`. Результат запроса записывается в коллекцию строк, где имеем ровно одну строку. Фрагмент работы программы показан на рис. 11.10.

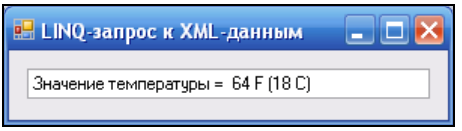


Рис. 11.10. LINQ-запрос к XML-документу

Убедиться в работоспособности программы можно, открыв решение Linq4.sln из папки Linq4.

Пример 81. Поиск строк (записей) в XML-данных с помощью LINQ-запроса

Имеем XML-данные, в которых содержится традиционная для нашей книги таблица с именами и телефонами, причем имена в этой телефонной табличке повторяются, например, строка с именем "Витя" содержит мобильный телефон, а потом по мере знакомства с этим Витей у нас появился уже и его домашний телефон. Задача состоит в том, чтобы в данной таблице телефонов (представленной в виде XML, см. листинг 11.8) найти все строчки с именем "Витя". Эта маленькая несерьезная, на первый взгляд, задача подразумевает, например, такую уже "серьезную" задачу. Имеем громадную базу данных, которую мы получили на каком-то этапе обработки в виде XML, и нам требуется "отфильтровать" записи в этой базе на предмет содержания в некотором поле определенной строки.

Прежде чем решать данную задачу, давайте посмотрим отображение обсуждаемых XML-данных в табличном редакторе MS Excel (рис. 11.11).

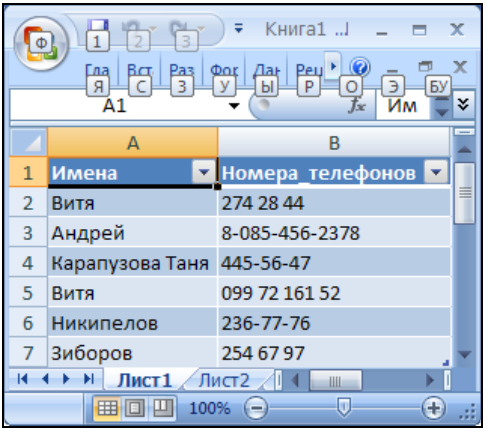


Рис. 11.11. Отображение XML-данных в MS Excel

Как видно, в редакторе MS Excel наши XML-данные представлены весьма наглядно. И очень понятно, что мы хотим получить, а именно, все номера телефонов напротив имени "Витя".

Для решения этой задачи запустим Visual Studio 2010 и выберем проект шаблона **Windows Forms Application**, укажем имя **Name** — Linq5. Далее, попав в конструктор формы, из панели элементов **Toolbox** перетащим текстовое поле **TextBox** для вывода в него найденных строк из таблицы XML. В свойствах текстового поля разрешим ввод множества строк (а не одной), для этого свойство `Multiline` переведем в состояние `true`. Затем на вкладке программного кода введем текст, представленный в листинге 11.8.

Листинг 11.8. Извлечение значения элемента из XML-данных

```
// Имеем XML-данные, в которых содержится таблица с именами и телефонами,
// причем имена в этой телефонной табличке повторяются. Задача состоит в том,
// чтобы в данной таблице телефонов (представленной в виде XML) найти все
// строчки с именем "Витя" с помощью LINQ-запроса
using System.Linq;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace Linq5
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
         this.Text = "LINQ-запрос к XML-данным"; textBox1.Multiline = true;
         // Инициализация XML-строки:
         string СтрокаXML =
           @"<?xml version=""1.0""?>
           <ТаблицаТелефонов>
              <Строка>
                <Имена>Витя</Имена>
                <Номера_телефонов>274 28 44</Номера_телефонов>
              </Строка>
              <Строка>
                <Имена>Андрей</Имена>
                <Номера_телефонов>8-085-456-2378</Номера_телефонов>
              </Строка>
              <Строка>
                <Имена>Карапузова Таня</Имена>
                <Номера_телефонов>445-56-47</Номера_телефонов>
```

```
            </Строка>
            <Строка>
              <Имена>Витя</Имена>
              <Номера_телефонов>099 72 161 52</Номера_телефонов>
            </Строка>
            <Строка>
              <Имена>Никипелов</Имена>
              <Номера_телефонов>236-77-76</Номера_телефонов>
            </Строка>
            <Строка>
              <Имена>Зиборов</Имена>
              <Номера_телефонов>254 67 97</Номера_телефонов>
            </Строка>
          </ТаблицаТелефонов>";
        var КорневойЭлемент = System.Xml.Linq.XElement.Parse(СтрокаXML);
        // Запись строки, содержащей XML, в файл:
        // System.IO.File.WriteAllText("ТаблицаТелефонов.xml", СтрокаXML);
        // var КорневойЭлемент = System.Xml.Linq.
        //                   XElement.Load("ТаблицаТелефонов.xml");
        var Записи = from x in КорневойЭлемент.Elements("Строка")
                     where (string)x.Element("Имена") == "Витя"
                     select x.Element("Номера_телефонов").Value;
        textBox1.Text = textBox1.Text +
                        @"Строки, содержащие имя ""Витя"":" + "\r\n";
        // Вывод коллекции записей в текстовое поле textBox1:
        foreach (var x in Записи)
           textBox1.Text = textBox1.Text + x + "\r\n";
        // Таких записей в этой коллекции - ровно одна
      }
   }
}
```

Как видно, в начале программы мы инициализируем (т. е. присваиваем начальные значения) XML-строку. Далее извлекаем корневой элемент из XML-документа, он, по сути, отличается от XML-документа отсутствием XML-объявления (в этом можно убедиться в отладчике программы). В комментарии указано, как можно получить корневой элемент в том случае, если он представлен в виде XML-файла во внешней памяти. Затем организуем типовой, стандартный LINQ-запрос. Результат запроса попадает в коллекцию записей, которую выводим в текстовое поле, используя оператор цикла `foreach`. Фрагмент работы программы показан на рис. 11.12.

Убедиться в работоспособности программы можно, открыв решение Linq5.sln из папки Linq5.

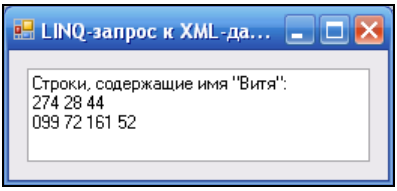


Рис. 11.12. LINQ-запрос к XML-документу

Пример 82. LINQ-запрос к набору данных *DataSet*

Весьма полезной оказывается организация LINQ-запросов к наборам данных `DataSet`, используемым, к примеру, при работе с базами данных. Объект класса `DataSet` представляет расположенный в памяти кэш (cache) данных (кэш — это промежуточная память с быстрым доступом, содержащая информацию, которая может быть запрошена с наибольшей вероятностью). Реляционные базы данных работают чаще всего с совокупностью таблиц. Каждая из этих таблиц задается как объект класса `DataTable`, один такой объект представляет ровно одну таблицу данных. Набор данных `DataSet` содержит в себе несколько объектов (таблиц) `DataTable`. Запросы LINQ к таблицам данных, кэшированным в объекте `DataSet`, упрощают и ускоряют процесс отбора.

Данная задача состоит в том, чтобы создать программу, которая обеспечивает ввод простейшей таблицы, содержащей два поля — название города и численность его населения. Программа способна фильтровать данные в таблице; будем производить отбор городов, численность населения которых превышает миллион жителей.

Для решения этой задачи запустим Visual Studio 2010 и выберем проект шаблона **Windows Forms Application**, укажем имя **Name** — LinqГорода. Далее, попав в конструктор формы, из панели элементов **Toolbox** перетащим элемент управления для отображения и редактирования табличных данных **DataGridView**, две командные кнопки **Button** и текстовое поле **TextBox**. Одна кнопка предназначена для команды сохранения данных, другая — для поиска данных в таблице, а текстовое поле — для вывода в него найденных строк из таблицы. В свойствах текстового поля разрешим ввод множества строк, для этого свойство `Multiline` переведем в состояние `true`. Затем на вкладке программного кода введем текст, представленный в листинге 11.9.

Листинг 11.9. Извлечение полей из набора данных `DataSet`

```
// В данной программе экранная форма содержит элемент управления
// для отображения и редактирования табличных данных DataGridView,
// две командные кнопки и текстовое поле. При старте программы,
// если есть соответствующий файл XML, программа отображает
```

```
// в DataGridView таблицу городов - название города и численность населения.
// При щелчке на кнопке "Сохранить" все изменения в таблице записываются
// в XML-файл. При щелчке на второй кнопке "Найти" выполняется LINQ-запрос
// к набору данных DataSet на поиск городов-миллионеров в искомой таблице.
// Результат запроса выводится в текстовое поле.
using System;
using System.Data;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace _LinqГорода
{
   public partial class Form1 : Form
   {
      DataTable Таблица = new DataTable(); // Создание объекта "таблица данных"
      DataSet НаборДанных = new DataSet(); // Создание объекта "набор данных"
      public Form1()
      {
         InitializeComponent();
         base.Text = "LINQ-запрос к набору данных DataSet";
         button1.Text = "Сохранить"; button2.Text = "Найти";
         textBox1.Multiline = true;
         if (System.IO.File.Exists("Города.xml") == false)
         {  // Если XML-файла НЕТ:
            // заполнение "шапки" таблицы
            Таблица.Columns.Add("Город"); Таблица.Columns.Add("Население");
            // Добавить объект Таблица в DataSet:
            НаборДанных.Tables.Add(Таблица);
            dataGridView1.DataSource = Таблица;
         }
         else // Если XML-файл ЕСТЬ:
         {
            НаборДанных.ReadXml("Города.xml");
            // Содержимое DataSet в виде строки XML для отладки:
            // string СтрокаXML = НаборДанных.GetXml();
            Таблица = НаборДанных.Tables["Города"];
            dataGridView1.DataMember = "Города";
            dataGridView1.DataSource = НаборДанных;
         }
      }
      private void button1_Click(object sender, EventArgs e)
      {  // Щелчок мышью на кнопке "Сохранить" - сохранить файл Города.xml:
         Таблица.TableName = "Города";
```

```
        НаборДанных.WriteXml("Города.xml");
    }
    private void button2_Click(object sender, EventArgs e)
    {  // Щелчок мышью на кнопке "Поиск" - запрос городов-миллионеров:
       textBox1.Clear(); // - очистка текстового поля
       var ГородаМлн = from Город in Таблица.AsEnumerable()
                       where Convert.ToInt32(Город.
                              Field<String>("Население")) >= 1000000
                       select new
                       {
                          A = Город.Field<string>("Город"),
                          B = Город.Field<string>("Население")
                       };
       textBox1.Text = textBox1.Text + "Города-миллионеры:\r\n";
       // Вывод результата запроса в текстовое поле textBox1:
       foreach (var Город in ГородаМлн)
          textBox1.Text = textBox1.Text +
                       Город.A + " - " + Город.B + "\r\n";
    }
  }
}
```

В начале программы создаем объекты классов `DataSet` и `DataTable` так, чтобы они были видимыми из всех процедур класса `Form1`. Далее сразу после инициализации компонентов экранной формы (т. е. после выполнения процедуры `InitializeComponent`) проверяем, существует ли файл Города.xml, куда мы записываем искомую таблицу. Если файл не существует, т. е. пользователь первый раз запустил нашу программу, то мы создаем таблицу, состоящую из двух полей (колонок): "Город" и "Население", добавляем (`Add`) эту таблицу в набор данных, а также указываем таблицу в качестве источника данных (`DataSource`) для сетки данных `dataGridView1`. Если же файл Города.xml уже создан, то мы читаем его в набор данных `DataSet` и из него заполняем таблицу данных, а также этот набор данных указываем в качестве источника для сетки данных.

При обработке события "щелчок мышью на кнопке" **Запись** программируем сохранение редактируемой таблицы в файле Города.xml. В процедуре обработки события "щелчок на кнопке" **Найти** организуем LINQ-запрос к заполненной пользователем таблице `DataTable`, являющейся представителем `DataSet`. Условием запроса является отбор таких полей таблицы, где население превышает миллион жителей. Заметим, что если не задавать условие `where`, то в результате запроса получим все содержимое источника данных, т. е. все строки таблицы городов. Результат запроса выводим в текстовое поле, используя цикл `foreach`.

Фрагмент работы программы показан на рис. 11.13.

Убедиться в работоспособности программы можно, открыв решение LinqГорода.sln из папки LinqГорода.

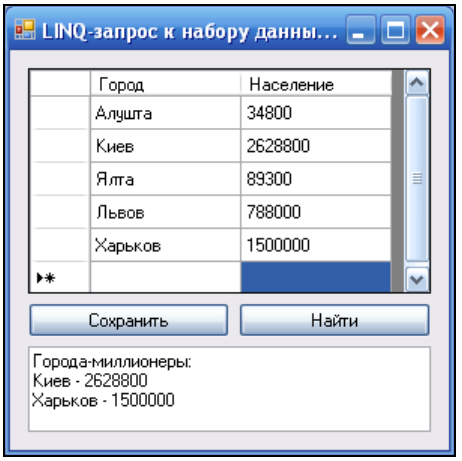


Рис. 11.13. LINQ-запрос к набору данных

Пример 83. Доступ к базе данных с помощью LINQ to SQL

В данном примере вначале создадим базу данных городов, содержащую два поля: название города и численность его населения. Затем организуем доступ к этой базе данных с помощью LINQ to SQL и создадим запрос на извлечение коллекции городов, численность населения в которых превышает миллион жителей. Эту задачу можно было бы решить, организовав LINQ-запрос через набор данных `DataSet`, как мы это делали в предыдущем разделе, однако мы хотим продемонстрировать и другой подход.

Вначале запустим Visual Studio 2010 и выберем проект шаблона **Windows Forms Application**, укажем имя **Name** — LinqToSqlГорода. Далее, попав в конструктор формы, из панели элементов **Toolbox** перетащим элемент управления для отображения и редактирования табличных данных **DataGridView**, на этот элемент в конечном итоге будет попадать результат запроса.

Теперь создадим базу данных SQL Server. Для этого в меню **Project** (Проект) выберем команду **Add New Item** (Добавить новый элемент). В появившемся окне выберем элемент **База данных, основанная на службах**, а в поле **Name** укажем имя базы данных Города.mdf. Далее в окне мастера настройки источника данных зададим тип модели базы данных — **Набор данных**. Затем согласимся на сохранение строки подключения в файле конфигурации приложения. Теперь после щелчка на кнопке **Готово** будет создан пустой набор данных. Этот набор данных Города.mdf теперь будет виден в окне **Solution Explorer** (Обозреватель решений).

Чтобы заполнить этот набор данных, дважды щелкнем мышью по значку **Города.mdf**; таким образом, мы попадаем в окно **Server Explorer/Database Explorer** (Обозреватель серверов/Обозреватель баз данных). Здесь в контекстном меню узла **Таблицы** выберем команду **Добавить новую таблицу**. В результате мы попадаем уже в другое окно — **dbo.Table1**, где зададим имена двух столбцов: **Город** и **Население** (рис. 11.14).

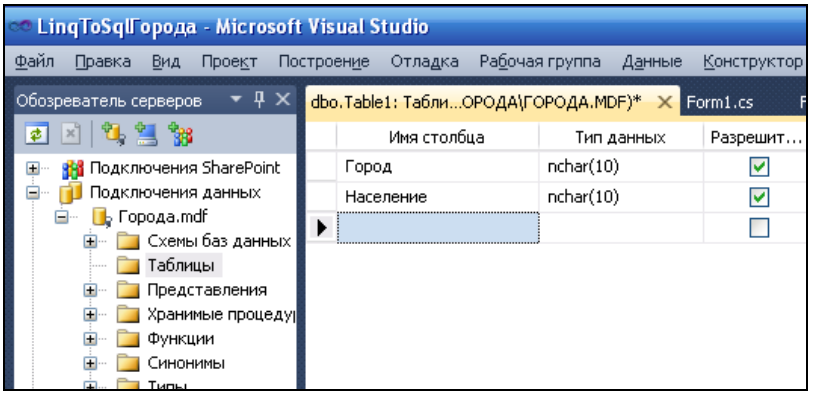


Рис. 11.14. Заказ полей таблицы в базе данных

При сохранении (<Ctrl>+<S>) пустой таблицы появится запрос на выбор имени для таблицы, здесь мы зададим имя **Города**.

Теперь будем заполнять сформированную таблицу. Для этого в **Обозревателе серверов** щелкнем правой кнопкой мыши на узле **Города** (имя нашей таблицы) и в появившемся контекстном меню выберем команду **Показать таблицу данных**. Теперь в окне **Города** мы имеем возможность заполнять нашу таблицу (рис. 11.15).

На этом этапе задача создания базы данных и заполнения в ней таблицы городов выполнена. Приступаем к организации запроса к таблице городов. Как уже указывалось ранее, LINQ-запрос можно построить через набор данных `DataSet`, а можно LINQ-запрос организовать с помощью классов LINQ to SQL. Эти классы сопоставляются с таблицами и представлениями базы данных и называются *классами сущностей* `DataContext`. Класс сущности сопоставляется с записью, а отдельные свойства класса сущности сопоставляются с отдельными столбцами, образующими запись. Сказанное, вероятно, звучит запутанно, но практически сводится к перетаскиванию мышью созданной нами таблицы Города из окна **Server Explorer**/**Database Explorer** (Обозреватель серверов/Обозреватель баз данных) на так называемый Object Relational Designer (реляционный конструктор объектов). В результате получим класс сущностей именно для нашей таблицы Города, наследованный от базового класса `DataContext`, и в тексте нашей программы уже легко сможем строить LINQ-запросы, обращаясь к объекту класса сущностей.

Чтобы получить в нашем проекте реляционный конструктор объектов, в меню **Project** выберем команду **Add New Item** (Добавить новый элемент), а в появившемся одноименном окне — шаблон (элемент) **LINQ to SQL Classes**. В поле **Name** укажем имя файла Сущности.dbml и щелкнем на кнопке **Add**. Внешний вид реляционного конструктора объектов можно увидеть на рис. 11.16.

Теперь, как мы уже говорили, просто перетаскиваем мышью таблицу Города из окна **Server Explorer/Database Explorer** (Обозреватель серверов/Обозреватель баз данных) на реляционный конструктор объектов. Реляционный конструктор объектов создает классы и применяет специфические для LINQ to SQL атрибуты, чтобы иметь функциональные возможности LINQ to SQL (возможности передачи данных

и редактирования, какие имеются у `DataContext`). А нам остается всего лишь на вкладке программного кода ввести текст, представленный в листинге 11.10.

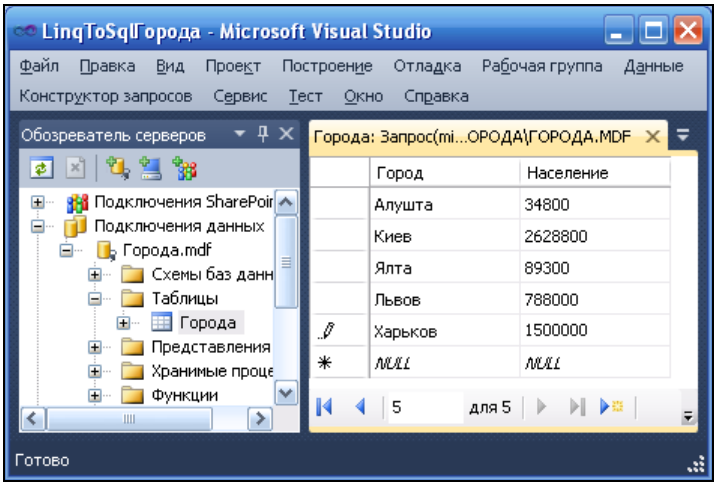


Рис. 11.15. Заполнение таблицы городов в базе данных

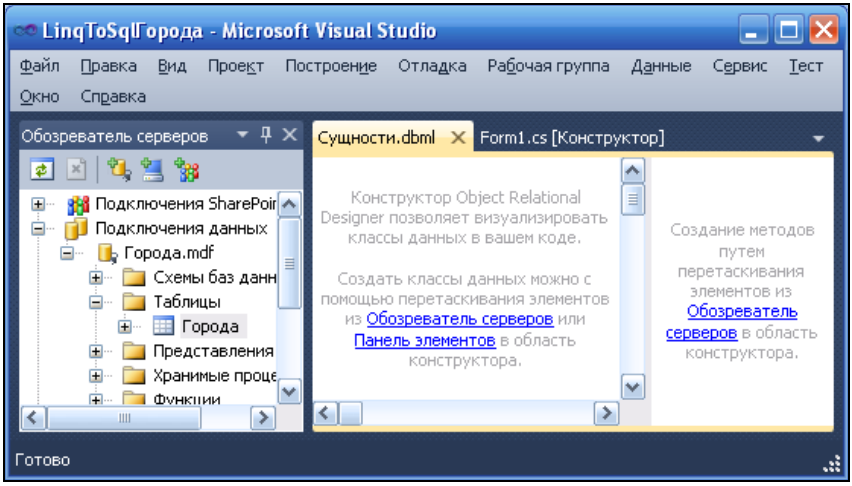


Рис. 11.16. Добавление в проект реляционного конструктора объектов

Листинг 11.10. Организация LINQ-запроса к базе данных

```
// Данное Windows-приложение состоит из экранной формы и элемента управления
// DataGridView. В программе организован LINQ-запрос к базе данных городов с
// помощью базового класса сущностей DataContext. Для этого в данную программу
// добавлен (Project | Add New Item) элемент (шаблон) "Классы LINQ to SQL",
// Name=Сущности.dbml. После связывания таблицы "Города" из базы данных
```

```
// с базовым классом сущностей (путем перетаскивания мышью таблицы
// из окна Server Explorer/Database Explorer в окно конструктора
// Object Relational Designer) автоматически был создан класс
// СущностиDataContext, производный (наследованный) от базоваго класса
// DataContext. С помощью этого класса в данной программе организован
// LINQ-запрос к базе данных на получение коллекции (списка) городов,
// численность населения в которых превышает миллион жителей. Результат
// запроса выведен на элемент управления DataGridView.
using System;
using System.Linq;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace LinqToSqlГорода
{
   public partial class Form1 : Form
   {
      // С помощью объекта базового класса DataContext будем иметь доступ
      // к таблице базы данных:
      private СущностиDataContext БД = new СущностиDataContext();
      public Form1()
      {
         InitializeComponent();
         // В результате запроса получаем коллекцию записей из таблицы базы
         // данных, удовлетворяющей условию where:
         var ГородаМлн = from города in БД.Города
                         where Convert.ToInt32(города.Население) > 1000000
                         select города;
         // или select new { города.Город, города.Население };
         // Результат запроса выводим на элемент управления DataGridView,
         // отображающий табличные данные:
         dataGridView1.DataSource = ГородаМлн;
      }
   }
}
```

Фрагмент работы программы показан на рис. 11.17.

Убедиться в работоспособности программы можно, открыв решение LinqToSqlГорода.sln папки LinqToSqlГорода.

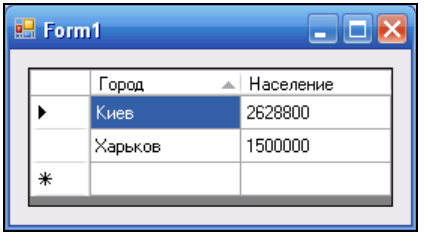


Рис. 11.17. Запрос к базе данных на города-миллионеры

Глава 12

Другие задачи, решаемые с помощью Windows Application

Пример 84. Проверка вводимых данных с помощью регулярных выражений

Данные, вводимые пользователем, должны быть проверены программой на достоверность. В этом примере мы обсудим синтаксический разбор введенной пользователем текстовой строки на соответствие ее фамилии на русском языке, а также разбор строки на соответствие ее положительному рациональному числу. Начнем с первой задачи; имеем на форме текстовое поле, метку и кнопку. В метке записано приглашение пользователю ввести фамилию на русском языке. После ввода программа должна сравнить эту строку с некоторым образцом (шаблоном, pattern) и сделать заключение, соответствует ли введенное пользователем шаблону русской фамилии.

Для решения этой задачи запустим Visual Studio 2010, выберем пункт **New Project**, закажем новый проект из шаблона **Windows Forms Application C#** и щелкнем на кнопке **OK**. Затем из панели элементов управления **Toolbox** в форму указателем мыши перетащим текстовое поле **TextBox**, метку **Label** и командную кнопку **Button**. В листинге 12.1 приведен текст программы.

Листинг 12.1. Проверка вводимой фамилии с помощью регулярных выражений

```
// Проверка данных, вводимых пользователем, на достоверность.
// Программа осуществляет синтаксический разбор введенной пользователем
// текстовой строки на соответствие ее фамилии на русском языке
using System.Text;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
```

```
namespace ПроверкаФамилии
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
         label1.Text = "Введите фамилию на русском языке:";
         button1.Text = "Проверка";
      }
      private void button1_Click(object sender, System.EventArgs e)
      {
         textBox1.Text = textBox1.Text.Trim();
         if (System.Text.RegularExpressions.Regex.Match(
             textBox1.Text, "^[А-ИК-ЩЭ-Я][а-яА-Я]*$").Success != true)
             MessageBox.Show ("Неверный ввод фамилии", "Ошибка");
      }
   }
}
```

При обработке события "щелчок мыши на кнопке" текстовое поле `textBox1.Text` обрабатывается методом `Trim`, который удаляет все пробельные символы в начале и в конце строки. Ключевым моментом программы является *проверка соответствия* введенной пользователем текстовой строки и шаблона с помощью функции `Regex.Match` (от англ. *match* — соответствовать):

```
Regex.Match(textBox1.Text, "^[А-ИК-ЩЭ-Я][а-яА-Я]*$")
```

`Match` представляет результаты из *отдельного совпадения* регулярного выражения. Как видно, мы начали регулярное выражение с символа `^` и закончили символом `$`. Символы `^` и `$` соответствуют *началу* и *концу строки* соответственно. Это побуждает регулярное выражение оценивать всю строку, а не возвращать соответствие, если успешно совпадает подстрока.

Далее в первых квадратных скобках указан диапазон допустимых букв для установления соответствия первой буквы фамилии. Первая буква должна быть прописной, т. е. верхнего регистра, в диапазоне алфавита (и таблицы символов) от А до И, от К до Щ и от Э до Я, т. е. *недопустим* ввод букв Й, Ъ, Ы, Ь в качестве первой буквы фамилии. Далее в следующих квадратных скобках указан диапазон букв либо нижнего, либо верхнего регистров, причем символ `*` означает, что второй диапазон символов может встретиться в строке ноль или более раз. Фрагмент работы программы представлен на рис. 12.1.

Вторая задача, которую мы рассмотрим в данном примере, — это проверка правильности ввода *положительного рационального числа*. Следует допустить возможность ввода любых вариантов, например "2010", "2.9", "5.", ".777". То есть допустим ввод цифровых символов и точки (или запятой).

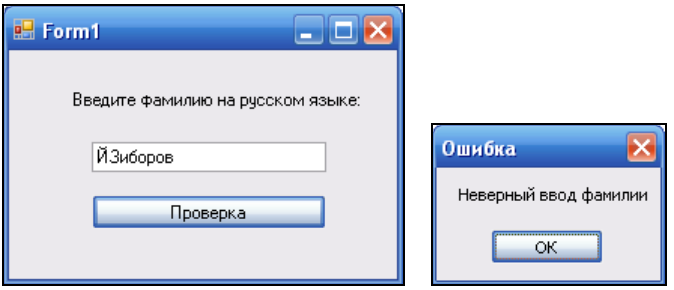


Рис. 12.1. Проверка корректности ввода фамилии на русском языке

Пользовательский интерфейс для решения данной задачи такой же, как и для предыдущей, поэтому сразу приведу программный код решения (листинг 12.2).

Листинг 12.2. Проверка вводимого числа с помощью регулярных выражений

```
// Проверка данных, вводимых пользователем, на достоверность. Программа
// осуществляет синтаксический разбор введенной пользователем текстовой
// строки на соответствие ее положительному рациональному числу.
using System;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace ПроверкаЧисла
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
         label1.Text = "Введите положительное рациональное число:";
         button1.Text = "Проверка";
      }
      private void button1_Click(object sender, EventArgs e)
      {
         textBox1.Text = textBox1.Text.Trim();
         if (System.Text.RegularExpressions.Regex.Match(textBox1.Text,
               "^(([0-9]+.[0-9]*)|([0-9]*.[0-9]+)|([0-9]+))$"
                                        ).Success == false)
            MessageBox.Show("Некорректный ввод", "Ошибка");
      }
   }
}
```

Данная программа построена аналогично предыдущей, в комментарии нуждается шаблон

```
"^(([0-9]+.[0-9]*)|([0-9]*.[0-9]+)|([0-9]+))$"
```

Здесь между символами `|`, означающими логическое ИЛИ, между круглыми скобками представлены *три группы* выражений. *Первая группа* допускает ввод цифровых символов от 0 до 9 до десятичной точки. Знак "плюс" (`+`) означает, что цифровой символ может встретиться в строке один или более раз. Символ `*` означает, что цифровой символ может встретиться в строке ноль или более раз. Таким образом, первая группа допускает ввод, например, рационального числа `6.`. Аналогично работает *вторая группа* выражений, она допускает ввод, например, числа `.77`. *Третья группа* — это совсем просто: она проверяет соответствие с любыми целыми числами.

Убедиться в работоспособности программ, рассмотренных в данном примере, можно, открыв решение ПроверкаФамилии.sln в папке ПроверкаФамилии и ПроверкаЧисла.sln в папке ПроверкаЧисла.

Пример 85. Управление прозрачностью формы

Создадим программу, которая демонстрирует стандартное Windows-окно, т. е. стандартную форму. Щелчок мышью в пределах этой формы *начинает постепенный процесс* исчезновения формы, форма становится все более прозрачной, а затем исчезает вовсе, далее она постепенно проявляется снова, и т. д. Еще один щелчок в пределах формы останавливает этот процесс, а следующий щелчок процесс возобновляет и т. д.

Для написания этой программы добавим в стандартную форму из панели элементов управления **Toolbox** объект **Timer** (Таймер). Этот невидимый во время работы программы элемент управления предназначен для периодического генерирования события `Tick`, которое происходит, когда таймер работает и прошел заданный интервал времени. Этот интервал времени `timer1.Interval` по умолчанию равен 100 миллисекунд, его в нашей программе мы изменять не будем.

Управлять прозрачностью формы можно с помощью свойства формы `Opacity`, задавая уровень непрозрачности от нуля до единицы. Текст программы приведен в листинге 12.3.

Листинг 12.3. Управление прозрачностью формы

```
// Программа демонстрирует стандартную форму. Щелчок мышью в пределах
// этой формы начинает постепенный процесс исчезновения формы: форма
// становится все более прозрачной, а затем исчезает вовсе. Далее она
// постепенно проявляется снова, и т. д. Еще один щелчок в пределах формы
// останавливает этот процесс, а следующий щелчок процесс возобновляет и т. д.
```

```
using System;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace Opacity
{
   public partial class Form1 : Form
   {
      double s = 0.1;
      public Form1()
      {
         InitializeComponent();
         // Timer1.Interval() = 400
      }
      private void timer1_Tick(object sender, EventArgs e)
      {
         if (this.Opacity <= 0 || this.Opacity >= 1) s = -s;
         this.Opacity += s;
      }
      private void Form1_Click(object sender, EventArgs e)
      {  timer1.Enabled = !timer1.Enabled; }
   }
}
```

Как видно, здесь при обработке события "щелчок в пределах формы" запускается таймер, а еще один щелчок мышью его останавливает. Каждые 100 миллисекунд появляется событие `timer1_Click`. Обрабатывая его, мы меняем значение непрозрачности `Opacity` от нуля до единицы с шагом `0.1`, который задаем через внешнюю переменную `s`.

Этот пример на языке Visual Basic приведен на сайте **http://subscribe.ru/archive/comp.soft.prog.visualbnet/200512/16181816.html**, автор переписал его на C#. Текст этой программы можно посмотреть, открыв решение Opacity.sln в папке Opacity.

Пример 86. Время по Гринвичу в полупрозрачной форме

Время, дату, день недели очень легко выяснить, посмотрев в правый нижний угол рабочего стола Windows. Однако, если, например, вы читаете новости, которые поступают в реальном времени, где время публикаций указывают по Гринвичу (GMT — Greenwich Meridian Time), это могут быть, например, новости валютного

рынка или фондового рынка Форекс или другой экономический календарь, то в этом случае важно знать, насколько актуальна, свежа новость. Конечно, можно держать в голове, что московское время отличается от времени по Гринвичу на 4 часа, а время по Киеву — на 3 часа, а затем мучительно соображать, необходимо прибавить эти 3 часа к киевскому времени или, наоборот, отнять.

Кроме того, следует помнить, что время по Гринвичу не переводится на летние и зимние часы. Поэтому, чтобы выяснить правильное время по Гринвичу, можно на стандартном рабочем столе Windows справа внизу двойным щелчком на отображении текущего местного времени на вкладке **Часовой пояс** выбрать **Время по Гринвичу** и сбросить флажок **Автоматический переход на летнее время и обратно**. При этом очень удобно для сравнения времени иметь перед глазами сайт **www.central-european-time.com**, где показано текущее местное время, среднеевропейское (или центральноевропейское время, Central European Time (CET)) и время по Гринвичу.

Предлагаю написать маленькую программу, которая в полупрозрачной экранной форме отображает текущее время по Гринвичу. То есть мы напишем программу, которая будет демонстрировать текущее время по Гринвичу и при этом, в силу своей прозрачности, не будет заслонять другие приложения.

Для решения этой задачи создадим форму и перетащим в нее из панели элементов управления **Toolbox** метку **Label** и объект **Timer**. Этот объект позволит обрабатывать событие `timer1.Tick` через периодически заданные интервалы времени. Далее на вкладке визуального проекта программы **Form1.cs [Design]** растянем мышью форму и расположим метку примерно так, как показано на рис. 12.2.

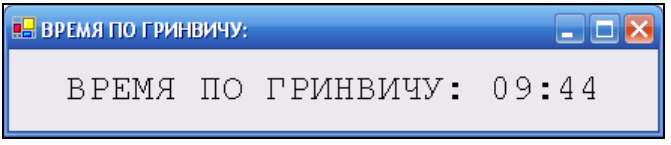


Рис. 12.2. Фрагмент работы программы определения времени по Гринвичу

Код данной программы представлен в листинге 12.4.

Листинг 12.4. Время по Гринвичу в полупрозрачной форме

```
// Программа в полупрозрачной экранной форме отображает текущее время
// по Гринвичу. Таким образом, программа демонстрирует текущее время
// по Гринвичу и при этом не заслоняет собой другие приложения.
using System;
using System.Drawing;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace Гринвич
```

```
{
   public partial class Form1 : Form
   {
      bool t = false;
      public Form1()
      {
         InitializeComponent();
         base.Text = "ВРЕМЯ ПО ГРИНВИЧУ:";
         base.Opacity = 0.75;    // Уровень непрозрачности формы
         label1.Font = new Font("Courier New", 18.0F);
         label1.Text = string.Empty;
         timer1.Interval = 1000; // 1000 миллисекунд = 1 секунда
         timer1.Start();
      }
      private void timer1_Tick(object sender, EventArgs e)
      {  // Обработка события, когда прошел заданный интервал
         // времени: 1000 миллисекунд = 1 секунда
         label1.Text = "ВРЕМЯ ПО ГРИНВИЧУ: ";
         string время;
         t = !t;   // То же, что и t = true ^ t;
         if (t == true) время = string.Format("{0:t}", DateTime.UtcNow);
         else время = string.Format("{0} {1:00}", DateTime.UtcNow.Hour,
                                                  DateTime.UtcNow.Minute);
         label1.Text = label1.Text + время;
      }
      private void Form1_MouseLeave(object sender, EventArgs e)
      {  // Указатель мыши выходит за пределы формы
         base.Opacity = 0.75;
      }
      private void label1_MouseEnter(object sender, EventArgs e)
      {  // Указатель мыши входит в область метки
         base.Opacity = 1;
      }
   }
}
```

Здесь сразу после выполнения процедуры `InitializeComponent` задаем текст "ВРЕМЯ ПО ГРИНВИЧУ:" в заголовке формы (свойство `base.Text`), указываем уровень непрозрачности формы `base.Opacity = 0.75`. Если `Opacity = 0`, то изображение формы совсем пропадает, программа будет благополучно работать в оперативной памяти, и мы будем догадываться о ее работе только потому, что она будет напоминать о своем существовании в свернутом виде в панели задач. При `Opacity = 1` будем иметь обычное непрозрачное изображение формы.

Далее включаем таймер `timer1.Start()` и задаем интервал работы таймера `timer1.Interval = 1000` (1000 миллисекунд, т. е. 1 секунда). Это означает, что через каждую секунду мы имеем возможность обрабатывать событие "прошла одна секунда" `timer1.Tick`.

Справиться с задачей демонстрации времени было бы очень легко, если бы нам надо было просто отображать текущее время. Мы бы написали:

```
время = string.Format("{0:t}", DateTime. Now)
```

Однако по условию задачи нам нужно показывать время по Гринвичу. Система Visual Studio 2010 имеет свойство `UtcNow` пространства имен `System.DateTime`. Это свойство возвращает так называемое *универсальное координированное время* (Universal Coordinated Time, UTC), которое с точностью до долей секунды совпадает со временем по Гринвичу. Поэтому выражение

```
время = string.Format("{0:t}", DateTime.UtcNow);
```

копирует в строку `время` значение текущего времени Гринвичского меридиана.

Далее, чтобы подчеркнуть, что время идет, между значением часа и значением минут зададим двоеточие (`:`), которое появляется в течение одной секунды и исчезает тоже на одну секунду. Для этого у нас есть внешняя булева переменная `t`, которая периодически меняет свое значение с `true` на `false`, потом с `false` на `true` и т. д., т. е. на противоположное: `t = !t`. Таким образом, при обработке события `Tick` (события, когда прошел заданный интервал времени `Interval`, равный 1000 миллисекунд = 1 секунда) объекта `timer1` в метку `label1` копируется новое значение времени то с двоеточием, то без него.

Как видно, в тексте данной программы использованы переменные с русскими именами, что добавило выразительности программному коду. Мы реализовали возможность становиться форме непрозрачной при наведении курсора мыши в область метки при обработке события `label1_MouseEnter` вхождения курсора мыши в границы метки. Из соображений эстетики (что, конечно же, может быть подвергнуто сомнению) мы запрограммировали возврат опять к прозрачной форме при выходе курсора мыши за пределы *формы*, а не *метки*, т. е. при наступлении события `Form1_MouseLeave`.

Текст этой программы можно посмотреть, открыв решение Гринвич.sln в папке Гринвич.

Пример 87. Ссылка на процесс, работающий в фоновом режиме, в форме значка в области уведомлений

Название раздела для новичка кажется чрезвычайно "заумным". На самом деле ничего сложного, как и многое другое в области компьютерных технологий, если разобраться в сути слов и попробовать практически испытать работу элементов этих технологий. В данном разделе речь идет о программах, работающих на ком-

пьютере *в фоновом режиме* при использовании, например, антивирусных программ или элемента управления громкостью. Такие программы внешне напоминают о своем существовании лишь присутствием *значка в области уведомлений* панели задач в правом нижнем углу экрана. Кстати программу "Время по Гринвичу в полупрозрачной форме", рассмотренную в предыдущем разделе, уместно было бы запрограммировать именно со значком в области уведомлений. Однако чтобы замысел одного примера не смешивался с замыслом другого, а программный код при этом выглядел *компактно* и *выразительно*, эти программы приведены в разных примерах.

Рассматриваемая в данном разделе программа также способна работать в фоновом режиме. Внешне программа имеет форму с меткой, где записаны возможности программы, командную кнопку, при щелчке на которой форма исчезает, но зато появляется значок в области уведомлений. При этом программа в оперативной памяти существует и работает в фоновом режиме. При щелчке правой кнопкой мыши на этом значке появляется *контекстное меню* из двух пунктов. Один пункт меню сообщает пользователю время работы данного компьютера с тех пор, как стартовала операционная система, а другой пункт меню обеспечивает выход из данной программы. При двойном щелчке на значке появляется опять исходная форма, а значок исчезает. Таким образом, проектируемая программа способна переводить свою работу в фоновый режим и обратно.

Для программирования данной задачи запустим Visual Studio 2010, выберем новый проект **Windows Forms Application C#.** В конструкторе формы из панели элементов перенесем в форму метку, командную кнопку, элемент управления **NotifyIcon** и контекстное меню **ContextMenuStrip**. Щелкая на значке **ContextMenuStrip**, спроектируем два пункта контекстного меню: **Время старта ОС** и **Выход**. Теперь перейдем на вкладку программного кода, где напишем текст программы, представленный в листинге 12.5.

Листинг 12.5. Перевод работы программы в фоновый режим и обратно

```
// Эта программа сообщает пользователю время, прошедшее с момента старта
// операционной системы на данном компьютере. Доступ к этой информации
// реализован через контекстное меню значка в области уведомлений панели задач
using System;
using System.Drawing;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace Значок_в_области_уведомлений
{  // Из панели элементов следует перенести в форму Button, Label,
   // NotifyIcon и ContextMenuStrip
   public partial class Form1 : Form
   {
      public Form1()
```

```
{
   InitializeComponent();
   base.Text = "Создаю значок в области уведомлений";
   // Задаем размер метки
   label1.Size = new System.Drawing.Size(292, 90);
   // Разрешаем продолжение текста в метке с новой строки
   label1.AutoSize = false;
   label1.Text =
     "При щелчке на командной кнопке данная программа " +
     "размещает значок в область уведомлений. Щелкните " +
     "правой кнопкой мыши на этом значке для доступа к " +
     "контекстному меню с пунктами \"Время\"" +
     "работы ОС\" и \"Выход\"" +
     ". Двойной щелчок на значке возвращает " +
     "на экран данную форму.";
   button1.Text = "Разместить значок в область уведомлений";
   //             SystemIcons - это стандартные значки Windows
   notifyIcon1.Icon = SystemIcons.Shield;   // значок щита
   //             SystemIcons.Information - значок сведений
   notifyIcon1.Text = "Время работы ОС";
   notifyIcon1.Visible = false;
   // "Привязываем" контекстное меню к значку notifyIcon1
   notifyIcon1.ContextMenuStrip = contextMenuStrip1;
}

private void button1_Click(object sender, EventArgs e)
{  // Скрыть экранную форму и сделать видимым значок
   // в области уведомлений
   base.Hide(); notifyIcon1.Visible = true;
}

private void notifyIcon1_MouseDoubleClick(object sender,
                                          MouseEventArgs e)
{  // Чтобы получить пустой обработчик этого события, можно, например,
   // в конструкторе формы дважды щелкнуть на значке notifyIcon1
   notifyIcon1.Visible = false; base.Show();
}

private void времяСтартаОСToolStripMenuItem_Click(object sender,
                                                  EventArgs e)
{
   double Время_работы_ОС_в_минутах =
                   System.Environment.TickCount / 1000 / 60;
```

```
            // Формат "{0:F0}" округляет Double до целого значения
            string ss = String.Format("ОС стартовала {0:F0} минут назад",
                                      Время_работы_ОС_в_минутах);
            MessageBox.Show(ss);
        }

        private void выходToolStripMenuItem_Click(object sender, EventArgs e)
        {  notifyIcon1.Dispose(); Application.Exit();  }
    }
}
```

Как видно из программного кода, сразу после запуска процедуры `InitializeComponent` выполняем очевидные инициализации свойств элементов управления, в том числе для компонента `notifyIcon1`. Для него среди стандартных значков Windows выбираем *значок щита* (`notifyIcon1.Icon = SystemIcons.Shield`), а сам значок делаем невидимым (`notifyIcon1.Visible = false`). При обработке события "щелчок на командной кнопке" скрываем экранную форму и делаем видимым значок в области уведомлений. Обработка события "двойной щелчок" на этом значке обеспечивает *противоположное действие*, а именно показывает форму, но скрывает значок. Две последние процедуры обрабатывают выбор пунктов меню **Время старта ОС** и **Выход**. На рис. 12.3 и 12.4 приведены фрагменты работы программы в обычном и фоновом режимах.

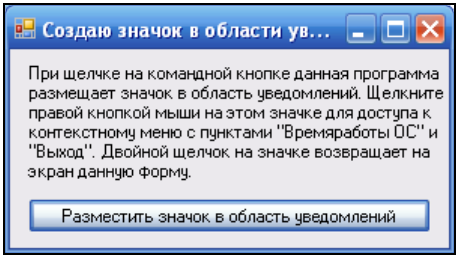


Рис. 12.3. Внешний вид формы при работе программы в обычном режиме

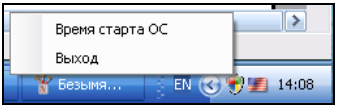


Рис. 12.4. Контекстное меню при работе программы в фоновом режиме

Убедиться в работоспособности данной программы можно, открыв соответствующее решение в папке Значок_в_области_уведомлений.

Пример 88. Нестандартная форма. Перемещение формы мышью

Обычно экранную форму, любое Windows-окно мы перемещаем при нажатой левой кнопке мыши, "зацепив" за заголовок, т. е. за синюю полосу вверху окна.

В заголовке расположены кнопки **Свернуть**, **Свернуть в окно** и **Закрыть**. Для демонстрации работы с событиями мыши, с аргументами процедуры обработки этих событий `MouseEventArgs` напишем программу, с помощью которой появляется возможность перемещать форму, "зацепив" ее указателем мыши в любой части формы, а не только за заголовок. Подобный пример, только на языке Visual Basic, приведен в книге "Visual Basic .NET. Библия пользователя"[1].

Итак, запускаем Visual Studio 2010, далее заказываем новый проект шаблона **Windows Forms Application C#** и получаем таким образом стандартную экранную форму. Далее вводим текст программы, приведенный в листинге 12.6.

Листинг 12.6. Перемещение формы мышью

```
// Нестандартная форма. Программа позволяет перемещать форму мышью,
// "зацепив" ее не только за заголовок, а в любом месте формы
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace ПеремещениеФормы
{
   public partial class Form1 : Form
   {
      bool Moving = false;
      // Перемещаем форму только тогда, когда Moving = true
      int MouseDownX;
      int MouseDownY;
      public Form1()
      {
         InitializeComponent();
      }

      private void Form1_MouseDown(object sender, MouseEventArgs e)
      {  // Здесь обрабатываем событие "щелчок любой кнопкой мыши". Во время
         // щелчка левой копкой мыши запоминаем текущее положение мыши
         if (e.Button == MouseButtons.Left)
         {
            Moving = true;
            MouseDownX = e.X;
            MouseDownY = e.Y;
         }
      }
```

[1] Ивьен Б., Берес Дж. Visual Basic .NET. Библия пользователя. — М.: Вильямс, 2002. — С. 591.

```
      private void Form1_MouseUp(object sender, MouseEventArgs e)
      {  // Здесь обрабатываем событие, когда
         // пользователь отпустил кнопку мыши
         if (e.Button == MouseButtons.Left) Moving = false;
      }

      private void Form1_MouseMove(object sender, MouseEventArgs e)
      {  // Здесь обрабатываем событие, когда указатель
         // мыши перемещается в пределах формы.
         // Перемещаем форму только тогда, когда Moving = true.
         if (Moving == true)
         {
            var temp = new System.Drawing.Point();
            temp.X = base.Location.X + (e.X - MouseDownX);
            temp.Y = base.Location.Y + (e.Y - MouseDownY);
            base.Location = temp;
         }
      }
   }
}
```

Вначале программы объявлены три внешние переменные, видимые в пределах класса `Form1`. Одна из переменных — булева переменная `Moving` — это как бы флажок, который сигнализирует, следует ли перемещать экранную форму. Если пользователь нажал левую кнопку мыши (событие `MouseDown`, `e.Button == Left`), то `Moving = true`, т. е. следует перемещать форму. Если пользователь отпустил кнопку мыши (событие `MouseUp`), то `Moving = false`, т. е. форму перемещать не следует. Остальные две переменные целого типа `int` предназначены для запоминания текущего положения (`e.X` и `e.Y`) мыши. Перемещаем форму только тогда, когда перемещается указатель мыши (событие `MouseMove`), и при этом флажок `Moving = true`.

Убедиться в работоспособности программы можно, открыв решение ПеремещениеФормы.sln в папке ПеремещениеФормы.

Пример 89. Проигрыватель Windows Media Player 11

Используя возможность подключения к проекту соответствующей библиотеки, можно написать программу для проигрывания различных файлов мультимедиа AVI, MPEG, MP3, MP4, FLV, WMV и других форматов. Причем ядром вашей программы будет, например, самый современный плеер Windows Media Player вер-

сии 11. Но пользовательский интерфейс, т. е. способ управления плеером: кнопки на форме или в виде выпадающего меню, цвет оболочки, значения параметров по умолчанию, размер экрана и проч., вы можете создать свой собственный.

Итак, запустим Visual Studio 2010 и закажем новый проект. Назовем этот новый проект Player. Далее в окне **New Project** выберем шаблон **Windows Forms Application C#** и нажмем кнопку **OK**. Чтобы добавить на панель **Toolbox** элемент управления **Windows Media Player**, следует подключить библиотеку Windows Media Player. Для этого, щелкая правой кнопкой мыши в пределах панели элементов, в контекстном меню выберем пункт **Choose Items**. Далее на вкладке **COM Components** установим флажок **Windows Media Player** и щелкнем на кнопке **OK**. При этом на панели элементов появится значок **Windows Media Player**. Перетащите этот значок в форму. Растяните изображения формы и плеера так, как вам представляется это привлекательным. После этого в папках проекта bin\Debug\ и \obj\x86\Debug\ должны появиться два файла: AxInterop.WMPLib.dll и Interop.WMPLib.dll. В этих двух файлах находятся динамически вызываемые объекты для работы элемента управления **Windows Media Player**.

Далее для программирования меню перенесем с панели **Toolbox** в форму значок **MenuStrip**. Слева вверху формы появится проект первого пункта меню **Type Here**. Здесь мы проектируем пункты меню будущей программы. Меню **Файл** имеет пункты **Открыть** и **Выход**. А меню **Сервис** содержит пункты, показанные на рис. 12.5.

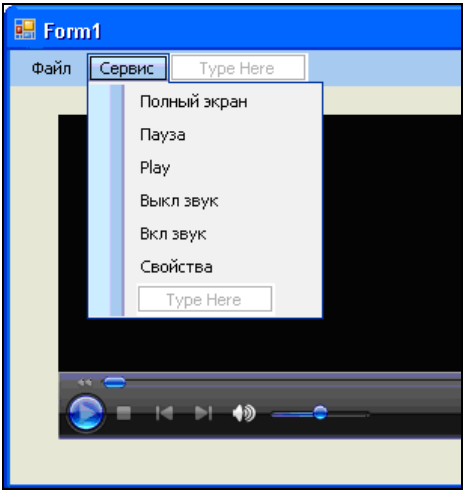


Рис. 12.5. Проектирование пунктов меню

Программный код приведен в листинге 12.7.

Листинг 12.7. Проигрыватель Windows Media Player 11

```
// Программа реализует функции проигрывателя Windows Media Player 11
using System;
```

```
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace Player
{
   public partial class Form1 : Form
   {
      OpenFileDialog openFileDialog1 = new OpenFileDialog();
      public Form1()
      {
         InitializeComponent();
         // ВЕРСИЯ ПЛЕЕРА
         base.Text = "Windows Media Player, версия = " +
                     axWindowsMediaPlayer1.versionInfo;
      }

      private void открытьToolStripMenuItem_Click(object sender, EventArgs e)
      {  // ПУНКТ МЕНЮ "Открыть".
         // Пользователь выбирает файл:
         openFileDialog1.ShowDialog();
         // Передача плееру имени файла
         axWindowsMediaPlayer1.URL = openFileDialog1.FileName;
         // axWindowsMediaPlayer1.URL = @"C:\WINDOWS\Media\tada.wav";
         // Команда на проигрывание файла
         axWindowsMediaPlayer1.Ctlcontrols.play();
         // ИЛИ ТАК: передача имени файла и сразу PLAY
         // axWindowsMediaPlayer1.openPlayer(openFileDialog1.FileName);
      }

      private void выходToolStripMenuItem_Click(object sender, EventArgs e)
      {  // ПУНКТ МЕНЮ "Выход"
         Application.Exit();
      }

      private void полныйЭкранToolStripMenuItem_Click(object sender,
                                                      EventArgs e)
      {  // ПУНКТ МЕНЮ "Полный экран".
         // Если плеер пребывает в состоянии PLAY, то можно
         // перейти в режим полного экрана:
         if (axWindowsMediaPlayer1.playState ==
                                  WMPLib.WMPPlayState.wmppsPlaying)
            axWindowsMediaPlayer1.fullScreen = true;
      }
```

```
        private void паузаToolStripMenuItem_Click(object sender, EventArgs e)
        {  // ПУНКТ МЕНЮ "Пауза"
           axWindowsMediaPlayer1.Ctlcontrols.pause();
        }

        private void playToolStripMenuItem_Click(object sender, EventArgs e)
        {  // ПУНКТ МЕНЮ "Play"
           axWindowsMediaPlayer1.Ctlcontrols.play();
        }

        private void выклЗвукToolStripMenuItem_Click(object sender, EventArgs e)
        {  // ПУНКТ МЕНЮ "Выкл звук"
           axWindowsMediaPlayer1.settings.mute = true;
        }

        private void вклЗвукToolStripMenuItem_Click(object sender, EventArgs e)
        {  // ПУНКТ МЕНЮ "Вкл звук"
           axWindowsMediaPlayer1.settings.mute = false;
        }

        private void свойстваToolStripMenuItem_Click(object sender, EventArgs e)
        {  // ПУНКТ МЕНЮ "Свойства"
           axWindowsMediaPlayer1.ShowPropertyPages();
        }
    }
}
```

Как видно из текста программы, перед загрузкой формы объявлено создание экземпляра класса `OpenFileDialog`. Этот объект можно было бы перенести в форму из панели элементов управления, как мы это делали в примере 25 *(см. главу 4)*, а можно его создать программно, как в данном программном коде.

Сразу после выполнения процедуры `InitializeComponent` задаем текст строки заголовка формы "Windows Media Player, версия = ". При этом номер версии получаем из свойства `versionInfo`.

При обработке события "щелчок мышью по пункту меню" **Открыть** функция `ShowDialog` обеспечивает выбор нужного файла мультимедиа. Передача плееру имени выбранного файла происходит через свойство плеера `URL`. Далее следует команда на проигрывание файла, хотя можно было бы предусмотреть эту команду в отдельном пункте меню. В комментарии приведена возможность проигрывания файла мультимедиа при передаче имени этого файла одной функцией `openPlayer`.

Далее по тексту программы реализованы очевидные функции в соответствующих обработчиках событий: переход в режим полного экрана (это возможно толь-

ко, если плеер пребывает в состоянии проигрывания файла `wmppsPlaying`), `Пауза`, `Play`, `ВклЗвук`, `ВыклЗвук`, вызов меню **Свойства**, выход из программы.

Убедиться в работоспособности программы можно, открыв решение Player.sln в папке Player.

Пример 90. Программирование контекстной справки. Стандартные кнопки в форме

Напишем программу, демонстрирующую организацию помощи пользователю вашей программы. В данной программе предусмотрена экранная форма, которая в заголовке имеет кнопку **Справка** (в виде вопросительного знака) и кнопку **Закрыть**. Здесь реализована контекстная помощь, когда после щелчка мыши на кнопке **Справка** можно получить контекстную всплывающую подсказку по тому или иному элементу управления, находящемуся в форме.

Для этого обычным путем создадим новый проект из шаблона **Windows Forms Application C#**, получим стандартную форму. Используя панель **Toolbox**, добавим в форму три текстовых поля, три метки и две кнопки, как показано на рис. 12.6.

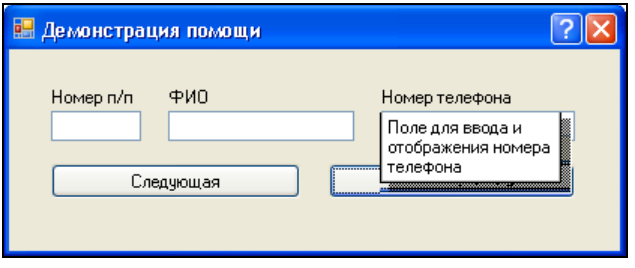


Рис. 12.6. Фрагмент работы программы с контекстной справкой

Таким образом, мы получили пользовательский интерфейс редактирования таблицы телефонов, знакомый нам из предыдущих разделов.

На вкладке программного кода напишем следующий текст (листинг 12.8).

Листинг 12.8. Программирование контекстной справки

```
// В программе предусмотрена экранная форма, которая в заголовке имеет только
// кнопку Справка (в виде вопросительного знака) и кнопку Закрыть. Здесь
// реализована контекстная помощь, когда после щелчка мыши на кнопке Справка
// можно получить контекстную всплывающую подсказку по тому или иному элементу
// управления, находящемуся в форме.
using System.Windows.Forms;
```

```
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace help
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
         MaximizeBox = false; // - отмена кнопки Развернуть
         MinimizeBox = false; // - отмена кнопки Свернуть
         // Чтобы стиль формы содержал кнопку помощи,
         // т. е. вопросительный знак
         HelpButton = true;
         this.Text = "Демонстрация помощи";
         button1.Text = "Следующая"; button2.Text = "Предыдущая";
         textBox1.Clear(); textBox2.Clear(); textBox3.Clear();
         label1.Text = "Номер п/п"; label2.Text = "ФИО";
         label3.Text = "Номер телефона";
         var helpProvider1 = new HelpProvider();
         helpProvider1.SetHelpString(textBox1,
                              "Здесь отображаются номера записи по порядку");
         helpProvider1.SetHelpString(textBox2,
                              "Поле для редактирования имени абонента");
         helpProvider1.SetHelpString(textBox3,
                              "Поле для ввода и отображения номера телефона");
         helpProvider1.SetHelpString(button1,
                              "Кнопка для перехода на следующую запись");
         helpProvider1.SetHelpString(button2,
                              "Кнопка для перехода на предыдущую запись");
         // Назначаем, какой help-файл будет вызываться
         // при нажатии клавиши <F1>
         helpProvider1.HelpNamespace = "mspaint.chm";
      }
   }
}
```

Здесь при обработке события загрузки формы мы программируем всю помощь пользователю. Так, чтобы стиль формы содержал кнопку **Справка** (вопросительный знак), следует запретить в форме кнопки **Развернуть** и **Свернуть**:

```
MaximizeBox = false;
MinimizeBox = false;
```

Затем вместо них задать кнопку **Справка:** `HelpButton = true`. Далее объявляем объект `helpProvider1`, который мы могли бы добавить в форму, используя **Toolbox**. Затем для каждого из текстовых полей и кнопок задаем текст контекстной подсказки. Эта подсказка будет работать так же, как и подсказка **ToolTip** (см. пример 8 в *главе 1*). Свойству `HelpNamespace` назначаем help-файл, который будет вызываться при нажатии функциональной клавиши <F1>. Мы указали файл mspaint.chm, который является справочным для графического редактора MS Paint, встроенного в ОС Windows, т. е. он имеется на любом компьютере под управлением Windows. Формат CHM был разработан Microsoft для гипертекстовых справочных систем. У CHM-файлов, как правило, есть **Содержание** — отдельная панель со списком статей для упрощения навигации. В наличии содержания, пожалуй, и заключается главное отличие CHM-файлов от использовавшихся ранее HLP-файлов справки Windows.

Убедиться в работоспособности программы можно, открыв решение help.sln в папке Help.

Создание инсталляционного пакета для распространения программы

Допустим, вы отладили свою первую (или очередную) программу, и теперь вам надо *создать дистрибутив* (инсталляционный пакет) для распространения программы (дистрибуции). Инсталляционный пакет будет включать в себя набор файлов (setup.exe, *.msi, *.dll, *.tlb и т. д.). Пользователь вашей программы должен запустить setup.exe, и в режиме диалога будет происходить распаковка msi-архива, регистрация в системном реестре и проч. Для создания таких инсталляционных пакетов существует множество программ-инсталляторов, например InstallShild, InnoSetup и др. Однако можно воспользоваться непосредственно системой Visual Studio 2010. Покажем процесс создания инсталляционного пакета на примере программы управления функциями AutoCAD (см. пример 60 в *главе 9*).

Запустим систему Visual Studio 2010, закажем новый проект, в окне **New Project** раскроем узел **Other Project Types** и узел **Setup and Deployment** (Установка и распространение), а здесь выберем пункт **Visual Studio Installer** и шаблон **Setup Project**. В строке **Location** укажем папку, в которой будет располагаться наш инсталляционный пакет, например папку C:\New, и нажмем кнопку **OK**. Далее в пункте меню **Project** выберем команду **Add | File**. Здесь добавим exe-файл и прочие файлы, которые мы хотим поместить в msi-архив инсталляционного пакета, в нашем случае это файл ACADm.exe, также добавим файл Interop.AutoCAD.dll. При этом в окне **Solution Explorer** (Обозревателе решений) мы увидим зависимые от exe-файла ссылки на tlb-, dll- и msm-файлы. В нашем случае в окне **Solution Explorer** появится файл ACAD.TLB. Можно также добавить еще необходимые на ваш взгляд компоненты, например какой-либо ocx-файл или тестовый файл с какими-

либо пояснениями. После создания дистрибутива все эти файлы окажутся внутри msi-архива.

Теперь дадим команду на непосредственное создание инсталляционного пакета **Build | Build Setup1**. После завершения данного процесса в папке C:\New\Debug появятся файлы Setup.exe и Setup1.msi. Эти файлы скопируем на какой-либо съемный носитель, например на компакт-диск или флэш-накопитель. Теперь инсталляционный пакет готов для распространения.

На компьютере вашего пользователя запустим setup.exe, далее в диалоге укажем папку, например C:\Program Files\Acadm\, и в конце диалога получим сообщение от системы о завершении инсталляции. Теперь ваша программа находится в указанной папке, в ней — exe-файл и все связанные с ним файлы dll- и tlb-файлы. Программа зарегистрирована в системном реестре, и вы можете найти ее среди установленных программ в Панели управления в разделе **Установка и удаление программ**.

Глава 13

Программирование простейших Web-ориентированных приложений на Visual C# 2010

Создание Web-страницы на языке HTML. Интернет-технологии

HTML (HyperText Markup Language) — это язык для отображения информации о Web-странице в браузере: текста, изображений, анимации, ссылок на другие Web-страницы. Это язык специальных инструкций браузеру, но его нельзя назвать языком программирования в обычном понимании этого слова, поскольку он не позволяет использовать логику, с его помощью невозможно выполнить даже элементарные арифметические операции. Мы уже приводили примеры HTML-кода для отображения таблиц. В качестве примера в листинге 13.1 представлен HTML-код простейшей Web-страницы, содержащей GIF-изображение и гиперссылку для перехода на другую Web-страницу.

Листинг 13.1. HTML-код простейшей Web-страницы

```
<html>
  <head>
    <title>Простейшая Web-страница</title>
  </head>
  <body>
    <p><img border="0" src="image1.gif" width="317" height="73" /></p>
    <p><a href="http://www.mail.ru/">Перейти на почтовый сервер</a></p>
  </body>
</html>
```

Здесь тег `<html>` описывает ваш документ как Web-страницу, выполненную в формате HTML. Тег `<head>` указывает на наличие текста, содержащего наименова-

ние страницы и дополнительные сведения о ней. В раздел <head> обычно вложен тег <title> для обозначения наименования страницы в строке заголовка. Наименования Web-страниц отображаются в строке заголовка окна браузера. Затем следует тег <body>, в котором размещаются: весь остальной текст, изображения, таблицы, гиперссылки и другие элементы содержимого Web-страницы. Тег <p> служит для обозначения начала нового абзаца.

Для вставки графических элементов служит тег <img>. Файл изображения image1.gif создан с помощью MS Office FrontPage. В этой программе автор задал художественно оформить текст "Весело живем!" (объект WordArt). В ответ FrontPage сгенерировал файл image1.gif. Объект WordArt можно также получить в текстовом редакторе MS Word через пункты меню **Вставка | WordArt.**

Для создания гиперссылки служит тег <a>. Синтаксис этого тега очевиден. Данный HTML-код записан с помощью Блокнота в файл primer.htm. На рис. 13.1 приведено отображение этого файла в браузере Internet Explorer.

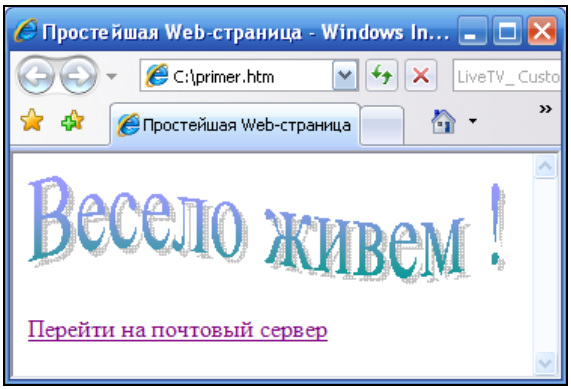


Рис. 13.1. Отображение графического файла в браузере Internet Explorer

Чтобы создавать подобные Web-страницы, вовсе не обязательно знать язык разметки HTML. Можно пользоваться каким-либо редактором Web-страниц, например Microsoft Office FrontPage или Microsoft Office SharePoint Designer. Здесь интерфейс для создания Web-страницы очень похож на интерфейс текстового редактора MS Word. В крайнем случае, содержание Web-страницы можно набрать непосредственно в редакторе MS Word, но сохранить файл с расширением htm и таким образом получить Web-страницу. Кстати говоря, создавать Web-страницы на языке HTML можно также с помощью Visual Studio, для этого следует создать новый проект шаблона **Empty ASP.NET Web Application** и к текущему проекту добавить новый элемент (**Project | Add New Item**) шаблона **HTML page.**

Однако Web-страница, написанная только на языке разметки HTML, будет *статичной*, она способна лишь отображать какие-либо сведения. Вы знаете, что современная Web-страница является, по сути, программой, способной реагировать на действия пользователя, обеспечивать полноценный интерактивный режим, диалог с пользователем, работу с базой данных и т. д. Реализовать такую интерактивность

можно, например, используя технологию Microsoft ASP.NET, в частности язык Visual C# 2010. Технология ASP (Active Server Pages, активные страницы сервера) представляет альтернативу статичных страниц, основанных на "чистом" HTML.

Web-хостинг на платформах UNIX и Windows

Интернет можно рассматривать как совокупность связанных между собой специализированных компьютеров, включенных 24 часа в сутки, называемых *хостами* (от англ. *host* — хозяин, принимающий гостей) или *серверами*. Пользователи, представляющие локальные компьютеры, подключаются через провайдеров к Интернету, могут, используя URL-адреса, запрашивать те или иные Web-страницы, расположенные на различных серверах.

Web-хостинг — это услуга в Интернете по предоставлению дискового пространства и программного обеспечения для Web-страниц различных пользователей. Если ваша Web-страница написана исключительно на языке HTML, то для ее размещения в Интернете подойдет любой Web-хостинг. Однако для размещения *активных Web-страниц* с интерактивным содержимым различные серверы предлагают разное программное обеспечение.

Так, серверы под управлением операционных систем семейства UNIX/Linux обеспечивают Web-хостинг с использованием таких технологий как PHP, CGI, Perl, Apache, GNU C/C++, базы данных MySQL а также все возможности HTML. При этом пользователи таких Web-страниц на своих локальных (клиентских) компьютерах могут работать под управлением любых операционных систем (UNIX, Windows, Mac OS X). Как видим, в этом перечне отсутствует поддержка технологии Microsoft ASP.NET.

Хостинг на основе ASP.NET существует только на Windows-платформе. То есть этот сервер может работать только под управлением операционной системы Windows. Это сравнительно новая технология от Microsoft, которая называется .NET Framework. Она является расширенным языком ASP. Данная платформа менее популярна, чем технология PHP. Между тем Web-страницы, написанные на языке Visual C# 2010, выполняются по разным оценкам примерно в полтора-два раза быстрее, чем написанные на языке PHP.

Клиент-серверное взаимодействие на основе технологии ASP.NET

Схема работы активной Web-страницы следующая. С помощью URL-адреса Web-браузер *клиентского компьютера* запрашивает необходимую активную Web-страницу. В ответ Web-сервер посылает браузеру ASPX-документ, содержащий HTML-разметку и теги `<asp>`. ASPX-документ интерпретируется браузером клиента и представляется в виде Web-страницы. Пользователь на клиентском компьютере заполняет предложенные в этом документе формы, задает какие-либо команды,

делает запросы и т. д., а затем отсылает (например, нажимая какую-либо командную кнопку) ASPX-документ назад на *сервер* для выполнения. Сервер реагирует на команды пользователя (клиента), выполняя команды с помощью библиотеки DLL, расположенной на сервере, и генерируя новый ASPX-документ, содержащий HTML-разметку вместе с тегами `<asp>`. Теги HTML, содержащие атрибут `runat="server"`, выполняются на сервере. На стороне пользователя Web-браузер только лишь интерпретирует полученный от сервера HTML-поток. DLL (Dynamic-link library) — это динамически подключаемая библиотека, имеющая формат EXE-файла, в ней содержится программный код и ресурсы, используемые активной Web-страницей. С помощью библиотеки DLL реализуется весь сценарий взаимодействия с пользователем. Таким образом, вся "активность" страницы выполняется на стороне сервера, а Web-браузеру пользователя (клиента) отсылается лишь результат выполнения в виде HTML-потока.

Как получить библиотеку DLL? Ничего сложного: управляющая среда Visual Studio 2010 сама генерирует aspx-файл разметки и aspx.cs-файл программной поддержки для Web-страницы, причем aspx.cs-файл является программным кодом Visual C# 2010. А для того чтобы получить dll-библиотеку, следует скомпилировать разрабатываемый проект. При этом получим новую dll-библиотеку в папке bin.

Отладка активного Web-приложения

Тот факт, что Web-страница выполняет команды пользователя на стороне сервера, затрудняет отладку Web-приложения, поскольку требует как минимум двух компьютеров: один будет работать как клиент, а другой — как сервер. На самом деле, как правило, обходятся одним компьютером. Но при этом устанавливают сервер на собственном компьютере для тестирования различных активных страниц, не выходя в Интернет. Среда Visual Studio 2010 имеет возможность использовать собственный локальнй сервер для отладки активных Web-приложений.

Прежние версии Visual Studio, например Visual Studio 2003, требовали обязательной установки локального сервера IIS (Internet Information Services) через **Панель управления | Установка или удаление программ или компонентов Windows | Установка компонентов Windows**. В этом случае система создавала папку C:\Inetpub\wwwroot, поддерживаемую локальным сервером, где должны были располагаться все разрабатываемые Web-приложения с интерактивным содержимым. Обращение к активным Web-страницам, расположенным в папке C:\Inetpub\wwwroot, осуществляется через задание в адресной строке браузера адреса локального сервера, например **http://localhost/primer.aspx**. Но заметьте, что они не будут проявлять себя *как активные* при обращении к ним таким образом: "C:\Inetpub\wwwroot\Имя страницы".

Среда Visual Studio 2010 поддерживает *обе* указанные *возможности* отладки Web-приложения. Менять установку данных возможностей можно через пункты меню **Project | *имя_проекта* Properties**, вкладка **Web**. Причем по умолчанию от-

ладка производится именно с использованием сервера Visual Studio. Это дает возможность отлаживать интерактивные Web-приложения в любой папке винчестера.

Пример 91. Создание простейшей активной Web-страницы на Visual C# 2010

Как уже отмечалось, статичные Web-страницы не способны выполнять даже элементарные арифметические действия. Создадим Web-страницу, демонстрирующую способность как раз складывать числа, введенные пользователем. И одновременно подчеркнем единый поход к разработке Windows- и Web-приложений. В обоих случаях имеем те же инструменты: форму, текстовые поля, метки, командные кнопки и проч. Таким образом, на данной разрабатываемой странице будем иметь два текстовых поля, куда пользователь будет вводить числа, командную кнопку и метку, на которую будет выводиться результат сложения двух чисел.

Для создания Web-приложения откроем Visual Studio 2010, в окне **New Project** на вкладке **Installed Templates** (Установленные шаблоны) **Visual C# Web** выберем шаблон **Empty ASP.NET Web Application**. В поле **Name** укажем имя данного проекта — Summa, при этом снимем флажок **Create directory for Solution**. После щелчка по кнопке **OK** управляющая среда создаст папку Summa, в которой будет находиться проект нового Web-приложения.

Теперь мы можем приступить к проектированию Web-страницы. Поскольку проект пустой (empty), добавим к текущему проекту Web-форму. Для этого в пункте меню **Project** выберем команду **Add New Item**, в появившемся окне укажем шаблон **Web Form** и щелкнем на кнопке **Add**. При этом в окне **Solution Explorer** будет добавлен значок WebForm1.aspx. Теперь мы можем проектировать внешний вид формы на вкладке **WebForm1.aspx**. Внизу вкладки увидим переключатели **Design**, **Source** и **Split**, которые позволяют изменить взгляд на форму, соответственно, в виде визуального проекта (**Design**), в виде HTML- и ASP-разметки (**Source**), а также одновременно обе эти возможности (**Split**).

Принцип перетаскивания из панели элементов **Toolbox** нужных для текущего проекта элементов управления точно такой же, как и при проектировании формы Windows. Поэтому перетащим из панели элементов управления раздела **Standard** необходимые нам два текстовых поля, командную кнопку и метку. Кроме того, в заголовке Web-страницы предусмотрим текст "Введите два числа". Для этого щелкнем правой кнопкой мыши в пределах проектируемой Web-страницы, в контектном меню выберем пункт **Properties**. В появившемся окне свойств **Properties** среди свойств документа (**DOCUMENT**) найдем свойство **Title**, которое содержит текст заголовка Web-страницы. Этому свойству присвоим значение "Введите два числа".

Уже на этом этапе посмотрим на внешний вид проекта, для этого попытаемся выполнить этот проект — нажмем клавишу <F5>. В результате Internet Explorer откроет файл WebForm1.aspx, сгенерированный управляющей средой, и мы получим Web-страницу, изображенную на рис. 13.2.

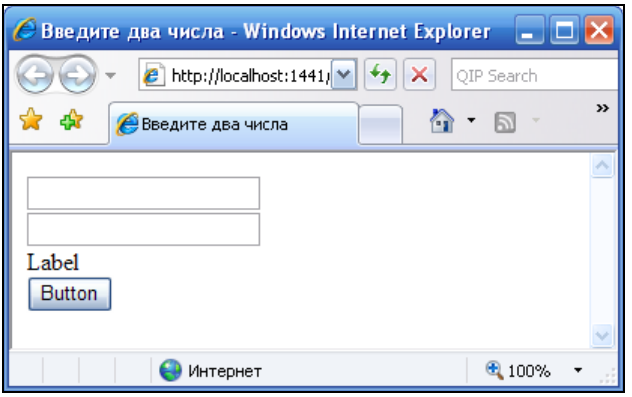


Рис. 13.2. Фрагмент работы активной Web-страницы

Понятно, что пока эта страница является *статической*. Закроем Web-браузер и вернемся к проекту Web-страницы. По умолчанию проектируемая Web-страница *имеет* имя WebForm1.aspx, но при желании в окне **Sulution Explorer** (Обозреватель решений), щелкнув правой кнопкой мыши на значке **WebForm1.aspx**, это имя можно изменить. Вообще говоря, интерактивность, динамичность Web-страницы обеспечивает aspx.cs-файл программной поддержки, содержащий код на Visual C# 2010. Чтобы перейти на вкладку файла программного кода **WebForm1.aspx.cs**, следует, например, воспользоваться контекстным меню проекта Web-формы и выбрать команду **View Code**. Как видите, вкладка программного кода здесь такая же, как и при проектировании формы Windows. На вкладке файла **WebForm1.aspx.cs** напишем следующий программный код (листинг 13.2).

Листинг 13.2. Создание простейшей активной Web-страницы

```
// Создание простейшей активной Web-страницы на Visual C# 2010. Web-страница
// демонстрирует способность складывать числа, введенные пользователем
using System;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace Summa
{
   public partial class WebForm1 : System.Web.UI.Page
   {
      protected void Page_Load(object sender, EventArgs e)
      {  // Обработка события "загрузка страницы"
         Page.Title = "Введите два числа";
         Label1.Text = string.Empty;
         Button1.Text = "Найти сумму двух чисел";
      }
```

```
    protected void Button1_Click(object sender, EventArgs e)
    { // Обработка события "щелчок на кнопке"
       double Z = Convert.ToDouble(TextBox1.Text) +
                  Convert.ToDouble(TextBox2.Text);
       Label1.Text = "Сумма = " + Z;
    }
  }
}
```

Как видно из текста программы, при обработке события загрузки страницы `Page_Load` очищаем (`string.Empty`) текстовую метку и задаем текст на командной кнопке. Обратите внимание, что по умолчанию названия элементов управления (например, `Button1`) система генерирует с прописной буквы. При обработке события "щелчок на кнопке", используя функцию `ToDouble` класса `Convert`, преобразуем символы, введенные пользователем (клиентом) в переменные типа `double`, складываем их и выводим сумму в текстовую метку. Готовая Web-форма будет иметь вид, представленный на рис. 13.3.

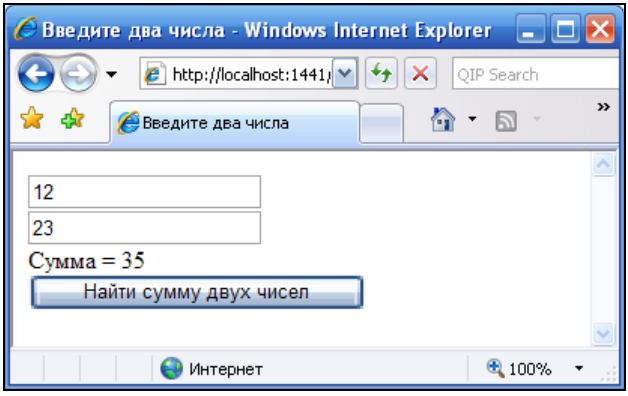


Рис. 13.3. Сумма двух введенных чисел

В результате работы управляющей среды была скомпилирована библиотека Summa.dll в папке bin. Теперь именно Summa.dll будет обеспечивать интерактивность файлу разметки WebForm1.aspx.

Таким образом, созданная Web-страница представляет собой два файла: WebForm1.aspx и WebForm1.aspx.cs. Файл WebForm1.aspx определяет HTML-содержимое сайта, а WebForm1.aspx.cs — программный код, который отделен от HTML-содержимого и предназначен для программного формирования данного содержимого или выполнения каких-либо иных действий на сервере. Эта связь определена в файле WebForm1.aspx в директиве `Page` через свойство `CodeBehind="WebForm1.aspx.cs"`.

Убедиться в работоспособности Web-страницы можно, открыв решение Summa.sln в папке Summa.

Пример 92. Проверка введенных пользователем числовых данных с помощью валидаторов

Очевидно, что созданная в предыдущем разделе Web-страница имеет много недостатков, в частности если пользователь не заполнил текстовые поля, то получим сообщение "Ошибка сервера...". Между тем следовало бы подсказать пользователю, что надо ввести что-либо в указанные поля. Кроме контроля пустых полей необходимо проверить, является ли введенное значение числом или символом (например, буквой). Для этих целей при разработке Windows-приложений мы использовали функцию `TryParse`, однако при разработке Web-приложений имеем среди элементов управления так называемые *валидаторы*, которые можно использовать в документах ASP.NET для осуществления любой проверки введенных данных. Валидаторы (их шесть) можно увидеть в разделе **Validation** в панели элементов **Toolbox**.

В данной задаче мы будем совершенствовать Web-страницу, созданную нами в предыдущем примере. Выполним проверку, ввел ли пользователь хоть что-либо в текстовые поля, а также проверим тип введенных данных и выясним, соответствуют ли они типу данных `Double`.

Для этого запустим Visual Studio 2010, в окне **New Project** выберем среди шаблонов **Visual C# Web** шаблон **Empty ASP.NET Web Application**, в поле **Name** укажем имя нового решения — Valid1 и щелкнем на кнопке **OK**. Теперь добавим к текущему проекту Web-форму. Для этого в пункте меню **Project** выберем команду **Add New Item**, в появившемся окне укажем шаблон **Web Form** и щелкнем на кнопке **Add**. Далее на вкладке конструктора формы перетащим из раздела **Standard** панели элементов **Toolbox** необходимые нам два текстовых поля, командную кнопку и метку.

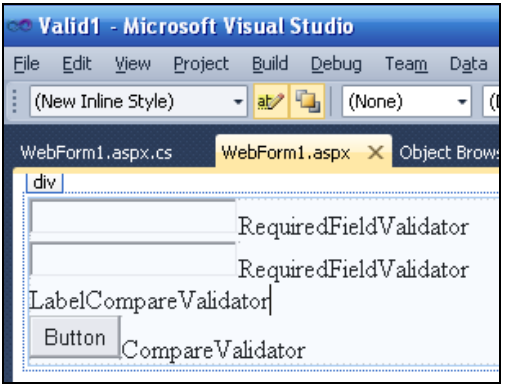


Рис. 13.4. Проектирование Web-формы программы, содержащей валидаторы

В данной задаче мы должны контролировать два текстовых поля, поэтому перенесем мышью из раздела **Validation** панели **Toolbox** соответственно два валидатора **RequiredFieldValidator**, проверяющих факт заполнения текстовых полей, и два валидатора **CompareValidator** для проверки, ввел пользователь числа или другие символы. Расположим их в форме, как показано на рис. 13.4.

Далее на вкладке файла программной поддержки WebForm1.aspx.cs усовершенствуем программный код так, как представлено в листинге 13.3.

Листинг 13.3. Использование валидаторов на Web-странице

```
// Проверка введенных пользователем числовых данных с помощью валидаторов.
// Выполним проверку, ввел ли пользователь хоть что-либо в текстовые поля,
// а также проверим тип введенных данных и выясним, соответствуют ли
// они типу данных Double.
using System;
using System.Web.UI.WebControls;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace Valid1
{
   public partial class WebForm1 : System.Web.UI.Page
   {
      protected void Page_Load(object sender, EventArgs e)
      {  // Обработка события "загрузка страницы"
         Page.Title = "Введите два числа";
         Label1.Text = string.Empty; Label1.Width = 180;
         Button1.Text = "Найти сумму двух чисел";
         Button1.Width = 180;  TextBox1.Focus();
         TextBox1.Width = 180; TextBox2.Width = 180;
         // Установки валидаторов.
         // Контролируем факт заполнения текстовых полей:
         RequiredFieldValidator1.ErrorMessage =
                         "* Следует заполнить это текстовое поле";
         RequiredFieldValidator1.ControlToValidate = "TextBox1";
         RequiredFieldValidator2.ErrorMessage =
                         "* Следует заполнить это текстовое поле";
         RequiredFieldValidator2.ControlToValidate = "TextBox2";
         // Проверяем, соответствуют ли данные, введенные в текстовые поля,
         // типу Double:
         CompareValidator1.ControlToValidate = "TextBox1";
         CompareValidator1.Type = ValidationDataType.Double;
         CompareValidator1.Operator = ValidationCompareOperator.DataTypeCheck;
         CompareValidator1.ErrorMessage = "* Следует вводить числа";
```

```
        CompareValidator2.ControlToValidate = "TextBox2";
        CompareValidator2.Type = ValidationDataType.Double;
        CompareValidator2.Operator = ValidationCompareOperator.DataTypeCheck;
        CompareValidator2.ErrorMessage = "* Следует вводить числа";
    }

    protected void Button1_Click(object sender, EventArgs e)
    {  // Обработка события "щелчок на кнопке"
        double Z = Convert.ToDouble(TextBox1.Text) +
                   Convert.ToDouble(TextBox2.Text);
        Label1.Text = "Сумма = " + Z;
    }
  }
}
```

Как видно, при загрузке страницы в свойствах валидаторов `RequiredField` мы указали имена полей, подлежащих контролю факта их заполнения, а также привели текст сообщения, как реакцию на данную ошибку. В свойствах валидаторов `Compare` также указали имена полей, подлежащих проверке типа данных `DataTypeCheck` и текст сообщения.

Пример работы запрограммированных валидаторов приведен на рис. 13.5.

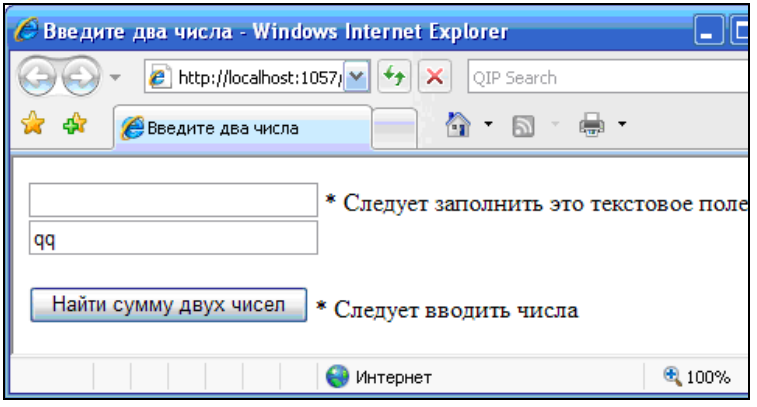


Рис. 13.5. Фрагмент работы Web-страницы с валидаторами

Среди имеющихся на панели **Toolbox** валидаторов уместно отметить **ValidationSummary**. Его можно поместить куда-нибудь на Web-страницу, где будут отображаться все сведения об ошибках от *разных валидаторов*. Причем чтобы сообщения не дублировались, можно указать в свойствах всех валидаторов `Visible = false`, но только свойство валидатора `ValidationSummary.Visible = true`.

Очень ценно то, что валидаторы обеспечивают проверку данных не на сервере, а на машине клиента. Это происходит следующим образом: браузер клиента за-

прашивает на Web-сервере соответствующую страницу; в ответ сервер отсылает на клиентский компьютер HTML-код, который содержит JavaScript с диагностикой, которую мы предусмотрели в cs-файле поддержки. В результате пользователь (клиент) не ощущает задержки во время диагностики данных, поскольку диагностика происходит на клиентском компьютере. Следует отметить то, что если бы мы сами писали валидаторы на языке JavaScript, то потребовалось бы написать гораздо больше строк программного кода.

Убедиться в работоспособности Web-страницы можно, открыв решение Valid1.sln в папке Valid1.

Пример 93. Проверка достоверности ввода имени, адреса e-mail, URL-адреса и пароля с помощью валидаторов

В данном примере рассмотрим типичную задачу проверки ввода пользователем имени, адреса e-mail, URL-адреса и пароля, например при регистрации пользователя. Причем если страница (Web-форма) успешно прошла все этапы проверки, то направляем пользователя на другую, уже разрешенную для этого пользователя, Web-страницу.

Решая поставленную в данном разделе задачу, запустим Visual Studio 2010, закажем новый проект из шаблона **Empty ASP.NET Web Application**, укажем в поле **Name** имя Validations и нажмем кнопку **OK**. К текущему проекту добавим Web-форму. Для этого в меню **Project** выберем команду **Add New Item** и в появившемся окне дважды щелкнем по шаблону **Web Form**.

Теперь в конструкторе формы на вкладке **WebForm1.aspx** из панели **ToolBox** раздела **Standard** перенесем в форму пять меток **Label**, пять текстовых полей **TextBox** и командную кнопку **Button**. Текстовые поля служат для ввода имени пользователя, адреса e-mail, URL-адреса персональной Web-страницы пользователя, пороля и подтверждения пароля.

Для контроля правильности ввода данных перетащим из панели **Toolbox** раздела **Validation** несколько валидаторов. Так, для контроля *обязательности ввода* в первые четыре текстовых поля (для пятого текстового поля — не нужно), перенесем в форму четыре валидатора **RequireFieldValidator**. Для контроля полей **Пароль** и **Подтверждение пароля** перенесем валидатор **CompareValidator**, он будет *сравнивать эти два поля*, поэтому валидатор обязательности ввода для поля **Подтверждение пароля** не нужен. Кроме того, *для контроля формата* ввода e-mail-адреса и URL-адреса Web-страницы на соответствие *заданному шаблону* выберем на панели **Toolbox** элемент управления **RegularExpressionValidator**. Расположим их в форме, как показано на рис. 13.6.

Теперь следует указать соответствующие свойства для каждого из используемых элементов управления. Это можно сделать так же, как и при программирова-

нии Windows-форм визуально, непосредственно в таблице свойств каждого из элементов управления. Свойства элементов управления можно задать также в программном коде, как это сделано в листинге 13.4.

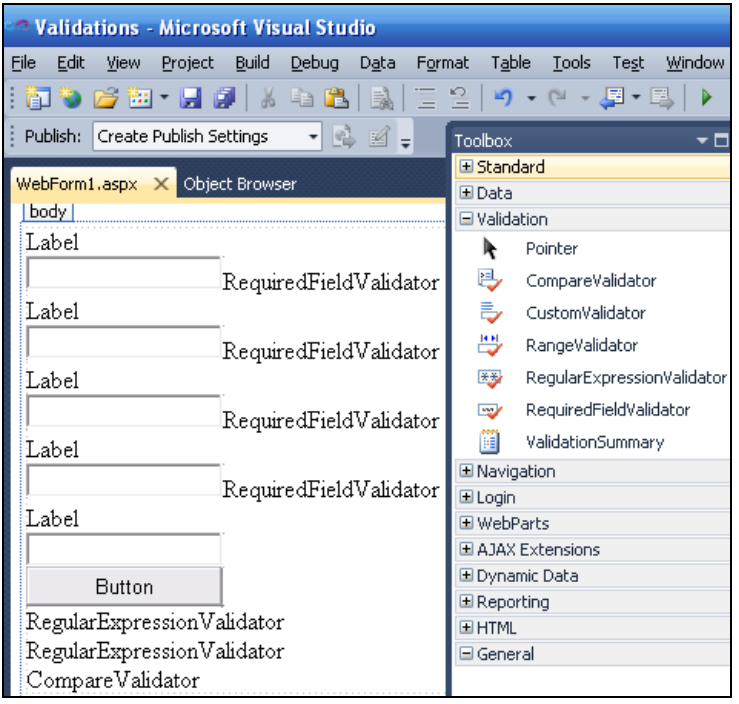


Рис. 13.6. Проектирование Web-формы для проверки личных введенных данных пользователя

Листинг 13.4. Проверка достоверности ввода имени, адреса e-mail, URL-адреса и пароля с помощью валидаторов

```
// Проверка достоверности ввода имени, адреса e-mail, URL-адреса и пароля.
// Web-страница проверяет ввод пользователем всех этих сведений,
// например, при регистрации пользователя. Причем если страница
// (Web-форма) успешно прошла все этапы проверки, то направляем пользователя
// на другую, уже разрешенную для этого пользователя, Web-страницу.
using System;
using System.Web.UI.WebControls;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace Validations
{
   public partial class WebForm1 : System.Web.UI.Page
```

```
{
   protected void Page_Load(object sender, EventArgs e)
   {  // Обработка события "загрузка страницы"
      Page.Title = "Заполните следующие поля:"; TextBox1.Focus();
      Label1.Text = "Имя"; Label2.Text = "E-Mail";
      Label3.Text = "Персональная Web-страница";
      Label4.Text = "Пароль"; Label5.Text = "Подтвердите пароль";
      TextBox4.TextMode = TextBoxMode.Password;
      TextBox5.TextMode = TextBoxMode.Password;
      // Контролируем факт заполнения четырех текстовых полей:
      RequiredFieldValidator1.ControlToValidate = "TextBox1";
      RequiredFieldValidator1.ErrorMessage =
                          "* Следует заполнить это текстовое поле";
      RequiredFieldValidator2.ControlToValidate = "TextBox2";
      RequiredFieldValidator2.ErrorMessage =
                          "* Следует заполнить это текстовое поле";
      RequiredFieldValidator3.ControlToValidate = "TextBox3";
      RequiredFieldValidator3.ErrorMessage =
                          "* Следует заполнить это текстовое поле";
      RequiredFieldValidator4.ControlToValidate = "TextBox4";
      RequiredFieldValidator4.ErrorMessage =
                          "* Следует заполнить это текстовое поле";
      RegularExpressionValidator1.ControlToValidate = "TextBox2";
      // Контроль правильности ведения адресов e-mail и Web-страницы:
      RegularExpressionValidator1.ValidationExpression =
                    @"\w+([-+.]\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*";
      RegularExpressionValidator1.ErrorMessage =
                    "* Следует ввести правильный адрес E-mail";
      RegularExpressionValidator2.ControlToValidate = "TextBox3";
      RegularExpressionValidator2.ValidationExpression =
                    @"http://([\w-]+\.)+[\w-]+(/[\w- ./?%&=]*)?";
      RegularExpressionValidator2.ErrorMessage =
                    "* Следует ввести правильный адрес Web-узла";
      // Контроль правильности введения паспорта путем сравнения
      // содержимого двух полей:
      CompareValidator1.ControlToValidate = "TextBox4";
      CompareValidator1.ControlToCompare = "TextBox5";
      CompareValidator1.ErrorMessage = "* Вы ввели разные паспорта";
      Button1.Text = "Готово";
   }

   protected void Button1_Click(object sender, EventArgs e)
   { // Обработка события "щелчок на кнопке"
```

```
        if (Page.IsPostBack == true)
           if (Page.IsValid == true)
              // Здесь можно записать введенные пользователем сведения
              // в базу данных. Перенаправление на следующую страницу:
              Response.Redirect("Next_Page.htm");
     }
  }
}
```

Как видно, при обработке события загрузки Web-страницы `Page_Load` устанавливаем значения свойств всех элементов управления. Мы могли бы это сделать в конструкторе на вкладке **WebForm1.aspx** в таблицах свойств, но в программном коде это выглядит более выразительно для читателя, здесь трудно пропустить и не учесть какой-либо элемент управления. В свойствах полей `TextBox4` (Пароль) и `TextBox5` (Подтвердите пароль) указанное свойство `TextMode` установлено в режим `Password`, что обеспечивает "защиту от посторонних глаз", т. е. при вводе пароля в данных текстовых полях будем видеть жирные точки вместо вводимых символов.

Далее следуют назначения свойств валидаторам, знакомым нам из предыдущего раздела, обязательности заполнения полей `RequiredField`. Остановимся подробнее на валидаторе `RegularExpression`. Регулярное выражение, используемое для проверки достоверности ввода, выглядит несколько запутанным. Наверное, найдется мало людей, способных в точности переписать регулярное выражение, например для e-mail-адреса:

```
"\w+([-+.]\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*"
```

Этот код можно расшифровать примерно таким образом: `\w+` — обязательный ввод любого количества текстовых символов (буквы и цифры); далее (`[-+.]\w+`) — означает, что допустим ввод точки, а затем опять любое количество букв и цифр, т. е. квадратные скобки означают необязательность, но возможность, а круглые скобки означают просто группировку выражения; затем `*@\w+` — обязательное наличие значка электронной почты `@`, после которого должно следовать любое количество символов и т. д. Вручную такое регулярное выражение вводить не стоит, поскольку это источник дополнительных ошибок. Для задания регулярных выражений лучше на вкладке конструктора формы через щелчок правой кнопкой мыши посмотреть в таблицу свойств (**Properties**) валидатора **RegularExpress**. Здесь, напротив свойства `ValidationExpression`, после щелчка мышью на многоточии получим список типичных регулярных выражений для многих случаев жизни (рис. 13.7).

Выбирать и устанавливать нужное регулярное выражение можно либо в данной таблице, либо копированием через буфер обмена (<Ctrl>+<C>, а затем <Ctrl>+<V>) в программный код.

Контроль заполнения полей **Пароль** и **Подтвердите пароль** осуществлен с помощью валидатора **CompareValidator** путем сравнения этих полей. Фрагмент работы данной Web-формы в момент ее заполнения показан на рис. 13.8.

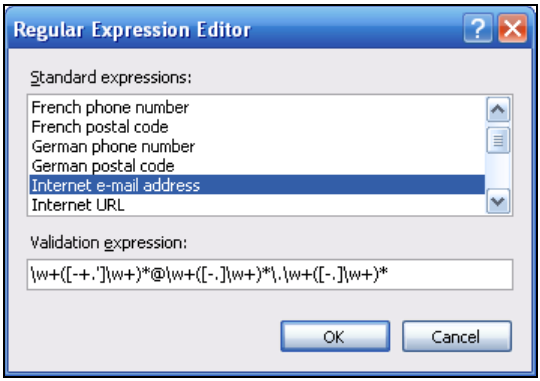


Рис. 13.7. Список типичных регулярных выражений

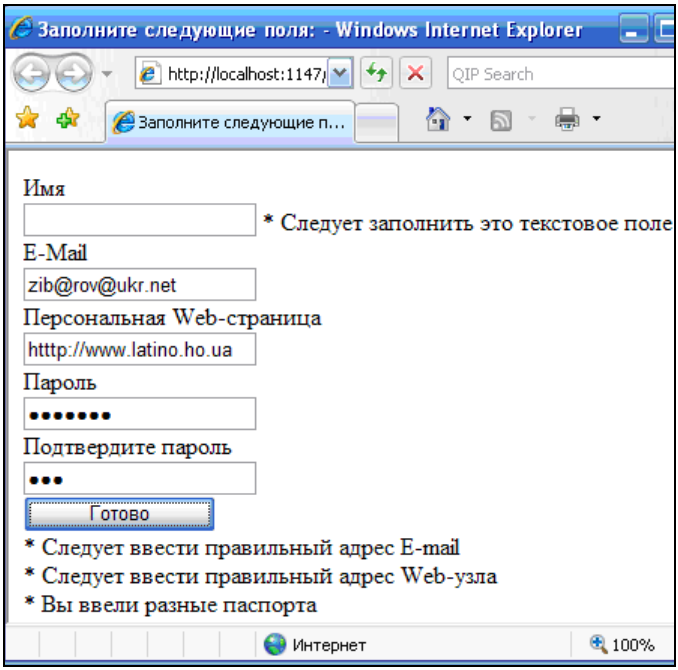


Рис. 13.8. Фрагмент работы Web-страницы при заполнении полей

Вернемся еще раз к программному коду. Здесь при обработке события "щелчок на кнопке" **Готово** происходит проверка, вызвана (загружена) ли данная Web-страница первый раз `IsPostBack = false` или в результате повторной отправки (постбэка) `IsPostBack = true`. Если страница загружена в результате постбэка и проверка правильности страницы с помощью валидаторов также была успешной `IsValid = true`, то можно записать сведения, предоставленные пользователем, в базу данных и перенаправить (`Response.Redirect`) его на следующую страницу Next_Page.htm, предусмотренную сценарием диалога с пользователем.

Создать в текущем проекте новую статическую Web-страницу очень просто, для этого следует в пункте меню **Project** выбрать команду **Add New Item** и дважды щелкнуть на шаблоне **HTML page**. Далее в конструкторе страницы, используя панель элементов **Toolbox**, спроектировать необходимую Web-страницу; заменить имя по умолчанию HTMLPage1.htm на Next_Page.htm удобно в окне **Solution Explorer** (Обозреватель решений).

Убедиться в работоспособности Web-страницы можно, открыв решение Validations.sln в папке Validations.

Пример 94. Регистрация и аутентификация пользователя с помощью базы данных Access

Данный пример включает в себя три Web-формы: Registration.aspx, Login.aspx и Secret.aspx. Первая форма приглашает пользователя ввести регистрационные данные, проверяет правильность ввода имени пользователя и пароля с использованием валидаторов, регистрирует пользователя в базе данных MS Access и перенаправляет пользователя на уже разрешенный после регистрации ресурс Secret.aspx. Страница Secret.aspx допускает пользователя к закрытой информации, если он пришел либо со страницы Registration.aspx, либо со страницы Login.aspx. Форма Login.aspx запрашивает имя пользователя и пароль, проверяет наличие пользователя с таким именем и паролем в базе данных, если такового пользователя не оказалось, то программа отправляет пользователя на регистрацию в Registration.aspx, а если есть, то он получает доступ к ресурсу Secret.aspx.

Для начала создадим базу данных Web.mdb, а в ней таблицу **Аутентифицированные пользователи**. Для этого запустим офисное приложение MS Access, далее в пункте меню **Файл** выполим команду **Создать | Новая база данных** и в нужной папке укажем файл Web.mdb. Далее следует создать таблицу в режиме конструктора в соответствии с очевидным интерфейсом. Имена полей и типы данных приведены на рис. 13.9.

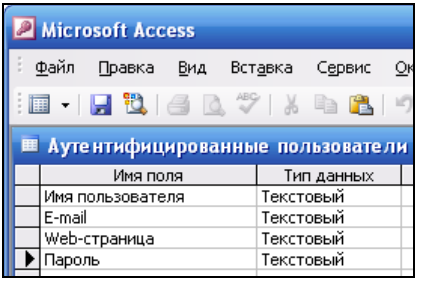


Рис. 13.9. Проектирование таблицы базы данных в режиме конструктора

В данном примере мы не собираемся заполнять поля **E-mail** и **Web-страница** и приводим их здесь, чтобы была преемственность с предыдущим примером. Вообще говоря, следует стараться чувствовать баланс между желанием больше знать о пользователе и *не злоупотреблять терпением пользователя*, который может потерять интерес к вашему сайту и перейти на другой ресурс.

Теперь создадим регистрационную форму в системе Visual studio 2010, для этого после запуска системы закажем новый проект из шаблона **Visual C# Empty ASP.NET Web Application**, в поле **Name** укажем имя Login.sln. К текущему проекту добавим Web-форму. Для этого в пункте меню **Project** выберем команду **Add New Item** и в появившемся окне дважды щелкнем на шаблоне **Web Form**.

Далее, попав в конструктор Web-формы, в окне **Solution Explorer** (включите кнопку **Show All Files**), щелкнув правой кнопкой мыши на изображении файла WebForm1.aspx, переименуем его в Registration.aspx. Вид регистрационной формы показан на рис. 13.10.

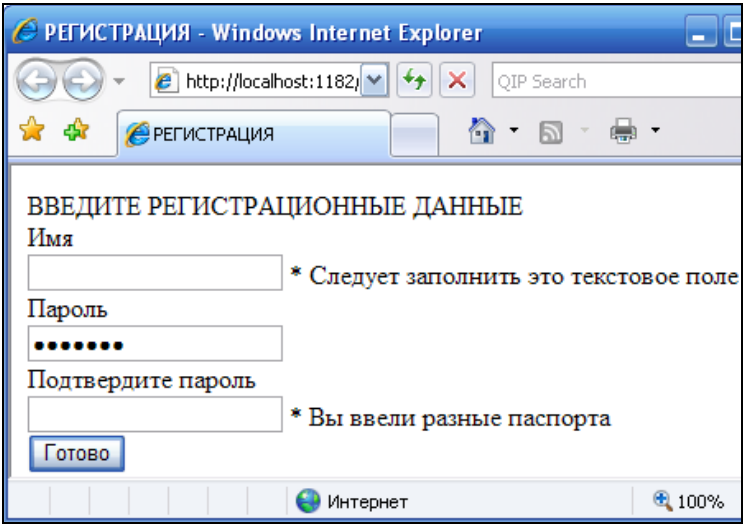


Рис. 13.10. Фрагмент работы регистрационной формы

Как видно, для создания такой формы потребовалось четыре метки **Label**, три текстовых поля **TextBox** и командная кнопка **Button**. Кроме того, для контроля правильности ввода данных перетащим из панели **Toolbox** несколько валидаторов. Так, для контроля обязательности ввода в первые два текстовых поля перенесем в форму два валидатора **RequireFieldValidotor**. Для контроля полей **Пароль** и **Подтверждение пароля** перенесем валидатор **CompareValidotor**, он будет сравнивать эти два поля, поэтому валидатор обязательности ввода для поля **Подтверждение пароля** не нужен. В листинге 13.5 приведено содержимое файла программной поддержки Registration.aspx.cs.

Листинг 13.5. Регистрация пользователя

```
// Регистрация и аутентификация пользователя с помощью базы данных MS Access.
// Данный пример включает в себя три Web-формы: Registration.aspx,
// Login.aspx и Secret.aspx. Первая форма Registration.aspx приглашает
// пользователя ввести регистрационные данные, проверяет правильность ввода
// имени пользователя и пароля с использованием валидаторов, регистрирует
// пользователя в базе данных MS Access и перенаправляет пользователя
// на уже разрешенный после регистрации ресурс Secret.aspx
using System;
using System.Web.UI.WebControls;
// Импортируем пространство имен для манипуляций с БД:
using System.Data.OleDb;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace Login
{
   public partial class WebForm1 : System.Web.UI.Page
   {
      protected void Page_Load(object sender, EventArgs e)
      {  // Обработка события "загрузка страницы"
         Page.Title = "РЕГИСТРАЦИЯ";
         Label1.Text = "ВВЕДИТЕ РЕГИСТРАЦИОННЫЕ ДАННЫЕ";
         Label2.Text = "Имя"; Label3.Text = "Пароль";
         Label4.Text = "Подтвердите пароль";
         Button1.Text = "Готово"; TextBox1.Focus();
         TextBox2.TextMode = TextBoxMode.Password;
         TextBox3.TextMode = TextBoxMode.Password;
         // Контролируем факт заполнения двух текстовых полей:
         RequiredFieldValidator1.ControlToValidate = "TextBox1";
         RequiredFieldValidator1.ErrorMessage =
                           "* Следует заполнить это текстовое поле";
         RequiredFieldValidator2.ControlToValidate = "TextBox2";
         RequiredFieldValidator2.ErrorMessage =
                           "* Следует заполнить это текстовое поле";
         // Контроль правильности введения паспорта путем сравнения
         // содержимого двух полей:
         CompareValidator1.ControlToValidate = "TextBox2";
         CompareValidator1.ControlToCompare = "TextBox3";
         CompareValidator1.ErrorMessage = "* Вы ввели разные паспорта";
      }
```

```
protected void Button1_Click(object sender, EventArgs e)
{  // Обработка события "щелчок на кнопке".
   // Запись в базу данных только при повторной отправке
   // и при достоверных данных
   if (IsPostBack == false || IsValid == false) return;
   // Здесь можно записать введенные пользователем сведения в БД.
   // Строка подключения:
   string СтрокаПодкл =
          "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" +
          Server.MapPath("Web.mdb");
   // MapPath - возвращает физический путь.
   // Создание экземпляра объекта Connection:
   var Подключение = new OleDbConnection(СтрокаПодкл);
   try
   {  // Открытие подключения
      Подключение.Open();
   }
   catch (Exception ex2)
   {
      Response.Write("<br><br>" + ex2.Message);
      return;
   }
   var Команда = new OleDbCommand();
   // ДОБАВЛЕНИЕ ЗАПИСИ О ПОЛЬЗОВАТЕЛЕ В БД.
   // Строка SQL-запроса
   string SQL_запрос = "INSERT INTO [Аутентифицированные пользовате" +
                  "ли] ([Имя пользователя], [Пароль]) VALUES ('" +
                  TextBox1.Text + "', '" + TextBox2.Text + "')";
   // Создание объекта Command с заданием SQL-запроса
   Команда.CommandText = SQL_запрос;
   // Для добавления записи в БД эта команда обязательна
   Команда.Connection = Подключение;
   try
   {  // Выполнение команды SQL, т. е. ЗАПИСЬ В БД
      Команда.ExecuteNonQuery();
   }
   catch (Exception ex3)
   {
      Response.Write("<br><br>" + ex3.Message);
      return;
   }
```

```
            Подключение.Close();
            // Перенаправление на уже разрешенную страницу
            Response.Redirect("Secret.aspx");
        }
    }
}
```

Как видно, в данном программном коде при обработке события загрузки формы происходит проверка достоверности введенных пользователем данных с помощью валидаторов так же, как и в предыдущем примере. При нажатии пользователем кнопки **Готово** запись в базу данных происходит только при повторной отправке данных `IsPostBack = true`, а также при достоверности введенных данных `IsValid = true`. Для записи в базу данных мы не стали переносить в форму соответствующие элементы управления из панели **Toolbox**, а очень кратко и выразительно использовали их непосредственно в программном коде.

Теперь создадим Web-страницу для проверки имени пользователя и пароля с использованием таблицы **Аутентифицированные пользователи** в базе данных. Для этого добавим к текущему проекту новую Web-форму, выбрав в пункте меню **Project** команду **Add New Item | Web Form**. Назовем эту новую форму Login.aspx. Внешний вид формы, запрашивающей у пользователя имя и пароль с их проверкой в базе данных, показан на рис. 13.11.

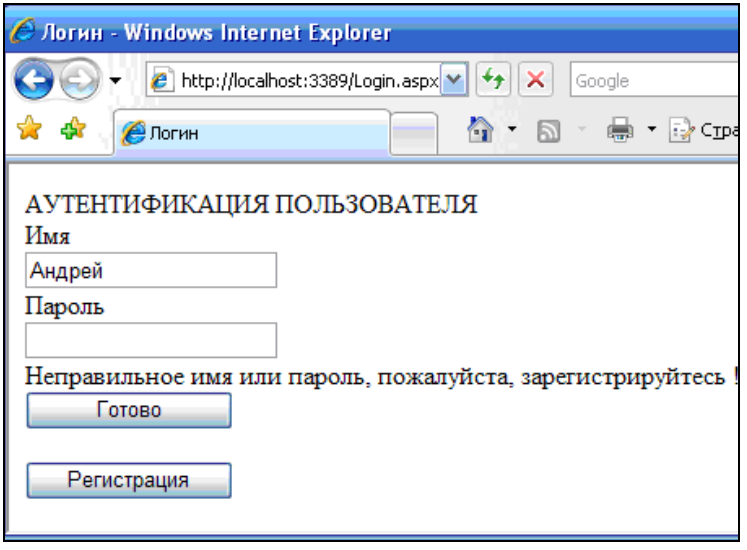


Рис. 13.11. Форма аутентификации пользователя

Здесь потребовалось пять меток, два текстовых поля и две командных кнопки. Пятую метку на рис. 13.11 не видно, мы расположили ее между кнопками. В листинге 13.6 приведено содержимое файла программной поддержки Login.aspx.cs.

Листинг 13.6. Аутентификация пользователя

```
// Вторая форма Login.aspx запрашивает имя пользователя и пароль, проверяет
// наличие пользователя с таким именем и паролем в базе данных. Если такого
// пользователя не оказалось, то форма отправляет пользователя на регистрацию
// Registration.aspx, а если есть, то он получает доступ к ресурсу Secret.aspx
using System;
using System.Web.UI.WebControls;
// Импортируем пространство имен для манипуляций БД:
using System.Data.OleDb;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace Login
{
   public partial class Login : System.Web.UI.Page
   {
      protected void Page_Load(object sender, EventArgs e)
      {
         Page.Title = "Логин"; Page.Form.Method = "post";
         Label1.Text = "АУТЕНТИФИКАЦИЯ ПОЛЬЗОВАТЕЛЯ";
         Label2.Text = "Имя";         Label3.Text = "Пароль";
         Label4.Text = string.Empty; Label5.Text = string.Empty;
         TextBox1.Focus(); TextBox2.TextMode = TextBoxMode.Password;
         Button1.Text = "Готово"; Button2.Text = "Регистрация";
         Button1.Width = 125;     Button2.Width = 125;
         TextBox1.Width = 140;    TextBox1.Width = 140;
      }

      protected void Button1_Click(object sender, EventArgs e)
      {  // Щелчок на кнопке "Готово"
         bool ПОЛЬЗОВАТЕЛЬ_АУТЕНТИФИЦИРОВАН = false;
         // Строка подключения
         string СтрокаПодкл =
              "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" +
              Server.MapPath("Web.mdb");
         // Создание экземпляра объекта класса Connection
         var Подключение = new OleDbConnection(СтрокаПодкл);
         try
         {  // Открытие подключения
            Подключение.Open();
         }
         catch (Exception ex1)
```

```
        {
           Label5.Text = ex1.Message;
        }
        // Строка SQL-запроса для проверки имени и пароля
        string SQL_запрос =
              "SELECT Пароль FROM [Аутентифицированные пользо" +
              "ватели] WHERE ([Имя пользователя] = '" +
              TextBox1.Text + "') AND (Пароль = '" + TextBox2.Text + "')";
        // Создание объекта Command с заданием SQL-запроса
        var Команда = new OleDbCommand();
        Команда.CommandText = SQL_запрос;
        Команда.Connection = Подключение;
        try
        {  // Выполнение команды SQL
           OleDbDataReader Читатель = Команда.ExecuteReader();
           if (Читатель.Read() == true)
           {
              ПОЛЬЗОВАТЕЛЬ_АУТЕНТИФИЦИРОВАН = true;
              Label4.Text = "Пользователь аутентифицирован";
           }
           else
           {
              ПОЛЬЗОВАТЕЛЬ_АУТЕНТИФИЦИРОВАН = false;
              Label4.Text =
              "Неправильное имя или пароль, пожалуйста, зарегистрируйтесь!";
           }
        }
        catch (Exception ex2)
        {
           Label5.Text = Label5.Text + "<br />" + ex2.Message;
        }
        Подключение.Close();
        if (ПОЛЬЗОВАТЕЛЬ_АУТЕНТИФИЦИРОВАН == true)
           // Направляем пользователя на уже разрешенную страницу
           Response.Redirect("Secret.aspx");
     }

     protected void Button2_Click(object sender, EventArgs e)
     {  // Щелчок на кнопке "Регистрация"
        Response.Redirect("Registration.aspx");
     }
  }
}
```

Здесь при нажатии пользователем кнопки **Готово** осуществляется SQL-запрос к базе данных на наличие записи с полями "имя пользователя" и "пароль", полученными от `TextBox1` и `TextBox2`. Если такая запись найдена, то делаем вывод, что *пользователь аутентифицирован*, и направляем его на уже разрешенную для него Web-страницу Secret.aspx. Если же запись в базе данных с такими полями (именем и паролем) *не найдена*, то сообщаем об этом пользователю на метку `Label4`. Пользователь при этом может либо уточнить имя и пароль, либо перейти к процедуре регистрации, нажав кнопку **Регистрация**.

А что же Web-форма Secret.aspx? Если в ней просто привести закрытую информацию, то недобросовестный пользователь, выяснив адрес этой страницы, очень легко наберет его в адресной строке браузера и получит без нашей санкции доступ к искомой странице, а все наши "умные" регистрации с использованием базы данных окажутся бесполезными. Исходя из этих соображений, добавляем в текущий проект (**Project | Add New Item | Web Form**) новую Web-форму, называем ее Secret.aspx. В эту форму из раздела **Standard** панели элементов перетаскиваем метку **Label** и командную кнопку **Button**. В листинге 13.7 приведен код программной поддержки для файла-разметки Secret.aspx.

Листинг 13.7. Web-форма Secret.aspx

```
// К этой странице имеют доступ только зарегистрированные пользователи.
// Вначале пытаемся (Try) определить, с какой Web-страницы пришел
// пользователь на данный ресурс. Если не удается (ветка catch,
// т. е. пользователь пытается открыть данный закрытый ресурс, набрав его
// URL-адрес в адресной строке браузера), то сообщаем пользователю, что он
// не зарегистрирован, и делаем видимой кнопку "Регистрация". Если удается
// определить URL-адрес страницы, с которой пользователь пришел на данную
// страницу, то разрешаем доступ к закрытой информации только для адресов
// /Login.aspx и /Registration.aspx.
using System;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace Login
{
   public partial class Secret : System.Web.UI.Page
   {
      protected void Page_Load(object sender, EventArgs e)
      {  // Обработка события "загрузка страницы"
         Button1.Visible = false; Button1.Text = "Регистрация";
         string URL_адрес;
         // Определение, с какой Web-страницы вы пришли на данную
         // страницу, т. е. определение локального адреса (пути)
         try
```

```
        {
            URL_адрес = Request.UrlReferrer.LocalPath;
            // Более эффективно определять абсолюный виртуальный адрес:
            // URL_адрес = Request.UrlReferrer.AbsoluteUri
            if (URL_адрес == @"/Login.aspx" ||
                            URL_адрес == @"/Registration.aspx")
            {
                Label1.Text = "Поскольку Вы являетесь " +
                    "зарегистрированным пользователем, то Вы имеете " +
                    "доступ к закрытой информации. Вы пришли на эту" +
                    " страницу со страницы " + URL_адрес;
                return;
            }
        }
        catch (Exception ex)
        {
            Label1.Text = "Вы не являетесь зарегистрированным " +
                "пользователем, поскольку пришли на эту страницу, " +
                "набрав URL-адрес в адресной строке браузера.<br />" +
                ex.Message;
            Button1.Visible = true;
            return;
        }
        Label1.Text = "Вы не являетесь зарегистрированным " +
            "пользователем, поскольку пришли со страницы " + URL_адрес;
        Button1.Visible = true;
    }

    protected void Button1_Click(object sender, EventArgs e)
    {  // Щелчок на кнопке "Регистрация"
        Response.Redirect("Registration.aspx");
    }
  }
}
```

Как видно из программного кода, используя свойство `LocalPath`, определяем, с какой Web-страницы пришел пользователь на данную страницу. Причем мы здесь определяем локальный виртуальный адрес, поскольку нам таким образом удобно производить отладку программы, но вообще говоря необходимо определять абсолютный вируальный адрес, как показано в комментарии. Далее разрешаем доступ к закрытой информации только пользователям, пришедшим на данную страницу с Web-страниц Registration.aspx или Login.aspx.

Убедиться в работоспособности данных Web-страниц (в рассмотренном примере их три) можно, открыв решение Login.sln в папке Login.

Пример 95. Таблица с переменным числом ячеек, управляемая двумя раскрывающимися списками

При Web-строительстве очень часто приходится работать с таблицами для различных целей. В данном примере продемонстрируем, как с помощью двух раскрывающихся списков **DropDownList** можно заказывать необходимое число рядов и столбцов в таблице.

Программируя поставленную задачу, запустим систему Visaul Studio 2010 и создадим новый проект шаблона **Empty ASP.NET Web Application**, назовем его tab.sln. К текущему проекту добавим Web-форму. Для этого в пункте меню **Project** выберем команду **Add New Item** и в появившемся окне дважды щелкнем по шаблону **Web Form**.

Далее в конструкторе **WebForm1.aspx** из раздела **Standard** панели элементов **Toolbox** в проектируемую форму перенесем элемент управления **Table**, два раскрывающихся списка **DropDownList**, две метки **Label** и командную кнопку **Button**. Разместим их на форме, как показано на рис. 13.12. Теперь, используя, например, контекстное меню проекта Web-формы, выберем команду **View Code** и попадем на вкладку файла программной поддержки **WebForm1.aspx.cs**. В листинге 13.8 приведен соответствующий программный код.

Листинг 13.8. Таблица с переменным числом ячеек

```
// Таблица с переменным числом ячеек, управляемая двумя раскрывающимися
// списками. Web-страница позволяет с помощью двух раскрывающихся списков
// DropDownList заказать необходимое число рядов и столбцов в таблице,
// а затем строить заказанную таблицу
using System;
using System.Web.UI.WebControls;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace tab
{
   public partial class WebForm1 : System.Web.UI.Page
   {
      protected void Page_Load(object sender, EventArgs e)
      {
         Page.Title = "Укажите размерность таблицы";
         if (Page.IsPostBack == true) return;
```

```
        // Заполнять выпадающий список необходимо при первой загрузке
        // страницы IsPostBack = false, иначе будут добавляться новые
        // пункты в выпадающей список при каждой перегрузке страницы
        DropDownList1.Items.Add("1"); DropDownList1.Items.Add("2");
        DropDownList1.Items.Add("3");
        DropDownList2.Items.Add("1"); DropDownList2.Items.Add("2");
        DropDownList2.Items.Add("3");
        Table1.Caption = "Название таблицы";
        Table1.CaptionAlign = TableCaptionAlign.Right;
        Table1.ToolTip = "Укажи количество рядов и столбцов и нажми кнопку";
        Table1.BorderStyle = BorderStyle.Solid;
        Table1.GridLines = GridLines.Both;
        Label1.Text = "Кол-во строк"; Label2.Text = "Кол-во столбцов";
        Button1.Text = "Обновить таблицу";
    }

    protected void Button1_Click(object sender, EventArgs e)
    {
        int i, j;
        for (i = 1; i <= int.Parse(DropDownList1.SelectedItem.Value); i++)
        {
            var РЯД = new TableRow();
            for (j = 1; j <= int.Parse(DropDownList2.SelectedItem.Value); j++)
            {
                var ЯЧЕЙКА = new TableCell();
                ЯЧЕЙКА.Text = "Ряд " + i + ", Кол " + j;
                ЯЧЕЙКА.HorizontalAlign = HorizontalAlign.Center;
                РЯД.Cells.Add(ЯЧЕЙКА);
            }
            Table1.Rows.Add(РЯД);
        }
    }
  }
}
```

Как видно из программного кода, при загрузке страницы `Page_Load` происходит заполнение значениями раскрывающихся списков **DropDownList**, причем заполнение осуществляется только при первоначальной загрузке страницы, когда `IsPostBack = false`. Если бы мы не предусмотрели обход присвоения начальных значений при `IsPostBack = true`, то значения в списке добавлялись бы каждый раз при очередной перезагрузке страницы. Заметим, что мы могли бы и не контролировать `IsPostBack`, если бы присвоение начальных значений выполнили в процедуре обработки события инициализации страницы `Page_PreInit` (в Visual Studio 2003 это событие называлось `Page_Init`).

Свойства других элементов управления также инициализированы при обработке события загрузки Web-страницы `Page_Load`. Мы могли бы это сделать в конструкторе, но для удобства читателя приводим их в программном коде. Назначения этих свойств очевидны и не требуют дополнительных комментариев.

При обработке события "щелчок на кнопке" **Обновить таблицу** имеем два вложенных цикла. Параметры обоих циклов `i` и `j` изменяются от 1 до значения, выбранного пользователем в соответствующем раскрывающемся списке. Метод `int.Parse` пространства имен `System` конвертирует строку из соответствующего свойства объекта `DropDownList` в переменную типа `int`. Внешний цикл перебирает ряды таблицы, а внутренний — ячейки таблицы. В теле внешнего цикла очередная итерация создает новый объект `РЯД` класса `TableRow`, аналогично в теле внутреннего цикла каждый раз создается новый объект `ЯЧЕЙКА` класса `TableCell`.

Для старта созданного проекта нажмем клавишу <F5>. В результате получим примерно такое отображение таблицы, как показано на рис. 13.12.

Убедиться в работоспособности Web-страницы можно, открыв решение tab.sln в папке tab.

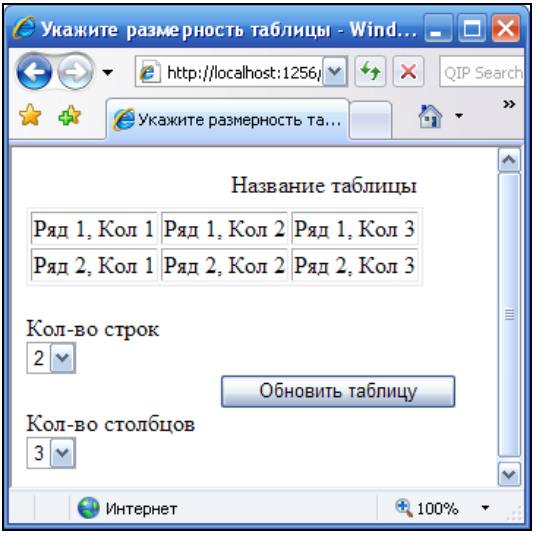


Рис. 13.12. Фрагмент работы Web-страницы с таблицей

Пример 96. Организация раскрывающегося меню гиперссылок с помощью *DropDownList*

Переходы на другие Web-страницы, так называемую *навигацию*, в статических HTML-страницах обычно организуют с помощью ряда гиперссылок. Например, сайт состоит из десяти Web-страниц. На каждой из этих страниц слева имеем ко-

лонку из десяти строчек текстовых гиперссылок для перехода на эти страницы. В данном примере продемонстрируем, как можно организовать переход на разные страницы сайта с помощью раскрывающегося списка **DropDownList**.

Для реализации этой задачи запустим систему Visual Studio 2010, далее — новый проект из шаблона **Empty ASP.NET Web Application**, имя — Menu. К текущему проекту добавим Web-форму. Для этого в пункте меню **Project** выберем команду **Add New Item** и в появившемся окне дважды щелкнем на шаблоне **Web Form**.

Затем на проектируемую Web-форму перетащим из раздела **Standard** панели **Toolbox** элемент управления **DropDownList** и метку **Label**. Теперь перейдем к вкладке файла программной поддержки **WebForm1.aspx.cs** и здесь организуем гиперссылки от каждого пункта меню на некоторые Web-страницы, созданные в данной главе. Текст программы приведен в листинге 13.9.

Листинг 13.9. Организация навигации по Web-страницам

```
// Web-страница демонстрирует, как можно организовать переход на разные
// страницы сайта (гиперссылки) с помощью раскрывающегося списка DropDownList
using System;
using System.Web.UI.WebControls;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace Menu
{
   public partial class WebForm1 : System.Web.UI.Page
   {
      protected void Page_Load(object sender, EventArgs e)
      {
         Page.Title = "Переход на другую страницу:";
         if (Page.IsPostBack == true) return;
         // Можно писать так:
         //  var it1 = new System.Web.UI.WebControls.ListItem();
         //  it1.Text = "Сложить два числа";
         //  it1.Value = "Summa.aspx";
         //  DropDownList1.Items.Add(it1);
         // А можно короче:
         // DropDownList1.Items.Add(new ListItem("Имя", "значение"))
         DropDownList1.Items.Add(new ListItem(
                     "Остаться на этой странице", "WebForm1.aspx"));
         DropDownList1.Items.Add(new ListItem(
               "Проверка достоверности введенных данных",
               "Validations.aspx"));
         DropDownList1.Items.Add(new ListItem(
               "Управляемая таблица", "tab.aspx"));
```

```
        // Выполнять ли повторную отправку (постбэк), когда
        // пользователь сделает выбор в раскрывающемся списке?
        DropDownList1.AutoPostBack = true;
        // При AutoPostBack = True будет работать
        // событие DropDownList1_SelectedIndexChanged.
        Label1.Text = "Перейти на другую Web-страницу:";
    }

    protected void DropDownList1_SelectedIndexChanged(object sender,
                                                      EventArgs e)
    {  // Выполнить команду перехода на другую страницу:
       Response.Redirect(DropDownList1.SelectedValue);
    }
    //protected void Page_PreInit(object sender, EventArgs e)
    //{
    //   Мы могли бы заполнять пункты раскрывающегося списка DropDownList
    //   не при загрузке формы, а при ее инициализации (событие
    //   Page_PreInit), тогда не надо думать про постбэк.
    //   Чтобы получить обработчик этого события, надо просто его
    //   написать: protected void Page_PreInit...
    //}
  }
}
```

Здесь, так же как и в предыдущем примере, мы заполняли пункты раскрывающегося списка при первой загрузке страницы `IsPostBack = false`. Это можно было бы делать также при инициализации Web-страницы (событие `Page_PreInit`). Причем заполняли список по формату (`"Имя"`, `"Значение"`). Здесь `"Имя"` будет видно в пунктах меню, а `"Значение"` можно использовать функционально.

Кроме того, для раскрывающегося списка мы указали свойство `AutoPostBack = true`, что приведет к повторной отправке, когда пользователь сделает свой выбор в раскрывающемся списке, причем в этом случае будет доступно событие `SelectedIndexChanged`. Именно этим событием мы воспользовались для перенаправления (`Redirect`) браузера на выбранную пользователем страницу.

Как видно, все выбираемые пользователем Web-страницы являются активными ASPX-страницами. Чтобы добавить их в текущий проект следует выбрать в пункте меню **Project** команду **Add New Item** и дважды щелкнуть на шаблоне **Web Form**. В окне **Solution Explorer** щелкнем правой кнопкой мыши на названии файла и переименуем новую Web-форму из WebForm2.aspx в Validations.aspx. Аналогично следует добавить вторую вызываемую Web-форму и назвать ее tab.aspx. Перед стартом программы (<F5>) в окне **Solution Explorer** правой кнопкой мыши щелкнем по изображению файла WebForm1.aspx и в контекстном меню выберем команду **Set As Start Page**, чтобы программа начала работу с этой страницы.

Фрагмент работы данной Web-страницы приведен на рис. 13.13.

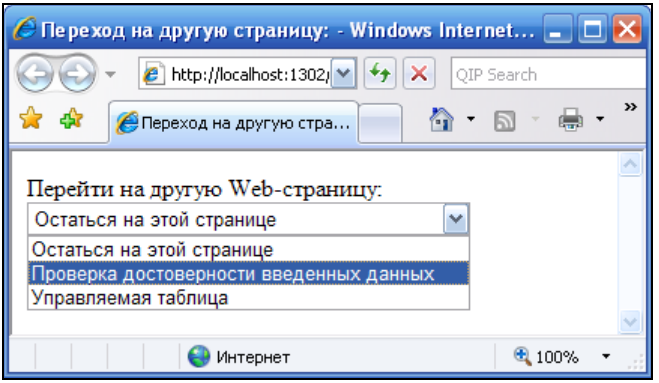


Рис. 13.13. Фрагмент работы Web-страницы, использующей раскрывающийся список

Упомянутая функция `Redirect` является методом объекта `Response`. Очень часто пользуются методом `Write` объекта `Response` для вывода в Web-форму каких-либо данных, в том числе для тестирования, отладки программы. Вообще говоря, в технологии ASP.NET имеются два очень важных объекта: `Response` и `Request`. Объект `Response` содержит данные для передачи от сервера клиенту, а объект `Request` — информацию, полученную от пользователя. В дальнейшем мы будем обращаться к этим двум объектам.

Убедиться в работоспособности Web-страницы можно, открыв решение Menu.sln в папке Menu.

Пример 97. Передача данных между Web-страницами через параметры гиперссылки

Передача данных между Web-страницами одного сайта может происходить через файлы, через объект `Session`, а может быть реализована через параметры, указанные в *гиперссылке*. Эти параметры также называют *строкой запроса*. Например, гиперссылка

```
<a href="Target.aspx?n1=77&n2='Зиборов'">
```

после вопросительного знака содержит два параметра — `n1` и `n2`. Значения этих параметров будут переданы в Web-документ Target.aspx. Прочитать эти параметры в Target.aspx можно с помощью объекта `Request`.

В данном примере создадим две Web-страницы: Source.aspx и Target.aspx. На первой странице Source.aspx с помощью генератора случайных чисел `Random` выбираем одну из пар "имя—фамилия", затем кодируем их, чтобы они не были видны в адресной строке. Щелчок пользователя по гиперссылке вызывает переход на стра-

ницу Target.aspx, причем в гиперссылке указаны оба закодированных параметра. На этой новой странице с помощью объекта `Request` получаем оба переданных параметра и выводим их в Web-документе.

Для реализации данной задачи запустим Visual Studio 2010, далее в окне **New Project** выберем шаблон **Empty ASP.NET Web Application**, в поле **Name** укажем имя Ssylka. К текущему проекту добавим Web-форму. Для этого в пункте меню **Project** выберем команду **Add New Item** и в появившемся окне дважды щелкнем на шаблоне **Web Form**.

В этом проекте будем программировать две Web-формы: одну назовем Source.aspx, а вторую — Target.aspx. Сначала спроектируем первую форму, для этого в окне **Solution Explorer** переименуем файл WebForm1.aspx в Source.aspx. Теперь на вкладку **Source.aspx** из раздела **Standard** панели **Toolbox** перенесем элемент управления **HyperLink** (Гиперссылка). Содержимое файла программной поддержки Source.aspx.cs приведено в листинге 13.10.

Листинг 13.10. Передача данных между Web-страницами через параметры гиперссылки

```
// Передача данных между Web-страницами через параметры гиперссылки. В данном
// примере имеем две Web-страницы: Source.aspx и Target.aspx. На первой
// странице Source.aspx с помощью генератора случайных чисел Random выбираем
// одну из пар "имя-фамилия", затем кодируем их, чтобы они не были видны
// в адресной строке. Щелчок пользователя по гиперссылке вызывает переход
// на страницу Target.aspx, причем в гиперссылке указаны оба закодированных
// параметра
using System;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace Ssylka
{
   public partial class Source : System.Web.UI.Page
   {
      protected void Page_Load(object sender, EventArgs e)
      {
         // var Генератор = new Random(DateTime.Now.Millisecond);
         // или просто
         var Генератор = new Random();
         int ЧИСЛО = Генератор.Next(0, 3);
         string t1 = null, t2 = null;
         switch (ЧИСЛО)
         {  // Случайное попадание на одну из меток case:
            case 0:
               t1 = "Витя"; t2 = "Зиборов"; break;
```

```
            case 1:
               t1 = "Света"; t2 = "Ломачинская"; break;
            case 2:
               t1 = "Андрей"; t2 = "Зиборов-младший"; break;
         }
         // Закодируем t1 и t2 для того, чтобы они не были видны
         // в адресной строке:
         t1 = System.Web.HttpUtility.UrlEncode(t1);
         t2 = System.Web.HttpUtility.UrlEncode(t2);
         HyperLink1.Text = "Щелкните эту ссылку для " +
                           "передачи данных на страницу Target.aspx";
         HyperLink1.NavigateUrl = "Target.aspx?n1=" + t1 + "&n2=" + t2;
      }
   }
}
```

Как видно из текста программы, при обработке события загрузки страницы `Page_Load` создаем объект `Генератор` класса `Random`, в скобках задаем начало псевдослучайной последовательности, равное, например, как приведено в комментарии, текущей миллисекунде текущей даты. Используя объект `Генератор`, получаем случайное число в интервале от 0 до 2. Это число, попадая в оператор множественного выбора `switch`, вызывает присваивание пары "имя—фамилия". Далее функция `UrlEncode` кодирует `t1` и `t2` для того, чтобы они не были видны в адресной строке браузера. В конце назначены свойства гиперссылки с указанием параметров `n1` и `n2` для передачи их в вызываемую Web-страницу Target.aspx. В результате работы Web-формы Source.aspx в браузере мы получим следующее (рис. 13.14).

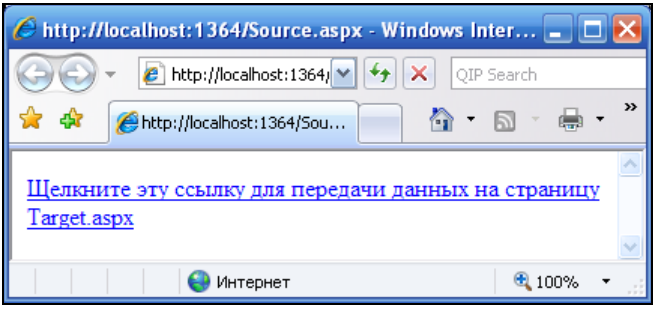


Рис. 13.14. Фрагмент работы Web-страницы с гиперссылкой

Далее, щелкнув на этой ссылке, мы попадаем на вызываемую Web-страницу Target.aspx. Однако в нашем проекте еще нет Web-страницы Target.aspx. Для добавления новой Web-формы в пункте меню **Project** выберем команду **Add New Item** и появившемся окне дважды шелкнем на значке шаблона **Web Form**. В окне **Solution Explorer** переименуем (щелкнув правой кнопкой мыши) новую Web-

форму из WebForm2.aspx в Target.aspx. Добавлять в эту форму из панели **Toolbox** ничего из элементов управления не станем, поскольку выводить в форму некоторый текст будем, используя метод `Write` объекта `Response`. В листинге 13.11 приведено содержимое файла программной поддержки Target.aspx.cs.

Листинг 13.11. Получение переданных параметров

```
// Передача данных между Web-страницами через параметры гиперссылки. На этой
// новой странице Target.aspx.cs с помощью объекта Request получаем оба
// переданных параметра и выводим их в Web-документе
using System;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace Ssylka
{
   public partial class Target : System.Web.UI.Page
   {
      protected void Page_Load(object sender, EventArgs e)
      {  // Получаем параметры, переданные с Web-страницы, с которой была
         // вызвана данная страница
         string ИМЯ1, ИМЯ2; // Request - это запрос
         ИМЯ1 = Request.QueryString.Get("N1");
         ИМЯ2 = Request.QueryString.Get("N2");
         string URL = Request.UrlReferrer.AbsoluteUri;
         Response.Write(@"<br />Вы попали на данную Web-" +
                        "страницу со страницы: " + URL + "<br />");
         Response.Write(@"<br />Страница " + URL + " передала на " +
                  "данную страницу имя: " + ИМЯ1 + " и фамилию " + ИМЯ2);
      }
   }
}
```

В этом программном коде извлекаем переданные параметры `n1` и `n2` (в данном случае прописные и строчные буквы равнозначны) методом `QueryString.Get` объекта `Request`. С помощью объекта `Request` получаем также абсолютный URL-адрес вызывающей страницы. Для вывода какого-либо текста на Web-страницу мы могли бы воспользоваться элементом управления **Label**, однако мы вывели текст "Вы попали..." методом `Write` объекта `Response`, чтобы продемонстрировать удобство метода `Write`, например, для отладки программного кода. Тег `<br />` использован нами для того, чтобы начать предложение с новой строки. Перед стартом программы в окне **Solution Explorer** правой кнопкой мыши щелкнем по изображению файла Source.aspx и в контекстном меню выберем команду **Set As Start Page**, чтобы

программа начала работу с этой страницы. Результат работы Web-страницы приведен на рис. 13.15.

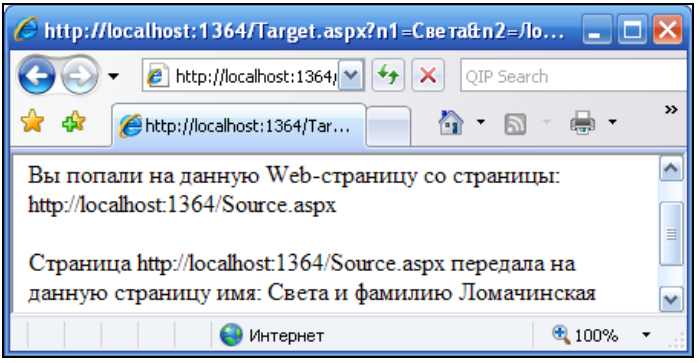


Рис. 13.15. Целевая Web-страница

При использовании строки запроса, т. е. параметров гиперссылки, следует помнить, что некоторые браузеры ограничивают длину URL. Поэтому необходимо следить, чтобы текст в адресной строке не превышал 255 символов.

Убедиться в работоспособности Web-страницы можно, открыв решение Ssylka.sln в папке Ssylka.

Пример 98. Передача данных *HTML*-формы на ASPX-страницу методами класса *Request*

Имеется возможность предложить пользователю заполнить в форме текстовые поля, установить в какое-либо положение переключатели, выбрать какой-либо элемент из списка и т. д., т. е. выполнить какие-либо манипуляции с элементами управления, а затем перейти на другую Web-страницу и отобразить на этой новой странице измененные свойства элементов управления предыдущей страницы. Замечу, что это — типичная ситуация.

Поставим следующую конкретную задачу. Имеем Web-форму, написанную на "чистом" HTML, т. е. в данной разметке нет тегов ASP, JavaScript-кода или PHP-скрипта, есть только теги HTML. Web-форма содержит два текстовых поля и кнопку для отправки данных. Одно поле приглашает пользователя ввести свое имя, а второе — адрес электронной почты. Щелчок на кнопке отправляет данные, заполненные пользователем в форме, на другую Web-страницу для обработки и отображения принятых данных на этой новой странице.

Для решения этой задачи запустим Visual Studio 2010, закажем новый проект из шаблона **Empty ASP.NET Web Application**. Добавим Web-форму, содержащую

только теги HTML, для этого в пункте меню **Project** выберем команду **Add New Item** и в появившемся окне укажем шаблон **HTML page**. В окне **Solution Explorer** переименуем новый появившийся файл в ishod.htm, на вкладке **View Code** этого файла введем HTML-разметку, представленную на рис. 13.16.

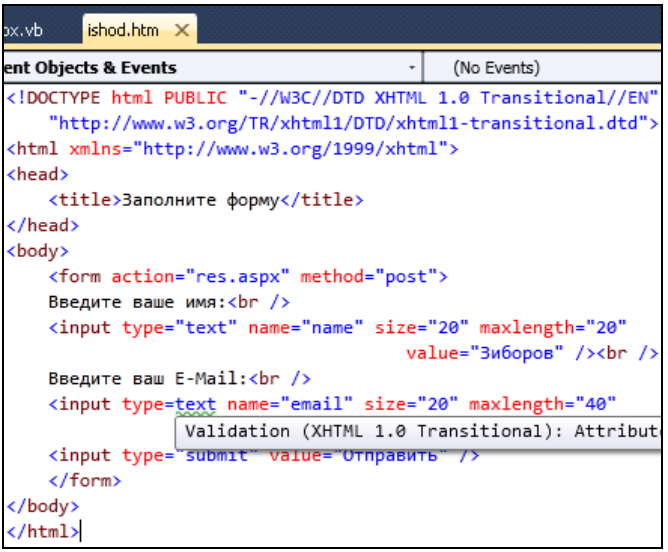


Рис. 13.16. HTML-разметка Web-страницы, содержащей форму

Обратите внимание на тег `<form>`: здесь атрибут `action` (действие) формы ссылается на страницу, которая будет обрабатывать данные, принятые с формы. Атрибут формы `method` определяет способ передачи данных. Существуют два метода: `get` и `post`. Метод `get` передает все данные формы в конце URL через параметры гиперссылки. Из-за различных ограничений, связанных со спецификой языков и длиной данных, этот метод применяется редко. Метод `post` передает все данные формы в теле запроса. Этот метод используется чаще, чем `get`.

Элемент `<input>` является базовым для всех элементов формы. Он используется для внедрения в форму кнопок, графических изображений, флажков, переключателей, паролей и текстовых полей. Атрибут `type` определяет тип элемента, для текстовых полей `type="text"`, для кнопок `type="submit"`, для флажков `type="checkbox"` и т. д. Атрибутом `name` мы будем пользоваться для идентификации элементов управления при обработке данных формы. Атрибутом `value` мы воспользовались, чтобы в первом поле по умолчанию каждый раз при запуске формы оказывалась фамилия автора.

Заметьте, что Visual Studio 2010 содержит удобный HTML-редактор, который зеленым подчеркиванием указывает на синтаксические ошибки, допущенные при вводе HTML-разметки. Для примера на рис. 13.16 значение атрибута `type` не заключено в двойные кавычки, редактор HTML-разметки подчеркнул этот фрагмент зеленой волнистой линией, и при "зависании" указателя мыши над ним мы получи-

ли сообщение об этой ошибке: "Validation (XHTML 1.0 Transitional): Attribute values must be enclosed in quotation marks".

Нажмем клавишу <F5> для запуска данного проекта, в результате получим форму, подлежащую заполнению (рис. 13.17).

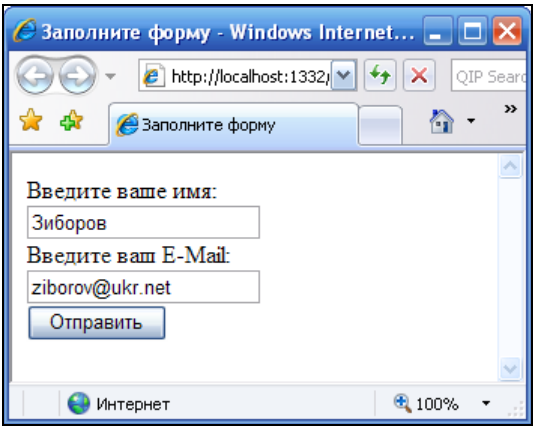


Рис. 13.17. Форма для заполнения и дальнейшей обработки

Теперь создадим Web-страницу, которая будет обрабатывать данные, принятые с формы. Для этого в пункте меню **Project** выберем команду **Add New Item**, в новом появившемся окне укажем шаблон **Web Form** и добавим его к нашему проекту. В окне **Solution Explorer** переименуем новый файл в res.aspx. На этой новой странице текущего проекта мы не будем использовать никакие элементы управления. Отображать принятые новой страницей данные будем с помощью объекта `Response`. На вкладке файла программной поддержки **res.aspx.cs** введем программный код, приведенный в листинге 13.12.

Листинг 13.12. Формирование Web-страницы, получающей данные

```
// Данная Web-страница получает имя пользователя и адрес его электронной
// почты от Web-формы ishod.htm и отображает эти данные на странице
using System;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace Передача
{
   public partial class res : System.Web.UI.Page
   {
      protected void Page_Load(object sender, EventArgs e)
      {
         // Page.Title = "Прием данных от страницы ishod.htm";
```

```
            string Метод = Request.HttpMethod;
            string Имя = Request.Form.Get("name");
            string Почта = Request.Form.Get("email");
            Response.Write("Передача данных произвдена методом " +
                           Метод + @"<br />");
            Response.Write("Получено имя: " + Имя + @"<br />");
            Response.Write("Получен адрес электронной почты: " + Почта);
        }
    }
}
```

Как видно из программного кода, при обработке события загрузки страницы чтение переданных данных организуем с помощью методов класса `Request`. Фрагмент работы Web-страницы представлен на рис. 13.18.

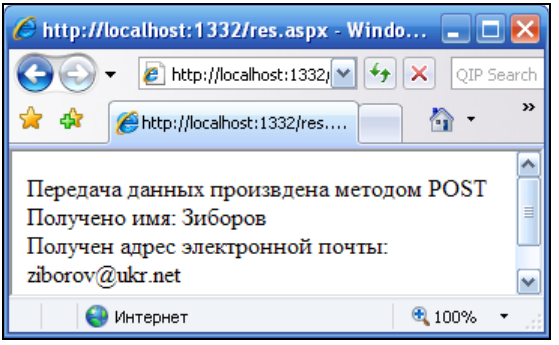


Рис. 13.18. Отображение данных формы на новой странице

Убедиться в работоспособности данного проекта можно, открыв решение Передача.sln в папке Передача.

Пример 99. Передача значений элементов управления на другую Web-страницу с помощью объекта *PreviousPage*

Покажем, как можно решить задачу из предыдущего примера методами объекта `PreviousPage`. Итак, на начальной Web-странице имеем командную кнопку **ПЕРЕХОД** и текстовое поле, которое заполняет пользователь. После щелчка на кнопке происходит переход на другую Web-страницу. На новой странице отображается содержимое текстового поля и надпись на кнопке из предыдущей страницы.

Для решения этой задачи запустим Visual Studio 2010, закажем новый проект из шаблона **Empty ASP.NET Web Application**. Далее добавим в текущий проект на-

чальную Web-форму, для этого в пункте меню **Project** выполним команду **Add New Item** и в появившемся окне выберем шаблон **Web Form**. В окне **Solution Explorer** переименуем (команда **Rename** в контекстном меню) новую появившуюся форму из WebForm1.aspx в ishod.aspx. На вкладке конструктора формы из раздела **Standard** панели элементов **Toolbox** перенесем командную кнопку **Button** и текстовое поле **TextBox**. На вкладке программного кода (клавиша <F7>) напишем текст, приведенный в листинге 13.13.

Листинг 13.13. Заполнение данных на первой Web-странице

```
// На начальной Web-странице имеем командную кнопку "ПЕРЕХОД" и текстовое поле,
// которое заполняет пользователь. После щелчка на кнопке происходит переход
// на другую Web-страницу. На новой странице отображается содержимое
// текстового поля и надпись на кнопке из предыдущей страницы
using System;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace Передача2
{
   public partial class ishod : System.Web.UI.Page
   {
      protected void Page_Load(object sender, EventArgs e)
      {
         Button1.Text = "ПЕРЕХОД"; Button1.PostBackUrl = "res.aspx";
      }
      protected void Button1_Click(object sender, EventArgs e)
      {
         // Response.Redirect("res.aspx");
      }
   }
}
```

Как видно из программного кода, при обработке события загрузки формы присваиваем надписи на кнопке строку "ПЕРЕХОД", а свойству кнопки `PostBackUrl` назначаем URL-адрес, на который должно перейти управление при щелчке на командной кнопке. Заметим, что, используя свойство `PostBackUrl`, мы можем не обрабатывать событие "щелчок на кнопке", поскольку переход при щелке на кнопке будет осуществляться на страницу, указанную в этом свойстве. Однако в комментарии указано как альтернатива использование метода `Redirect` объекта `Response`.

Замечу, что Web-формы res.aspx, на которую имеем ссылку в программном коде, еще нет в нашем проекте. Добавим эту форму так же, как добавили исходную форму, т. е. используя в пункте меню **Project** команду **Add New Item**, выберем шаблон новой формы **Web Form**. Никаких элементов управления на эту форму не

добавляем. В окне **Solution Explorer** переименуем новую появившуюся форму в res.aspx. На вкладке программого кода **res.aspx.cs** напишем текст, приведенный в листинге 13.14.

Листинг 13.14. Получение данных на второй Web-странице

```
// Получаем данные со страницы ishod.aspx
using System;
using System.Web.UI.WebControls;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace Передача2
{
   public partial class res : System.Web.UI.Page
   {
      protected void Page_Load(object sender, EventArgs e)
      {
         try
         {  // Создаем объекты, как на форме-источнике:
            TextBox ТекстовоеПоле = (TextBox)Page.
                                    PreviousPage.FindControl("TextBox1");
            Button Кнопка = (Button)Page.PreviousPage.FindControl("Button1");
            // Значения элементов управления могут быть также получены
            // с помощью метода Form.Get объекта Request:
            // string S1 = Request.Form.Get("TextBox1");
            // string S2 = Request.Form.Get("Button1");
            Response.Write(@"На начальной странице: <br />");
            Response.Write(" - содержимое текстового поля: " +
                           ТекстовоеПоле.Text + @"<br />");
            Response.Write(" - надпись на кнопке: " + Кнопка.Text);
         }
         catch (Exception ex)
         {
            Response.Write("Начальная Web-страница должна " +
                        @"содержать<br />текстовое поле и командную " +
                        @"кнопку.<br />" + ex.Message);
         }
      }
   }
}
```

Как видно из данного программного кода, новая, результирующая форма не содержит ни одного элемента управления. Вывод текста в форму осуществляем с по-

мощью метода `Write` объекта `Response`. Чтобы вывести в новую форму значения элементов управления первоначальной формы, воспользуемся методом `FindControl` объекта `PreviousPage`. На вход этого метода подадим идентификатор (ID) соответствующего элемента управления, указанного в HTML-разметке первоначальной формы ishod.aspx. Функция `FindControl` возвращает объект класса `Web.UI.Control`, его удобно конвертировать в объект класса `TextBox` или `Button` с помощью неявного преобразования. В комментарии показано, как можно получить значения элементов управления первоначальной формы также с помощью метода `Form.Get` объекта `Request`.

Перед тем как запустить созданный проект на выполнение, в окне **Solution Explorer** назначим в качестве *стартовой страницы* проекта файл ishod.aspx: в контекстном меню для этого файла выберем команду **Set As Start Page**. На рис. 13.19 показана исходная форма ishod.aspx с текстовым полем, подлежащим заполнению, и командной кнопкой.

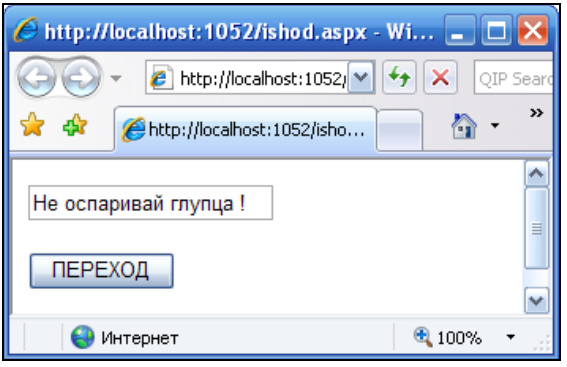


Рис. 13.19. Первоначальная форма

На рис. 13.20 продемонстрирована результирующая форма, где отображены значения элементов управления первоначальной формы.

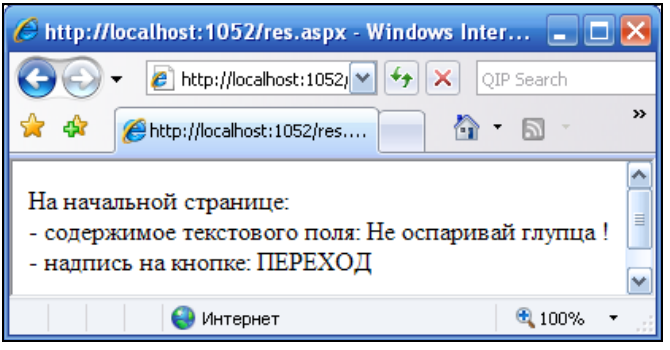


Рис. 13.20. Результирующая форма

Убедиться в работоспособности Web-страницы можно, открыв решение Передача2.sln в папке Передача2.

Пример 100. Отображение табличных данных в Web-форме с помощью элемента управления *GridView*

В данном примере покажем, как можно вывести какие-либо данные, например два строковых массива в виде таблицы в Web-форму с помощью элемента управления **GridView** (просмотр или обзор сетки данных в таблице) и объекта `DataTable`. Мы решали подобную задачу для Windows-приложения. Между тем для Web-приложения это также типичная задача, но она имеет некоторые особенности, в частности и там, и здесь в выводе таблицы принимают участие похожие элементы управления (**DataGridView** и **GridView**), но свойства и методы этих объектов в разных приложениях немного отличаются. В данной задаче таблица телефонов представлена в виде двух строковых массивов. Требуется минимальными усилиями вывести в Web-форму эти массивы в виде таблицы.

Для решения данной задачи запустим Visual Studio 2010, закажем новый проект из шаблона **Empty ASP.NET Web Application**, в поле **Name** введем имя TabGrdWeb. К текущему проекту добавим Web-форму. Для этого в пункте меню **Project** выберем команду **Add New Item** и в появившемся окне дважды щелкнем на шаблоне **Web Form**.

Далее перетащим в форму из панели **Toolbox** (из подраздела **Data**) элемент управления **GridView**. Содержимое файла программной поддержки WebForm1.aspx.cs приведено в листинге 13.15.

Листинг 13.15. Вывод табличных данных в Web-форму с помощью элемента управления *GridView*

```
// Вывод табличных данных в Web-форму с помощью элемента управления GridView.
// В данной Web-странице организован вывод двух строковых массивов в таблицу
// Web-формы с помощью элемента управления GridView и объекта класса DataTable
using System;
using System.Web.UI.WebControls;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace TabGrdWeb
{
   public partial class WebForm1 : System.Web.UI.Page
   {
      protected void Page_Load(object sender, EventArgs e)
```

```
    {
        Page.Title = "Вывод таблицы в Web-форму";
        string[] Imena = { "Андрей - раб", "ЖЭК",
                           "Мама - дом", "Карапузова Таня" };
        string[] Tel = { "274-88-17", "22-345-72",
                         "570-38-76", "201-72-23-прямой моб" };
        var Таблица = new System.Data.DataTable();
        // Заполнение "шапки" таблицы
        Таблица.Columns.Add("ИМЕНА");
        Таблица.Columns.Add("НОМЕРА ТЕЛЕФОНОВ");
        // Заполнение клеток (ячеек) таблицы
        for (int i = 0; i <= 3; i++)
           Таблица.Rows.Add(new string[] { Imena[i], Tel[i] });
        // Немного другое свойство, чем в WindowsApplication
        GridView1.Caption = "Таблица телефонов";
        GridView1.BorderWidth = Unit.Pixel(2);
        GridView1.BorderColor = System.Drawing.Color.Gray;
        GridView1.DataSource = Таблица;
        // Этого нет в WindowsApplication
        GridView1.DataBind();
    }
  }
}
```

Содержание программного кода очевидно. Вначале инициализируются два строковых массива: массив имен и массив телефонов, далее с помощью этих массивов заполняются ячейки объекта класса `DataTable`. Затем этот объект назначается нами источником данных `DataSource` для сетки данных `GridView1`. Фрагмент работы данной Web-страницы показан на рис. 13.21.

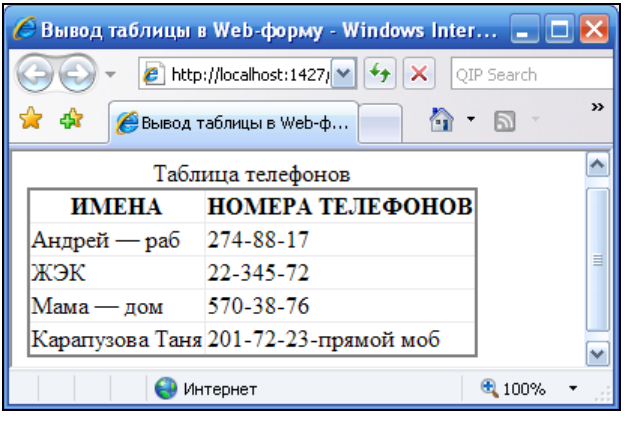


Рис. 13.21. Фрагмент работы Web-страницы с табличными данными

Замечу, что данную таблицу можно было бы вывести с помощью элемента управления **DataList**, однако в этом случае пришлось бы задавать шаблоны Template для каждой колонки в ASPX-файле, что на субъективный взгляд автора не технологично.

Убедиться в работоспособности Web-страницы можно, открыв решение TabGrdWeb.sln в папке TabGrdWeb.

Пример 101. Отображение в Web-форме хэш-таблицы

Хэш-таблица — это таблица из двух столбцов, один из них содержит ключи, а второй — значения. То есть каждая строка в этой таблице образует пару "ключ—значение". Имея ключ в хэш-таблице, можно быстро найти значение. Хэш-таблицу можно назвать таблицей соответствий. Простейшим примером хэш-таблицы является таблица телефонов, которая участвовала в предыдущем разделе, однако там мы программировали ее просто как два массива. Если эти два массива поместить в хэш-таблицу, то ключом в данном случае было бы `ФИО`, а значением — `номер телефона`. При этом программирование поиска значения по ключу оказалось бы тривиальной задачей, операция добавления и удаления пары также упрощается, поскольку хэш-таблица — это объект, который содержит соответствующие эти и другие методы. В реальной жизни много разнообразных примеров представления данных в виде хэш-таблицы. Например, таблица, где расширения файлов (txt, jpg, mdb, xls) являются ключами, а соответствующими значениями — программы, которые открывают файлы с такими расширениями (Notepad.exe, Pbrush.exe, MSAccess.exe, Excel.exe). Следующим примером является англо-русский и другие словари. База данных доменных имен, которая доменному имени сопоставляет IP-адрес. По принципу хэш-таблицы организованы объекты `ViewState` и `Session` технологии ASP.NET.

Поставим следующую задачу, сопоставим в хэш-таблице государства в качестве ключей, а их столицы — в качестве значений. Далее, используя элемент управления **GridView**, выведем эту хэш-таблицу на Web-страницу. Для этого запустим Visual Studio 2010, закажем новый проект шаблона **Empty ASP.NET Web Application**, в поле **Name** укажем имя Hash_Grid. К текущему проекту добавим Web-форму. Для этого в пункте меню **Project** выберем команду **Add New Item** и в появившемся окне дважды щелкнем на шаблоне **Web Form**.

Далее перетащим в форму из панели **Toolbox** (из подраздела **Data**) элемент управления **GridView**. Содержимое файла программной поддержки WebForm1.aspx.cs приведено в листинге 13.16.

Листинг 13.16. Хэш-таблица в Web-форме

```
// Вывод в Web-форму хэш-таблицы, которая позволяет поставить в соответствие
// государства их столицам. То есть в качестве ключей имеем государства, а их
```

```
// столицы - в качестве значений. Далее, используя элемент управления
// GridView, программа выводит эту хэш-таблицу на Web-страницу
using System;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace Hash_Grid
{
   public partial class WebForm1 : System.Web.UI.Page
   {
      protected void Page_Load(object sender, EventArgs e)
      {
         Page.Title = "Пример хэш-таблицы";
         var Хэш = new System.Collections.Hashtable();
         // Заполнение хэш-таблицы.
         // Можно добавлять записи "ключ-занчение" так:
         Хэш["Украина"] = "Киев";
         // А можно добавлять так:
         Хэш.Add("Россия", "Москва");
         // Здесь государство - это ключ, а столица - это значение
         Хэш.Add("Белоруссия", "Минск");
         // Создаем обычную таблицу (не хэш):
         var Таблица = new System.Data.DataTable();
         // Заполнение "шапки" таблицы вывода
         Таблица.Columns.Add("ГОСУДАРСТВА");
         Таблица.Columns.Add("СТОЛИЦЫ");
         // В цикле заполняем обычную таблицу парами из хэш-таблицы по рядам:
         foreach (System.Collections.DictionaryEntry ОднаПара in Хэш)
            // Здесь структура DictionaryEntry
            // определяет пару "ключ - значение"
            Таблица.Rows.Add(new object[] { ОднаПара.Key, ОднаПара.Value });
         // Немного другое свойство, чем в WindowsApplication
         GridView1.Caption = "Таблица государств"; // Заголовок таблицы
         GridView1.BorderWidth = System.Web.UI.WebControls.Unit.Pixel(2);
         GridView1.BorderColor = System.Drawing.Color.Gray;
         // Источник данных для GridView
         GridView1.DataSource = Таблица;
         // Этого нет в WindowsApplication
         GridView1.DataBind();
      }
   }
}
```

Здесь при обработке события загрузки Web-страницы создается объект класса `Hashtable`. Хэш-таблица заполняется тремя парами "код—значение", причем, как показано в программном коде, допустимы обе формы записи: через присваивание и посредством метода `Add`. Далее создается вспомогательный объект класса `DataTable`, который следует заполнить данными из хэш-таблицы. Хэш-таблица имеет структуру типа `DictionaryEntry`, которая позволяет перемещаться по рядам в цикле и таким образом получить все пары из хэш-таблицы. В этом цикле происходит заполнение объекта класса `DataTable`. Далее, также как и в предыдущем примере, для `GridView1` указываем в качестве источника данных заполненный объект `DataTable`. Пример работы данной Web-страницы показан на рис. 13.22.

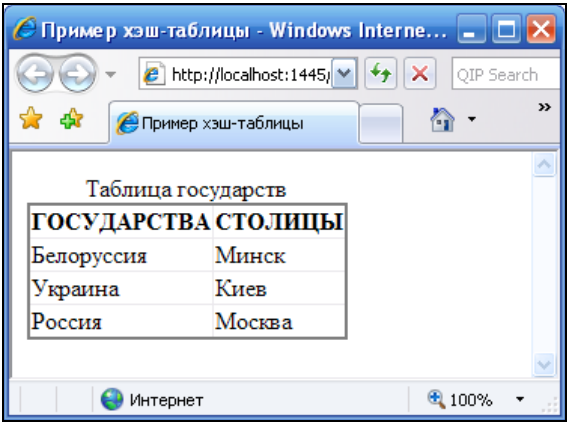


Рис. 13.22. Фрагмент работы Web-страницы с хэш-таблицей

В заключение отметим, что хэш-таблицу называют *ассоциативным массивом*, но в этом "массиве" роль индекса играет ключ. Для реализации хэш-таблицы можно было бы использовать обычный одномерный массив, в котором элементы с четным индексом являются ключами, а с нечетным — значениями. Однако для реализации трех основных операций с хэш-таблицей: добавления новой пары, операции поиска и операции удаления пары по ключу потребовалось бы отлаживать довольно-таки много строчек программного кода.

Убедиться в работоспособности Web-страницы можно, открыв решение Hash_Grid.sln в папке Hash_Grid.

Глава 14

Типичные Web-ориентированные приложения ASP.NET на Visual C# 2010

Пример 102. Чтение/запись текстового файла Web-приложением

Активная Web-страница может сохранять, а также читать данные на диске в каких-либо файлах, в том числе текстовых. Принцип чтения/записи на диск текстовых файлов такой же как и в Windows-приложениях, однако есть некоторые особенности. Поставим задачу написать Web-приложение, читающее текстовый файл и выводящее его в текстовое поле, пользователь имеет возможность редактировать текст и сохранять его в том же файле.

Для решения поставленной задачи запустим Visual Studio 2010, закажем новый проект из шаблона **Empty ASP.NET Web Application**, имя Web-приложения — RW_txt. К текущему проекту добавим Web-форму. Для этого в пункте меню **Project** выберем команду **Add New Item** и в появившемся окне дважды щелкнем на шаблоне **Web Form**.

В конструкторе формы из раздела **Standard** панели **Toolbox** перетащим мышью текстовое поле **TextBox** и две командных кнопки **Button**. Расположим их на форме так, как показано на рис. 14.1. Содержимое файла программной поддержки приведено в листинге 14.1.

Листинг 14.1. Чтение/запись текстового файла

```
// Чтение/запись текстового файла Web-приложением. Web-приложение читает
// текстовый файл в текстовое поле, а пользователь имеет возможность
// редактировать текст и сохранять его в том же файле
using System;
using System.Web.UI.WebControls;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
```

```
namespace RW_txt
{
   public partial class WebForm1 : System.Web.UI.Page
   {
      string filename; // - имя файла используется в обеих процедурах
      // Чтобы русские буквы читались корректно, объявляем объект КОДИРОВКА
      System.Text.Encoding КОДИРОВКА = System.Text.Encoding.GetEncoding(1251);
      protected void Page_Load(object sender, EventArgs e)
      {
         Page.Title = "Здесь кодировка Win1251";
         Button1.Text = "Читать"; Button2.Text = "Сохранить";
         Button1.Focus();
         // Разрешаем многострочие
         TextBox1.TextMode = TextBoxMode.MultiLine;
         filename = Request.PhysicalApplicationPath + "txt.txt";
      }

      protected void Button1_Click(object sender, EventArgs e)
      {  // Чтение файла:
         try
         {  // Создаем экземпляр StreamReader для чтения из файла
            var ЧИТАТЕЛЬ = new System.IO.StreamReader(filename, КОДИРОВКА);
            TextBox1.Text = ЧИТАТЕЛЬ.ReadToEnd();
            ЧИТАТЕЛЬ.Close();
         }
         catch (System.IO.FileNotFoundException Ситуация)
         {
            Response.Write("Нет такого файла <br />" + Ситуация.Message);
         }
         catch (Exception Ситуация) // Отчет о других ошибках
         {
            Response.Write("Ошибка чтения файла  <br />" +
                           Ситуация.Message + " <br />");
         }
      }

      protected void Button2_Click(object sender, EventArgs e)
      {  // Сохранение файла:
         try
         {  // Создание экземпляра StreamWriter для записи в файл
            var ПИСАТЕЛЬ = new System.IO.StreamWriter(filename,
                                          false, КОДИРОВКА);
```

```
            ПИСАТЕЛЬ.Write(TextBox1.Text);
            ПИСАТЕЛЬ.Close();
        }
        catch (Exception Ситуация)
        {  // Отчет обо всех возможных ошибках
            Response.Write("Ошибка записи файла <br />" +
                           Ситуация.Message + " <br />");
        }
    }
  }
}
```

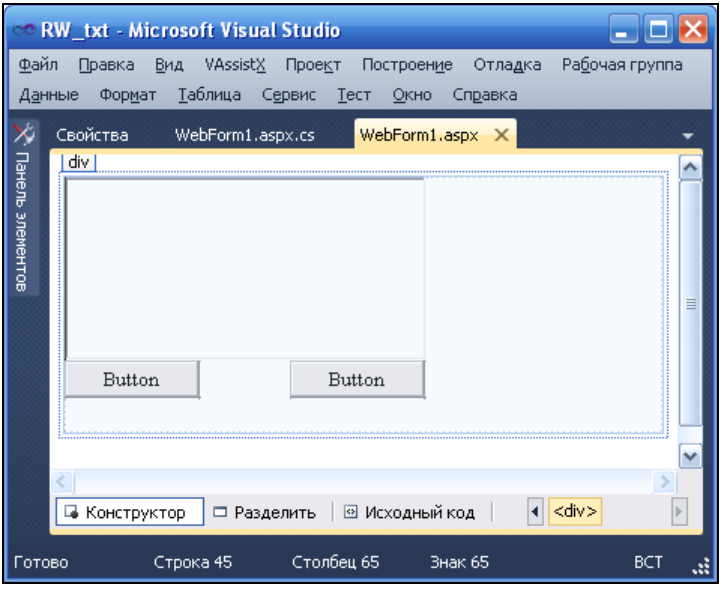


Рис. 14.1. Форма приложения в режиме конструктора

В начале программного кода объявляем имя файла `filename` и объект `КОДИРОВКА` в качестве внешних переменных по отношению к процедурам класса, чтобы они были видны из всех процедур обработки событий. При загрузке страницы `Page_Load` задаем надписи на кнопках `Button1` и `Button2`, а текстовое поле переводим в режим `Multiline`, т. е. разрешаем размещать текст на нескольких строчках. Далее, поскольку мы не можем разрешить пользователю выбирать папку на удаленном сервере для размещения файла, в качестве такой папки указываем папку, где расположено текущее Web-приложение. При этом мы не сможем воспользоваться методом `GetCurrentDirectory` класса `IO.Directory`, поскольку эта функция нам укажет на системную папку C:\Windows\system32. Поэтому воспользуемся методом `PhysicalApplicationPath` объекта `Request`. При обработке события "щелчок на кнопке" **Читать** читаем текстовый файл, используя объект `StreamReader`.

Так же как и при записи файла, данная процедура ничем не отличается от программирования ее в Windows-приложении.

Однако существует большая вероятность получить отказ в доступе к файлу. Операционная система, в зависимости от настроек, может выдать сообщение об ошибке сервера в вашем приложении вследствие отказа в доступе к вашему текстовому файлу. То есть можем получить запрет на запись файла на диск сервера. В этом сообщении об ошибке указано, что следует на вкладке **Безопасность** свойств текстового файла разрешить чтение/запись. Однако весьма вероятна ситуация, когда, щелкнув правой кнопкой мыши на изображении файла txt.txt и выбрав команду **Свойства**, мы получим свойства файла в таком виде, как показано на рис. 14.2.

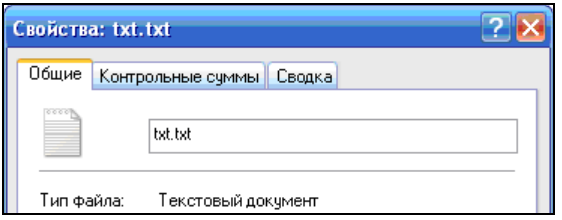


Рис. 14.2. Вкладки окна свойств файла

Как видно, в окне свойств файла txt.txt отсутствует вкладка **Безопасность**. Чтобы ее получить, следует в Проводнике выбрать пункт меню **Сервис | Свойства папки** и в появившемся окне на вкладке **Вид** снять флажок **Использовать простой общий доступ к файлам**. Далее — щелчок на кнопках **Применить** и **OK** (рис. 14.3).

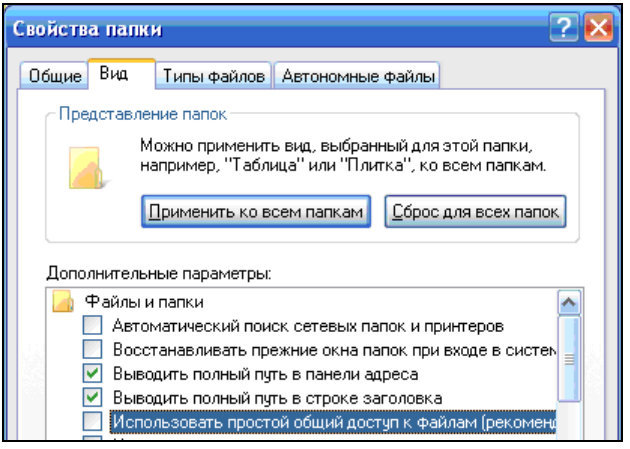


Рис. 14.3. Вкладки окна свойств папки

Теперь опять перейдем к свойствам файла txt.txt. Вкладка **Безопасность**, как видно, уже появилась. На ней нажмем кнопку **Добавить**, перейдя таким образом в

окно **Выбор**. Далее нажмем кнопки **Дополнительно**, **Поиск** и, выбрав в появившемся списке имя пользователя **ASP.NET**, щелкнем на кнопке **OK**. Теперь на вкладке **Свойства** в строках **Чтение** и **Запись** установим флажок разрешения и нажмем кнопки **Применить** и **OK** (рис. 14.4).

Теперь еще раз запустим Web-приложение и убедимся в его корректной работе. На рис. 14.5 показан фрагмент работы созданной Web-страницы.

Используя инструмент чтения/записи текстового файла, можно обмениваться данными между страницами Web-приложения, организовать форум на Web-сайте, гостевую книгу, счетчик посещений и проч.

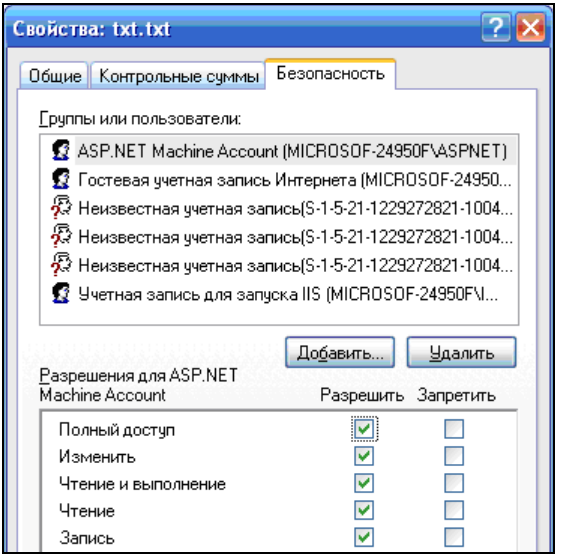


Рис. 14.4. Вкладка **Безопасность** окна свойств файла

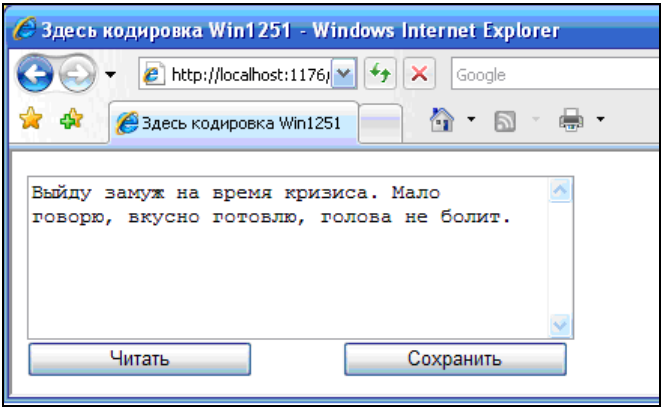


Рис. 14.5. Фрагмент работы Web-страницы, осуществляющей чтение/запись текстового файла

Убедиться в работоспособности Web-страницы можно, открыв соответствующее решение в папке RW_txt.

Пример 103. Программирование счетчика посещений сайта с использованием базы данных и объекта *Session*

Поставим задачу создать счетчик посещения вашего сайта. Каждое посещение будем фиксировать в виде записи в базе данных MS Access. Чтобы читатель мог реально повторить описываемые действия, подробно их рассмотрим. Запустим офисное приложение MS Access, далее в меню **Файл** выберем команду **Создать | Новая база данных**, файл — Web.mdb. Затем создадим "шапку" новой таблицы в режиме конструктора. Имена полей и типы данных приведены на рис. 14.6.

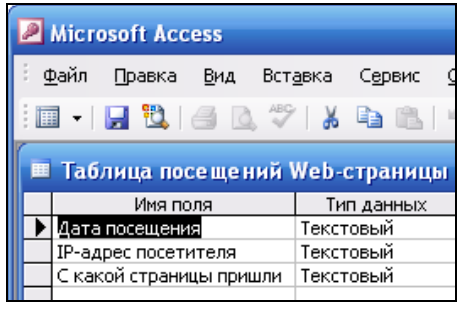


Рис. 14.6. Проектирование полей таблицы базы данных

Как видно, автор не предусмотрел поля типа **Счетчик**, а ведь нам нужно считать именно количество посещений, т. е. количество записей. Сделано это сознательно, т. к. количество записей в базе данных выяснить очень просто, например, с помощью SQL-запроса:

```
SELECT COUNT(*) FROM [Таблица посещений Web-страницы]
```

Теперь запустим Visual Studio 2010, закажем новый проект из шаблона **Empty ASP.NET Web Application C#**, назовем его **Counter**. К текущему проекту добавим Web-форму. Для этого в пункте меню **Project** выберем команду **Add New Item** и в появившемся окне дважды щелкнем по шаблону **Web Form**.

Здесь из раздела **Standard** панели **Toolbox** перенесем на Web-форму метку **Label**, куда будем выводить количество посещений данной страницы. Метку **Label** разместим пока в самом верху формы, но, в конце концов, ее следует расположить скромно где-нибудь в самом низу страницы.

Содержимое файла программной поддержки **WebForm1.aspx.cs** приведено в листинге 14.2.

Листинг 14.2. Счетчик посещений сайта

```
// Web-приложение, реализующее счетчик посещений сайта с использованием базы
// данных и объекта Session
using System;
// Добавляем эту директиву для краткости выражений:
using ОлеДиБи = System.Data.OleDb;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace Counter
{
   public partial class WebForm1 : System.Web.UI.Page
   {
      protected void Page_Load(object sender, EventArgs e)
      {
         Page.Title = "Счетчик посещений сайта";
         if (Page.IsPostBack == true) return;
         // При первой загрузке страницы выясняем IP-адрес посетителя сайта
         string IP_адрес = Request.UserHostAddress;
         string URL_адрес;
         try
         {  // Определение, с какой Web-страницы вы сюда пришли
            URL_адрес = Request.UrlReferrer.AbsoluteUri;
            Response.Write("<br /><br />Вы пришли на эту" +
                           " страницу со страницы " + URL_адрес);
         }
         catch // (System.NullReferenceException Ситуация)
         {  // Если пришли на эту страницу, набрав URL-адрес
            // в адресной строке браузера
            URL_адрес = "Адресная строка браузера";
            Response.Write("<br /><br />Вы пришли на эту " +
                        "страницу набрав URL-адрес в адресной строке");
         }
         Response.Write("<br /><br />Вы пришли на эту страницу " +
                        "с IP-адреса " + IP_адрес);
         // МАНИПУЛЯЦИИ С БД О ПОСЕЩЕНИИ ПОЛЬЗОВАТЕЛЯ.
         // Строка подключения
         string СтрокаПодкл =
           "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" +
                                        Server.MapPath("Web.mdb");
         // Создание экземпляра объекта Connection
         var Подключение = new ОлеДиБи.OleDbConnection(СтрокаПодкл);
```

```
try
{  // Открытие подключения
   Подключение.Open();
}
catch (Exception Ситуация)
{
   Response.Write("<br /><br />" + Ситуация.Message);
}
string SQL_запрос;
var Команда = new ОлеДиБи.OleDbCommand();
// ЕСЛИ ОБ ЭТОМ ПОСЕЩЕНИИ ЕЩЕ НЕТ ЗАПИСИ
if (Page.Session["ЕСТЬ ЛИ ЗАПИСЬ ОБ ЭТОМ ПОСЕЩЕНИИ?"] != "ДА")
{  // ДОБАВЛЕНИЕ ЗАПИСИ В БД О ПОСЕЩЕНИИ
   string ДАТА = System.DateTime.Now.ToString();
   // Строка SQL-запроса
   SQL_запрос =
      "INSERT INTO [Таблица посещений Web-страницы] " +
      "([Дата посещения], [IP-адрес посетителя], " +
      "[С какой страницы пришли]) VALUES ('" + ДАТА +
      "', '" + IP_адрес + "', '" + URL_адрес + "')";
   // Создание объекта Command с заданием SQL-запроса
   Команда.CommandText = SQL_запрос;
   // Для добавления записи эта команда обязательна
   Команда.Connection = Подключение;
   try
   {  // Выполнение команды SQL, т. е. ЗАПИСЬ В БД
      Команда.ExecuteNonQuery();
      Response.Write(
         "<br /><br />В таблицу БД посещений добавлена запись");
      // ТЕПЕРЬ ПОЯВИЛАСЬ ЗАПИСЬ ОБ ЭТОМ ПОСЕЩЕНИИ
      Page.Session["ЕСТЬ ЛИ ЗАПИСЬ ОБ ЭТОМ ПОСЕЩЕНИИ?"] = "ДА";
   }
   catch (Exception Ситуация)
   {
      Response.Write("<br /><br />" + Ситуация.Message);
   }
}
// ОПРЕДЕЛЕНИЕ КОЛИЧЕСТВА ЗАПИСЕЙ О ПОСЕЩЕНИИ.
// Новый SQL-запрос — это одна ячейка в новой таблице
SQL_запрос = "SELECT COUNT(*) FROM [Таб" +
             "лица посещений Web-страницы]";
Команда.CommandText = SQL_запрос;
```

```
            Команда.Connection = Подключение;
            // ExecuteScalar выполняет запрос и возвращает первую
            // колонку первого ряда таблицы запроса
            int КОЛ_ВО = 0;
            try
            {
               КОЛ_ВО = (int)Команда.ExecuteScalar();
            }
            catch (Exception Ситуация)
            {
               Response.Write("<br /><br />" + Ситуация.Message);
            }
            Подключение.Close();
            Label1.Text = "Количество посещений страницы = " + КОЛ_ВО;
        }
    }
}
```

Как видно из программного кода, программирование счетчика посещений реализовано при обработке событий загрузки Web-страницы `Page_Load`, причем при ее первой загрузке, когда `isPostBack = false`. Используя объект `Request`, выясняем IP-адрес посетителя данной Web-страницы. Далее с помощью этого же объекта `Request` в блоке `try...catch` определяем, с какой Web-страницы пользователь перешел на данную страницу. Если попытка (`try`) определить оказалась неудачной, то управление переходит на `Exception` и делается вывод, что пользователь оказался на данной странице, набрав URL-адрес в адресной строке своего браузера. Для вывода текста в форму использован метод `Response.Write`. Роль этой печати здесь вспомогательная, как бы отладочная. Читатель может ее закомментировать или вовсе удалить.

Далее происходит подготовка к записи строки из трех полей в базу данных. Вначале задаем строку подключения к базе данных, затем — непосредственное подключение `Connection`. Далее перед непосредственной записью строки посещения в базу данных происходит проверка, есть ли (зарегистрирована ли уже) запись об этом посещении. Может быть, пользователь в своем браузере просто обновил страницу путем нажатия клавиши <F5>. Вы можете себе представить, читатель, какой соблазн появляется у пользователя нажимать <F5>, если он видит, что после каждого нажатия увеличивается количество посещений сайта! Чтобы избежать этого, воспользуемся объектом страницы `Page.Session`.

В технологии ASP.NET каждое Web-приложение при обращении к нему пользователя (клиента) создает объект `Page.Session`, этот объект называют *сессией* или *сеансом пользователя*. Объект `Session` содержит в себе методы для создания коллекции данных. Причем созданная коллекция данных на одной странице Web-приложения оказывается доступной на любой другой странице данного приложе-

ния. Даже если пользователь перешел на другое Web-приложение (на другой Web-узел), а затем вернулся в данное приложение, то исходная коллекция данных сессии будет сохранена. Она будет сохранена до тех пор, пока пользователь не закрыл свой Web-браузер. Однако время жизни (лимит времени) сессии все же ограничено параметром `timeout`. По умолчанию время жизни равно 20 минутам, его можно регулировать в файле программной поддержки свойством объекта `Session.Timeout` или прямо в тексте программы.

Программировать создание сессии пользователя и получение из нее информации, доступной во всем приложении, несложно. Например, *включение* (добавление) в коллекцию данных сессии пар "имя (ключ) — значение" можно программировать таким образом: `Session("Имя пользователя")="Андрей"` или `Session.Add("Имя пользователя", "Андрей")`. Похоже на манипуляции с записями в хэш-таблице. На любой дугой странице данного Web-приложения имеем доступ к данным, записанным в коллекцию данных сессии: `Label1.Text = Session("Имя пользователя")` или `Label1.Text = Session.Item("Имя пользователя")`. Таким образом можно обмениваться данными между страницами Web-приложения.

Теперь вернемся к *добавлению* в БД записи о данном посещении страницы. После команды `ExecuteNonQuery` запись в базу данных будет добавлена, затем сразу после этого включаем в коллекцию текущей сессии пару `Page.Session("ЕСТЬ ЛИ ЗАПИСЬ ОБ ЭТОМ ПОСЕЩЕНИИ?")="ДА"`. Теперь эта пара будет существовать не только после нажатия клавиши <F5> (команда в браузере **Обновить страницу**), но и после того, как пользователь покинет данное Web-приложение и затем вернется в него опять. (Но при этом не будет закрывать свой Web-браузер.) Поэтому в программном коде, прежде чем добавлять запись в базу данных о посещении пользователя, предусмотрена проверка с помощью коллекции объекта `Session`.

Фрагмент работы счетчика посещений представлен на рис. 14.7.

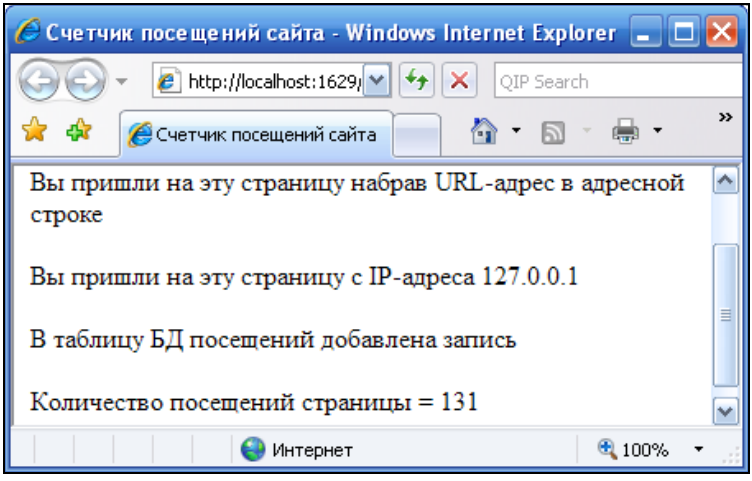


Рис. 14.7. Фрагмент работы Web-страницы, подсчитывающей посещеня

Теперь давайте проверим полученный IP-адрес. Является ли он действительно адресом нашего локального компьютера в Интернете? Для этого на панели задач вашего компьютера найдем значок **Подключение по локальной сети**. В контекстном меню выберем пункт **Состояние**. В появившемся окне на вкладке **Поддержка** увидим свой правильный IP-адрес. Он будет отличаться от выведенного в форму. Почему? Дело в том, что фрагмент работы данной программы получен на *локальном сервере*, поэтому и IP-адрес мы получили как адрес из внутренней подсети. После того как проект Web-приложения будет скопирован на *удаленный сервер*, предоставляющий Web-хостинг, мы получим корректный IP-адрес. Как указывалось ранее, три первых строки, выведенные в форму, носят отладочный характер и могут быть удалены из программного кода.

Убедиться в работоспособности Web-страницы можно, открыв решение Counter.sln в папке Counter.

Пример 104. Чтение/запись cookie-файлов

Cookie (от англ. *cookie* — печенье) — это небольшой фрагмент данных (обычно до 4096 байт), созданный Web-сервером и хранимый на компьютере клиента (пользователя) в виде файла. Каждый раз при запросе какой-либо страницы сайта браузер пользователя посылает серверу в HTTP-запросе этот cookie-файл. Данные, записанные в cookie-файле, обычно используются для аутентификации пользователя, т. е. для его распознавания среди прочих пользователей, для хранения его персональных предпочтений, настроек пользователя, для ведения статистики пользователей и проч. В самом начале использования cookie служило для определения, посещал ли пользователь сайт ранее.

Зайдя на любой сайт, вы можете увидеть активные на данном сайте cookies, набрав в адресной строке браузера следующее:

```
javascript:alert("Cookies: "+document.cookie)
```

В ответ ваш Web-браузер покажет вам примерно такое окно, как показано на рис. 14.8.

Рис. 14.8. Отображение активного cookie на некотором сайте

В этом окне содержаться те сведения, которые сервер некоторого Web-узла закодировал о вас в cookie-файле, записанном на ваш компьютер.

В данном примере мы научимся читать cookie при загрузке страницы и записывать cookie-файл при нажатии пользователем командной кнопки. То есть постановка задачи следующая. Web-страница предлагает посетителю ввести данные о себе: его имя и род занятий. При нажатии кнопки **Запись Cookie** введенные в текстовые поля сведения будут записаны в cookie-файл. Этот cookie-файл будет храниться на компьютере пользователя сутки. В течение этих суток, каждый раз вызывая данную страницу, в текстовых полях мы будем видеть введенные нами сведения, которые мы можем тут же исправлять и опять записывать в cookie. Фрагмент функционирования подобной Web-страницы приведен на рис. 14.9.

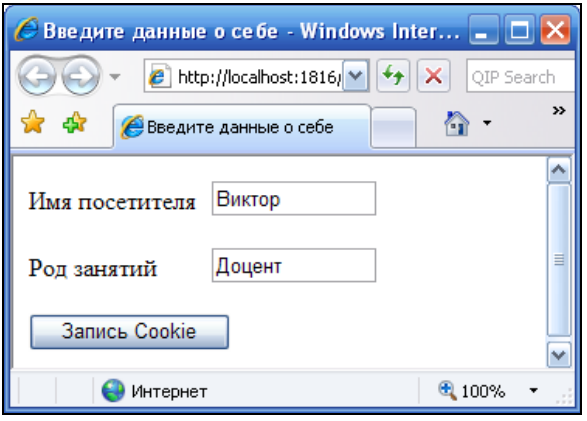


Рис. 14.9. Фрагмент работы Web-страницы записи cookie

Для реализации поставленной задачи запустим Visual Studio 2010, закажем новый проект из шаблона **Empty ASP.NET Web Application**, назовем его Cookie. К текущему проекту добавим Web-форму. Для этого в пункте меню **Project** выберем команду **Add New Item** и в появившемся окне дважды щелкнем на шаблоне **Web Form**.

Затем из раздела **Standard** панели **Toolbox** перенесем в проектируемую Web-форму две метки **Label**, два текстовых поля **TextBox** и командную кнопку **Button**. Содержимое файла программной поддержки WebForm1.aspx.cs приведено в листинге 14.3.

Листинг 14.3. Чтение/запись cookie-файлов

```
// Чтение/запись cookie-файлов. Web-страница предлагает посетителю ввести
// данные о себе: имя и род занятий. При нажатии кнопки "Запись Cookie"
// введенные в текстовые поля сведения будут записаны в cookie-файл.
// Этот cookie-файл будет храниться на компьютере пользователя сутки.
// В течение этих суток, каждый раз вызывая данную страницу, в текстовых
// полях мы будем видеть введенные нами сведения, которые мы можем тут же
// исправлять и опять записывать в cookie
```

```
using System;
using System.Web;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace Cookie
{
   public partial class WebForm1 : System.Web.UI.Page
   {
      protected void Page_Load(object sender, EventArgs e)
      {
         Page.Title = "Введите данные о себе";
         // При повторной отправке выйти из процедуры
         if (Page.IsPostBack == true) return;
         Label1.Text = "Имя посетителя";
         Label2.Text = "Род занятий";
         Button1.Text = "Запись Cookie";
         // ЧТЕНИЕ COOKIE.
         // Cookie может быть целая коллекция:
         // HttpCookieCollection CookieN;
         HttpCookie Cookie1;
         // Читаю только один раздел cookie "О посетителе страницы"
         Cookie1 = Request.Cookies.Get("О посетителе страницы");
         // Если на машине клиента нет такого cookie
         if (Cookie1 == null) return;
         // Если есть, то заполняю текстовые поля из cookie
         TextBox1.Text = Cookie1["Имя посетителя"];
         TextBox2.Text = Cookie1["Род занятий посетителя"];
      }

      protected void Button1_Click(object sender, EventArgs e)
      {  // ЗАПИСЬ COOKIE.
         HttpCookie Cookie1 = new HttpCookie("О посетителе страницы");
         // Запись двух пар "имя (ключ) - значение"
         Cookie1["Имя посетителя"] = TextBox1.Text;
         Cookie1["Род занятий посетителя"] = TextBox2.Text;
         // Установка даты удаления cookie: сейчас плюс один день
         Cookie1.Expires = DateTime.Now.AddDays(1);
         // Добавление раздела "О посетителе страницы" в cookie-файл
         Response.Cookies.Add(Cookie1);
      }
   }
}
```

Здесь во время загрузки страницы `Page_Load` при `isPostBack = false` инициализируем надписи на метках **Label**, а затем читаем cookie-файл посредством объекта класса `Request`. В данном случае функция `Request.Cookies.Get` возвращает раздел cookie-файла "О посетителе страницы". В данной программе мы предусмотрели только один раздел, хотя их может быть несколько (коллекция `HttpCookieCollection`). Если компьютер пользователя не содержит данного cookie-файла, то программируем выход `return` из данной процедуры. Если cookie прочитан, то текстовые поля заполняем соответствующими значениями.

При обработке события "щелчок на кнопке" **Запись Cookie** происходит запись в cookie-файл, в его раздел "О посетителе страницы" двух пар "имя—значение" `Имя посетителя` и `Род занятий`, скопированных из текстовых полей. В предложении `Cookie.Expires` указываем срок хранения cookie. Если дата не указана, то cookies удаляются сразу, как только пользователь закроет браузер. Например, интернет-магазин может использовать cookies для хранения названий товаров, которые пользователь выбрал и поместил в виртуальную корзину покупок. В таком случае даже если пользователь закроет браузер, не совершив покупки, то при последующем посещении интернет-магазина ему не придется формировать корзину заново.

Вернемся к нашему программному коду. Оператор `Add(Cookie1)` объекта `Response.Cookies` добавляет раздел "О посетителе страницы" в cookie-файл. Теперь давайте найдем этот cookie-файл на винчестере вашего компьютера. Различные браузеры для разных операционных систем пишут cookies-файлы в различные папки. Скорее всего, вы найдете данный cookie в папке C:\Documents and Settings\Администратор\Cookies. Структура имени cookie-файла следующая:

```
Имя_пользователя_компьютера@имя_сервера[1]
```

Поскольку в данном случае речь идет о локальном сервере, то, скорее всего, имя обсуждаемого в данном разделе cookie-файла будет:

```
администратор@localhost[2].txt
```

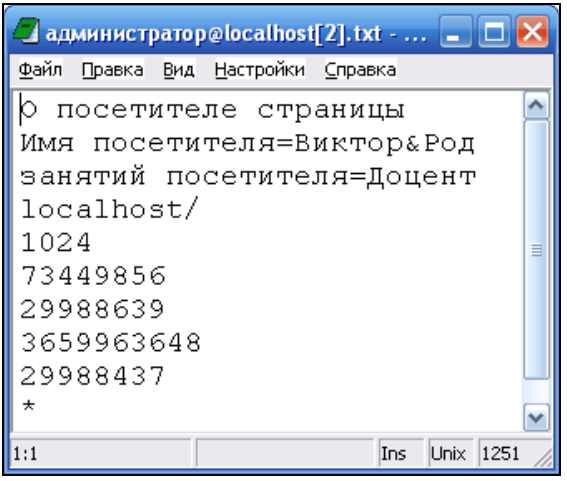


Рис. 14.10. Содержимое cookie-файла

Поскольку cookie — это текстовый файл, его можно легко открыть, например, с помощью Блокнота (рис. 14.10).

Как видно, первая строка содержимого cookie-файла — это имя раздела, затем пары "имя—значение", запрограммированные нами в программе, далее имя сервера и служебная информация.

В заключение замечу, что cookies представляют собой лишь данные, а не программный код, т. е. они не могут стереть или прочитать информацию с компьютера пользователя, поэтому не стоит демонизировать cookie.

Убедиться в работоспособности Web-страницы можно, открыв решение Cookie.sln в папке Cookie.

Пример 105. Вывод изображения в Web-форму

Демонстрация изображения в Web-форме является тривиальной задачей. Для ее решения Visual Studio имеет класс `Image` пространства имен `System.Web.UI.WebControls`. С помощью свойства `ImageUrl` этого класса задаем URL-адрес изображения, а с помощью `AlternateText` — альтернативный текст, отображаемый в элементе управления **Image**, когда изображение недоступно. Немножко усложним задачу и сформулируем ее так: на странице имеем некоторое изображение, при щелчке мышью на нем изображение увеличивается вдвое без перезагрузки Web-страницы.

Сначала решим эту задачу, используя исключительно HTML-разметку с небольшим включением программного кода на JavaScript. Очень удобно выполнить HTML-разметку в редакторе Visual Studio, поэтому запустим систему Visual Studio 2010 и создадим новый проект из шаблона **Empty ASP.NET Web Application**, назовем его Web-изображение. К текущему проекту добавим HTML-страницу. Для этого в пункте меню **Project** выберем команду **Add New Item** и в появившемся окне дважды щелкнем по шаблону **HTML Page**. Далее на вкладке **HTMLPage1.htm** введем текст, представленный в листинге 14.4.

Листинг 14.4. Увеличение изображения в Web-форме (HTML-код)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
   <title>Щелкнуть мышью для увеличения</title>
</head>
<body>
   <img src="poryv.png" width="100" height="100"
        alt="Двойной щелчок возвращает в исходный размер"
```

```
        onclick="this.src='poryv.png';this.height=200;this.width=200"
        ondblclick="this.src='poryv.png';this.height=100;this.width=100" />
</body>
</html>
```

На самом деле вводить придется лишь разметку между тегами `<body>` и `</body>`. Остальной код управляющая среда сгенерировала сама. В приведенной разметке в качестве атрибутов тега `<img>`, размечающего изображение, записаны две обработки событий `onclick` (щелчок мышью на изображении) и `ondblclick` (двойной щелчок) на JavaScript. Теперь для просмотра этой страницы в браузере мы можем нажать клавишу <F5> или непосредственно открыть файл HTMLPage1.htm, например, в Internet Explorer. При этом мы увидим следующее (рис. 14.11).

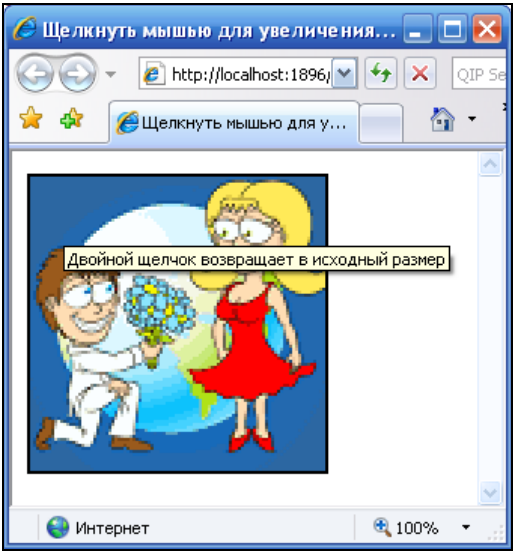


Рис. 14.11. Щелчок мышью увеличивает изображение

Теперь решим эту же задачу программированием на Visual C# 2010. Для этого добавим в текущий проект новую Web-форму. В пункте меню **Project** выполним команду **Add New Item** и в появившемся окне выберем шаблон **Web Form**. На вкладке конструктора формы перетащим из панели **Tollbox** (эту панель можно вызвать, например, используя комбинацию клавиш <Ctrl>+<Alt>+<X>) из раздела **Standard** элемент управления изображения **Image**, а на вкладке программного кода **WebForm1.aspx.cs** напишем текст, представленный в листинге 14.5.

Листинг 14.5. Увеличение изображения в Web-форме (Visual C# 2010-код)

```
// На странице имеем изображение - файл poryv.png, при щелчке мышью на нем
// изображение увеличивается вдвое без перезагрузки Web-страницы
```

```
using System;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace Web_изображение
{
   public partial class WebForm1 : System.Web.UI.Page
   {
      protected void Page_Load(object sender, EventArgs e)
      {
         Page.Title = "Щелкнуть мышью для увеличения";
         // Указываем виртуальный путь к файлу изображения
         Image1.ImageUrl = Request.ApplicationPath + "poryv.png";
         // Получаем URL, который используется в браузере
         string адрес = ResolveClientUrl(Image1.ImageUrl);
         // Добавляем атрибут Alt
         Image1.AlternateText = "Двойной щелчок возвращает в исходный размер";
         // Добавляем два события JavaScript
         Image1.Attributes.Add("onclick", "this.src='" +
                           адрес + "';this.height=200;this.width=200");
         Image1.Attributes.Add("ondblclick", "this.src='" +
                           адрес + "';this.height=100;this.width=100");
      }
   }
}
```

Как видно из программного кода, кроме двух очевидных свойств `ImageUrl` и `AlternateText` объекта `Image1` свойство `Attributes` добавляет объекту `Image1` два события JavaScript. Нам пришлось включать таким образом события JavaScript, поскольку объект `Image1` не имеет событий мыши.

Поставим другую, более интересную задачу. На Web-странице имеем изображение, например, мужчины. Это изображение используем для ссылки на другую Web-страницу, скажем, на WebForm1.aspx. Причем при наведении на него указателя мыши происходит смена изображения на изображение женщины.

Для решения этой задачи добавим в текущий проект еще одну HTML-страницу и на вкладке разметки **HTMLPage2.htm** напишем текст, приведенный в листинге 14.6.

Листинг 14.6. Смена изображения в Web-форме (HTML-код)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
   <title>Смена изображения при наведении указателя мыши</title>
```

```
</head>
<body>
  <a href="HTMLPage1.htm">
  <img onmouseover="this.src='g.jpg'" onmouseout="this.src='m.jpg'"
       alt="Щелкните, чтобы перейти на HTMLPage1.htm"
       src="m.jpg" border="0" /> </a>
</body>
</html>
```

Здесь тег `<a>` обеспечивает гиперссылку на другую Web-страницу — HTMLPage1.htm, причем в качестве гиперссылки используется изображение m.jpg (изображение мужчины). Это изображение меняется на изображение женщины (файл g.jpg) при наведении на него указателя мыши (JavaScript-событие `onmouseover`) и возвращается в исходное, когда указатель мыши покидает элемент (событие `onmouseout`).

Установим в качестве стартовой страницы проекта файл HTMLPage2.htm. Для этого в контекстном меню окна **Solution Explorer** для этого файла укажем **Set As Start Page**. Внешний вид данной Web-страницы в браузере показан на рис. 14.12.

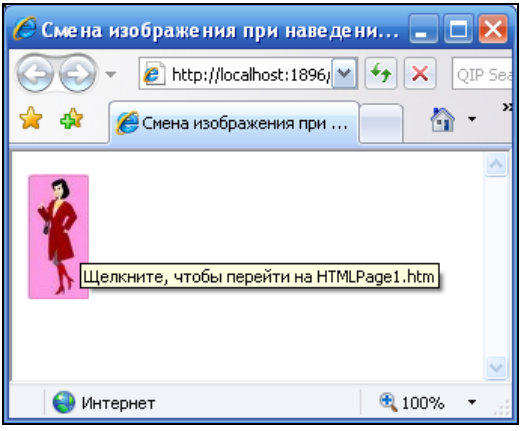


Рис. 14.12. Смена изображения при наведении указателя мыши

Для решения этой же задачи с помощью Visual Studio 2010 в текущий проект добавим новую Web-форму — WebForm2.aspx. В конструкторе формы из раздела **Standard** панели **Tollbox** добавим элемент управления **ImageButton**, отображающий изображение и отвечающий на нажатия на нем кнопки мыши. На вкладке программного кода этой формы напишем текст программы, приведенной в листинге 14.7.

Листинг 14.7. Смена изображения в Web-форме (Visual C# 2010-код)

```
// На Web-странице имеем изображение, например мужчины, - файл m.jpg. Это
// изображение используем для ссылки на другую Web-страницу, например, на
```

```
// WebForm1.aspx. Причем при наведении на него указателя мыши происходит
// смена изображения на изображение женщины - файл g.jpg
using System;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace Web_изображение
{
   public partial class WebForm2 : System.Web.UI.Page
   {
      protected void Page_Load(object sender, EventArgs e)
      {
         // Элемент управления ImageButton отображает изображение
         // и отвечает за нажатия на нем указателя мыши
         ImageButton1.PostBackUrl = "WebForm1.aspx";
         // Указываем виртуальный путь к файлу изображения
         ImageButton1.ImageUrl = Request.ApplicationPath + "m.jpg";
         // Задаем альтернативный текст
         ImageButton1.AlternateText =
                        "Щелкните, чтобы перейти на WebForm1.aspx";
         // Добавляем два события JavaScript
         ImageButton1.Attributes.Add("onmouseover", "this.src='g.jpg'");
         ImageButton1.Attributes.Add("onmouseout", "this.src='m.jpg'");
      }
   }
}
```

Как видно из программного кода, мы использовали те же приемы, что и в предыдущей задаче. Убедиться в работоспособности Web-страниц, рассмотренных в данном разделе, можно, открыв соответствующее решение в папке Web_ изображение.

Пример 106. Формирование изображения методами класса *Graphics* и вывод его в Web-форму

При создании Web-страниц часто бывает удобно сначала создать изображение, что-либо на нем нарисовать, например, график какой-нибудь зависимости, актуальной именно в момент загрузки страницы, затем это изображение записать на диск сервера и вывести его клиенту в обозреватель. Таким графиком может быть, скажем, график статистики посещений сайта по месяцам, по неделям, по дням.

Задача, решаемая в данном разделе, состоит в следующем. Во время загрузки Web-страницы создать изображение, методами класса `Graphics` вывести на это изображение текстовую строку, представляющую текущую дату. С целью демонстрации возможностей методов `Graphics` развернуть данную строку на некоторый угол относительно горизонта. Далее сохранить рисунок в текущем каталоге сервера и вывести его на Web-страницу.

Для решения этой задачи запустим Visual Studio 2010, закажем новый проект шаблона **Empty ASP.NET Web Application**. Добавим Web-форму, для этого в пункте меню **Project** выберем команду **Add New Item**, в появившемся окне укажем шаблон **Web Form** и добавим его к нашему проекту. На вкладке дизайнера формы **WebForm1.aspx** добавим элемент управления **Image**. А на вкладке **WebForm1.aspx.cs** введем программный код, представленный в листинге 14.8.

Листинг 14.8. Формирование изображения и вывод его в Web-форму

```
// Web-страница формирует файл изображения методами класса Graphics. На
// изображение выводится текстовая строка, наклоненная к горизонту на 356
// градусов (наклон вверх). Далее этот файл изображения отображается в форме
using System;
// Добавляем данное пространство имен для сокращения программного кода:
using System.Drawing;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace ТекстНаклWeb
{
   public partial class WebForm1 : System.Web.UI.Page
   {
      protected void Page_Load(object sender, EventArgs e)
      {  // Создаем точечное изображение размером 215 x 35 точек
         // с глубиной цвета 24
         Bitmap Рисунок = new Bitmap(215, 35, System.Drawing.Imaging.
                                    PixelFormat.Format24bppRgb);
         // Создаем новый объект класса Graphics из изображения РАСТР
         Graphics Графика = Graphics.FromImage(Рисунок);
         // Теперь становятся доступными методы класса Graphics!
         // Заливка поверхности указанным цветом
         Графика.Clear(Color.AliceBlue);
         // или Графика.Clear(ColorTranslator.FromHtml("#ECECFF"));
         // Вывод в строку полной даты:
         string Дата = string.Format("Сегодня {0:D}", DateTime.Now);
         // Разворачиваем мир на 356 градусов по часовой стрелке
         Графика.RotateTransform(356.0F);
         // Выводим на изображение текстовую строку Дата,
```

```
            // x=5, y=15 - координаты левого верхнего угла строки
            Графика.DrawString(Дата, new Font("Times New Roman", 14,
                               FontStyle.Regular), Brushes.Red, 5, 15);
            // Определяем физический путь файла для текущего Web-узла,
            // сохраняем изображение в файле risunok.jpg каталога Web-узла
            Рисунок.Save(Request.PhysicalApplicationPath + "risunok.jpg",
                         System.Drawing.Imaging.ImageFormat.Jpeg);
            // Возможность вывода изображения в исходящий поток ответа HTTP:
            // Рисунок.Save(Response.OutputStream, System.Drawing.
            //              Imaging.ImageFormat.Jpeg);
            // Цвет и ширина рамки рисунка:
            Image1.BorderColor = Color.Red; Image1.BorderWidth = 2;
            // Указываем виртуальный путь к файлу изображения
            Image1.ImageUrl = Request.ApplicationPath + "risunok.jpg";
            // Освобождение ресурсов
            Рисунок.Dispose(); Графика.Dispose();
        }
    }
}
```

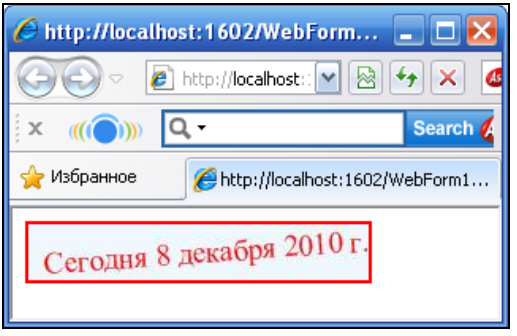


Рис. 14.13. Вывод в форму изображения, сформированного программно

Как видно из программного кода, при загрузке страницы создаем точечное изображение указанного размера, формат `Format24bppRgb` указывает, что отводится 24 бита на точку: по 8 бит на красный, зеленый и синий каналы. Данное изображение позволяет создать новый объект класса `Graphics` методом `FromImage`. Теперь разворачиваем поверхность рисования на 356° методом `RotateTransform` и выводим на поверхность рисования текстовую строку с текущей датой. Задавая физический путь файла изображения, методом `Save` сохраняем его в каталоге Web-узла в формате JPEG. В комментарии приведена возможность вывода рисунка в исходящий поток ответа HTTP, при этом изображение не будет записываться на диск, но пользователь будет видеть его в браузере. Далее элементу управления `Image1` указываем виртуальный путь к файлу изображения. Замечу, что физический путь не

должен отправляться клиенту, поскольку он может использоваться злоумышленниками для получения сведений о приложении.

На рис. 14.13 показан фрагмент работы программы.

Убедиться в работоспособности Web-страницы, рассмотренной в данном примере, можно, открыв соответствующее решение в папке ТекстНаклWeb.

Пример 107. Гостевая книга

Допустим, на своем сайте вы обсуждаете какую-либо тему и хотите, чтобы посетители сайта оставляли свои мнения, впечатления, пожелания, замечания, адресованные владельцу или будущим посетителям. Причем так, чтобы сообщения, написанные посетителями, были бы тут же *отображены* и *сохранены*, и тем самым стимулировали желание новых посетителей написать свое сообщение. Такое программное обеспечение называют *гостевой книгой сайта*.

В гостевой книге следует предусмотреть поля для ввода имени посетителя, адреса его электронной почты, а также поле для непосредственного сообщения. Все записи, оставленные посетителями сайта, будем сохранять в текстовом файле kniga.txt на *винчестере сервера*, предоставляющего Web-хостинг. Впрочем, вместо текстового файла можно воспользоваться какой-либо базой данных.

Для программирования гостевой книги запустим Visual Studio 2010, в окне **New Project** выберем шаблон **Empty ASP.NET Web Application**, в поле **Name** укажем имя нового решения ГостеваяКнига и щелкнем на кнопке **OK**. Теперь добавим к текущему проекту Web-форму. Для этого в пункте меню **Project** выберем команду **Add New Item**, в появившемся окне укажем шаблон **Web Form** и щелкнем на кнопке **Add**.

На вкладке конструктора формы в панели элементов раздела **Standard** нам понадобятся четыре метки **Label**, три текстовых поля **TextBox**, соответственно, для имени пользователя, его электронной почты и сообщения, одна командная кнопка **Button** с надписью "Добавить сообщение" и сетка данных **GridView** из раздела **Data** для отображения всех вводимых записей. Кроме того, возле каждого текстового поля разместим валидатор **RequiredFieldValidator** (раздел **Validation**), проверяющий факт *заполнения* текстовых полей. Правильность заполнения проверять не будем, поскольку в данной ситуации если мы будем слишком "принципиальничать" по поводу вводимых данных, то посетитель потеряет терпение и уйдет на другой ресурс (не наш!). Не забываем, что мы гораздо больше заинтересованы во всех новых сообщениях, чем посетители их написать. С другой стороны (другая крайность), если мы вовсе не будем контролировать заполнения текстовых полей, то у недобросовестного пользователя появится соблазн нажимать кнопку **Добавить сообщение** без заполнения полей и заполнять таблицу сетки данных пустыми строками, что будет вовсе *дискредитацией замысла* гостевой книги. В конструкторе формы разместим выбранные элементы управления так, как показано на рис. 14.14.

После размещения в конструкторе необходимых элементов управления переходим на вкладку программного кода. Текст программы представлен в листинге 14.9.

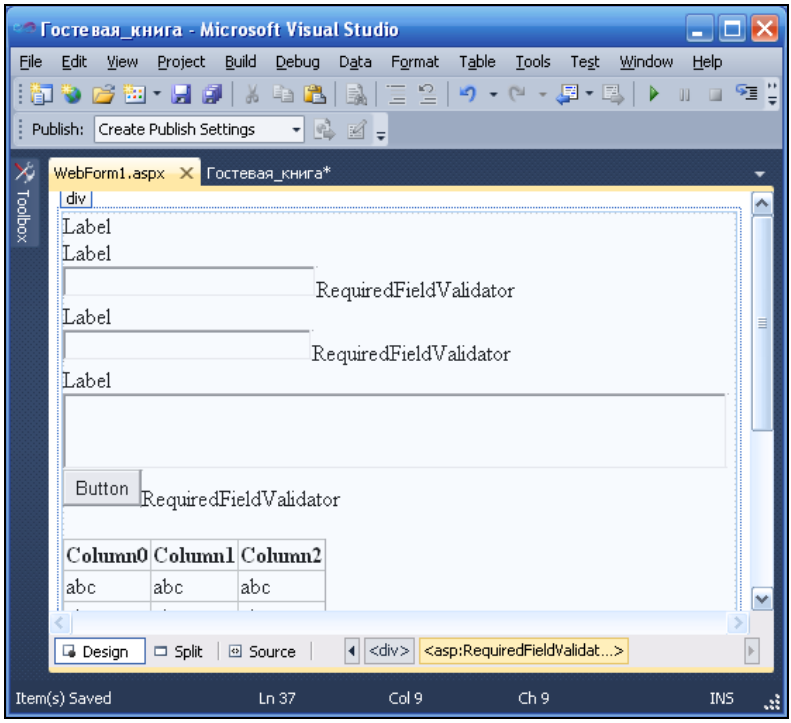


Рис. 14.14. Размещение элементов управления в конструкторе формы

Листинг 14.9. Гостевая книга

```
// Данная Web-страница приглашает посетителя оставить какие-либо записи,
// которые могут прочитать другие посетители страницы. Записи сохраняются
// в текстовом файле kniga.txt. Записи отображаются на Web-странице
// с помощью сетки данных GridView
using System;
using System.Web.UI.WebControls;
// Добавляем эти две директивы для сокращения программного кода:
using System.Data;
using System.IO;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace ГостеваяКнига
{
   public partial class WebForm1 : System.Web.UI.Page
   {
      DataTable Таблица = new DataTable();
      StreamReader Читатель;
```

```
StreamWriter Писатель;
protected void Page_Load(object sender, EventArgs e)
{
   Label1.Text = "ВЫ МОЖЕТЕ НАПИСАТЬ КАКОЕ-НИБУДЬ " +
                 "СООБЩЕНИЕ В НАШЕЙ ГОСТЕВОЙ КНИГЕ";
   Label2.Text = "Ваше имя:";
   Label3.Text = "Ваш E-mail:";
   Label4.Text = "Ваше сообщение:";
   Button1.Text = "Добавить сообщение";
   // Разрешаем многострочие:
   TextBox3.TextMode = TextBoxMode.MultiLine;
   // Контролируем обязательность заполнения всех полей:
   RequiredFieldValidator1.ErrorMessage =
                       "* Следует заполнить это текстовое поле";
   RequiredFieldValidator1.ControlToValidate = "TextBox1";
   RequiredFieldValidator2.ErrorMessage =
                       "* Следует заполнить это текстовое поле";
   RequiredFieldValidator2.ControlToValidate = "TextBox2";
   RequiredFieldValidator3.ErrorMessage =
                       "* Следует заполнить это текстовое поле";
   RequiredFieldValidator3.ControlToValidate = "TextBox3";
   Таблица.Columns.Add("Дата");   Таблица.Columns.Add("Имя");
   Таблица.Columns.Add("E-mail"); Таблица.Columns.Add("Сообщение");
   GridView1.BorderWidth = Unit.Pixel(2);
   GridView1.BorderColor = System.Drawing.Color.Gray;
   // Расстояние между содержимым ячейки и ее границей:
   GridView1.CellPadding = 3;
   GridView1.Caption = "Записи гостевой книги";
   GridView1.CaptionAlign = TableCaptionAlign.Right;
   ЗаполнитьGridView();
}

public void ЗаполнитьGridView()
{  // Эта процедура читает файл kniga.txt (если его нет, то создает),
   // разбивает каждую строку файла на четыре части (дата, имя, e-mail
   // и сообщение) и заполняет этими частями строки таблицы. Затем
   // записывает эту таблицу в сетку данных GridView.
   // ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
   // Открываем файл kniga.txt, а если его нет, то его создаем:
   var Открыватель = new FileStream(Request.PhysicalApplicationPath +
                           "kniga.txt", FileMode.OpenOrCreate);
   // Открываем поток для чтения всех записей из файла
   Читатель = new StreamReader(Открыватель);
```

```
        // В качестве разделителя частей строки файла выбираем Tab,
        // поскольку Tab невозможно ввести в текстовое поле. После нажатия
        // клавиши <Tab> происходит переход в следующее текстовое поле
        char[] Разделитель = new char[] { '\t' };  // - массив
        string[] Массив_частей_строки;
        while (Читатель.EndOfStream == false)
        {
           string Одна_строка = Читатель.ReadLine();
           // Функция Split делит строку на четыре части и присваивает
           // каждую часть элементам массива
           Массив_частей_строки = Одна_строка.Split(Разделитель);
           // Загружаем, т. е. заполняем одну строку таблицы
           Таблица.LoadDataRow(Массив_частей_строки, true);
        }
        GridView1.DataSource = Таблица;
        // Обновление сетки данных:
        GridView1.DataBind();
        Таблица.Clear();
        Читатель.Close(); Открыватель.Close();
     }

     protected void Button1_Click(object sender, EventArgs e)
     {  // Щелчок на кнопке "Добавить".
        // Открываем поток для добавления данных в конец файла
        Писатель = new StreamWriter(Request.
                         PhysicalApplicationPath + "kniga.txt", true);
        // true означает разрешить добавление строк в файл.
        // Записываем в файл новое сообщение, между полями - символ табуляции
        Писатель.WriteLine("{0:D} \t{1} \t{2} \t{3}", DateTime.Now,
                         TextBox1.Text, TextBox2.Text, TextBox3.Text);
        // Очищаем поля и закрываем поток
        TextBox1.Text = string.Empty; TextBox2.Text = string.Empty;
        TextBox3.Text = string.Empty;
        Писатель.Close();
        ЗаполнитьGridView();
     }
  }
}
```

Как видно из программного кода, вначале создаем таблицу данных класса `DataTable` и два потока данных для чтения и записи файлов так, чтобы они были видны из всех процедур класса. При обработке события загрузки формы организуем контроль обязательности заполнения текстовых полей формы, задаем "шапку"

таблицы (т. е. названия колонок таблицы), а последней командой вызываем процедуру `ЗаполнитьGridView`.

Эта процедура в цикле `while` читает строки текстового файла kniga.txt и заполняет ими строки таблицы данных методом `LoadDataRow()`. Далее заполненную таблицу данных указываем в качестве источника данных (`DataSource`) для сетки данных `GridView1`, предназначенной для визуализации (отображения) таблицы в форме. После этого важно очистить таблицу данных методом `Clear()`.

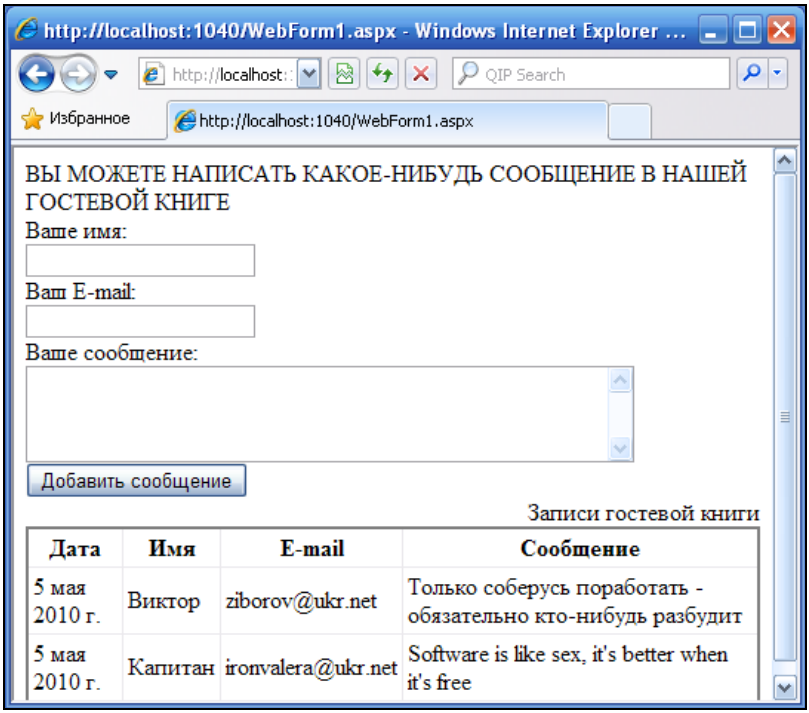


Рис. 14.15. Интерфейс гостевой книги

При обработке события "щелчок на кнопке" открываем поток данных `Писатель` для добавления данных в конец файла kniga.txt и записываем в этот файл текстовую строку, содержащую текущую дату, имя посетителя Web-страницы, его e-mail и непосредственно его сообщение. Причем в качестве разделителя между этими текстовыми строками мы используем управляющий символ Tab, поскольку его невозможно ввести в текстовое поле, т. к. после нажатия клавиши <Tab> предусмотрен переход в следующее текстовое поле. После закрытия потока данных `Писатель` вызываем процедуру `ЗаполнитьGridView`, которая как бы начинает опять "с чистого листа": снова открывает поток для чтения файла и опять заполняет таблицу данных, которая, как мы помним, была очищена от предыдущих данных. Именно поэтому строки (команды) процедуры `ЗаполнитьGridView` были оформлены в отдельную процедуру, поскольку заполнение таблицы и ее визуализация *происходят дважды*: один раз при загрузке формы, а другой при добавлении новой записи.

Фрагмент работы гостевой книги представлен на рис. 14.15.

Убедиться в работоспособности данной программы можно, открыв соответствующее решение в папке ГостеваяКнига.

Пример 108. Программирование капча

Слово Capcha — это английская аббревиатура выражения "Completely Automated Public Turing test to tell Computers and Humans Apart", которое можно перевести, как "полностью автоматический публичный тест Тьюринга для различия компьютеров и людей". Другими словами, это компьютерное тестирование, используемое в каком-либо пользовательском интерфейсе для того, чтобы определить, кем является пользователь системы: человеком или компьютером.

Процедура капча чаще всего используется при необходимости предотвратить вредоносное использование интернет-сервисов ботами (программами-роботами). В частности для предотвращения автоматических (без участия человека) отправок сообщений, регистраций, скачиваний файлов, массовых рассылок, мошенничества, сетевых атак и т. п. Как вы понимаете, бот (или робот, или интернет-робот) — это специальная программа, выполняющая автоматически какие-либо действия через те же интерфейсы, что и обычный пользователь-человек.

Для противодействия таким автоматическим действиям интернет-роботов сейчас довольно-таки широко применяется обсуждаемая в данном разделе капча. Основной замысел тестирования с помощью капча в том, чтобы предложить пользователю такую задачу, которую очень легко тот может решить, но которую несоизмеримо сложно предоставить для решения компьютеру. В основном это задачи на *распознавание символов*. Это весьма актуальная задача; автор заметил на сайте оффшорного программирования **www.rentacoder.com** (это сайт-посредник между работодателем и программистом), что имеет место спрос на процедуру капча.

Итак, для решения этой задачи запустим Visual Studio 2010, закажем новый проект из шаблона **Empty ASP.NET Web Application**, укажем в поле **Name** имя Капча и щелкнем на кнопке **OK**. К текущему проекту добавим Web-форму. Для этого в меню **Project** выберем команду **Add New Item** и в появившемся окне дважды щелкнем на шаблоне **Web Form**. Теперь в конструкторе формы на вкладке **WebForm1.aspx** из раздела **Standard** панели **Toolbox** перенесем в форму элементы управления: изображение **Image**, текстовое поле **TextBox**, командную кнопку **Button**, метку **Label** и гиперссылку **HyperLink**. Расположим их в форме, как показано на рис. 14.16.

На вкладке файла программной поддержки **WebForm1.aspx.cs** введем код, представленный в листинге 14.10.

Листинг 14.10. Программирование капча

```
// Программа тестирует пользователя на "человечность", предлагает
// распознать искаженную строку символов
using System;
```

```
using System.Drawing;
using System.Drawing.Imaging;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace Капча
{
   public partial class WebForm1 : System.Web.UI.Page
   {
      protected void Page_Load(object sender, EventArgs e)
      {
         Page.Title = "Капча";
         // При повторной отправке данных оставляем все как есть:
         if (Page.IsPostBack == true) return;
         // Получить сгенерированный рисунок:
         Bitmap РАСТР = ГенерироватьРисунок();
         // Сохранить сгенерированный рисунок:
         // bmp.Save(Response.OutputStream, ImageFormat.Jpeg);
         РАСТР.Save(Request.PhysicalApplicationPath + "risunok.jpg",
                    ImageFormat.Jpeg);
         РАСТР.Dispose();
         // Указываем виртуальный путь к файлу изображения:
         Image1.ImageUrl = Request.ApplicationPath + "risunok.jpg";
         Label1.Text = "Введите символы, изображенные на картинке";
         Button1.Text = "Регистрация";
         TextBox1.Focus(); TextBox1.Font.Size = 18;
         HyperLink1.Text = "Обновить картинку";
         // При обновлении картинки отправляем на загрузку самой себя:
         HyperLink1.NavigateUrl = "WebForm1.aspx";
      }

      private Bitmap ГенерироватьРисунок()
      {  // Эта функция генерирует рисунок и возвращает его в формате Bitmap
         Color ЦветПера = Color.Black;
         Brush ЦветФона = Brushes.White, ЦветРисования = Brushes.Black;
         // Генератор псевдослучайных чисел:
         Random Случай = new Random();
         // Выбор цвета случайным образом из двух вариантов:
         ВыборЦвета(Случай, ref ЦветПера, ref ЦветФона, ref ЦветРисования);
         // Размеры изображения в точках:
         int Width = 136; int Height = 40;
         // Создаем точечный рисунок:
         Bitmap РАСТР = new Bitmap(Width, Height,
                                   PixelFormat.Format32bppArgb);
```

```
    // Создаем экземпляр класса Graphics из точечного рисунка:
    Graphics Графика = Graphics.FromImage(РАСТР);
    // Рисуем прямоугольник, заполненный цветом фона:
    Графика.FillRectangle(ЦветФона, 0, 0, Width, Height);
    // Искажаем рисунок сеткой линий и случайно разбросанными точками:
    Сетка_и_точки(Графика, ЦветПера, ЦветРисования, Случай,
                  Width, Height);
    // В прямоугольнике рисуем шесть случайно выбранных символов:
    РисованиеСимволов(Графика, ЦветРисования);
    Графика.Dispose();
    return РАСТР;
}

private void ВыборЦвета(Random Случай, ref Color ЦветПера,
                        ref Brush ЦветФона, ref Brush ЦветРисования)
{  // Выбираем случайным образом "комплект" цветов из двух вариантов
   // и передаем их как параметры по ссылке reference
   int R = Случай.Next(100);
   // Определяем, является ли случайное число R четным или нечетным:
   if (Четное(R) == true)
   {  // Первый вариант:
      ЦветПера = Color.Black;
      ЦветФона = Brushes.White;
      ЦветРисования = Brushes.Black;
   }
   else // Если R - нечетно:
   {  // Второй вариант:
      ЦветПера = Color.White;
      ЦветФона = Brushes.Black;
      ЦветРисования = Brushes.White;
   }
}

private bool Четное(int K)
{  // Функция определяет, число K является четным или нечетным:
   if (K % 2 == 0) return true;
   // Оператор % возвращает целочисленный остаток от деления
   return false;
}
private void Сетка_и_точки(Graphics Графика, Color ЦветПера,
                           Brush ЦветРисования, Random Случай,
                           int Width, int Height)
{  // Искажаем рисунок сеткой линий и случайно разбросанными точками.
   // Создание пера для рисования им фигур:
```

```
   Pen Перо = new Pen(ЦветПера, 1F);
   // Рисование сетки линий на рисунке:
   for (int x = 0; x <= Height; x += 15)
      Графика.DrawLine(Перо, 0, x, Width, x);
   for (int x = 0; x <= Width; x += 15)
      Графика.DrawLine(Перо, x, 0, x, Height);
   // Рисование случайно расположенных точек, их плотность равна
   // одной точке на 10 квадратных пикселов:
   int Кол_во_точек = (int)Math.Round((decimal)(Width * Height / 10), 0);
   for (int i = 0; i < Кол_во_точек; i++)
   {
      int x1 = Случай.Next(Width);
      int y1 = Случай.Next(Height);
      // Ширина эллипса - 1 пиксел, а его высота - 2 пиксела:
      Графика.FillEllipse(ЦветРисования, x1, y1, 1, 2);
   }
}

private void РисованиеСимволов(Graphics Графика, Brush Цвет_Рис)
{  // В прямоугольнике рисуем шесть случайно выбранных символов.
   // Задаем строку из шести случайно выбранных символов:
   string СтрокаШести = СлучайнаяСтрока();
   // Запоминаем эту строку на время текущей сессии:
   Page.Session.Add("Символы", СтрокаШести);
   // Копируем символы из строки в массив символов:
   char[] МассивШести = СтрокаШести.ToCharArray();
   // Задаем два шрифта: один обычный, а другой - наклонный (Italic):
   Font Шрифт1 = new Font("Arial", 20);
   // 20 - это размер самой широкой буквы
   Font Шрифт2 = new Font("Arial", 20, FontStyle.Italic);
   // Размер массива равен шести:
   int len = СтрокаШести.Length;
   // Цикл по каждой букве (их шесть):
   for (int i = 0; i < len; i++)
   {  // Point - координаты левого верхнего угла прямоугольника:
      Rectangle Прямоугольник = new Rectangle
                     (new Point(i * 22, 2), new Size(24, 36));
      // Размер прямямоугольника - Size(24 x 36)
      string ОдинСимвол = МассивШести[i].ToString();
      // Рисую один символ из тех самых шести символов:
      if (Четное(i) == true)
         Графика.DrawString(ОдинСимвол, Шрифт1, Цвет_Рис, Прямоугольник);
```

```
            else
               Графика.DrawString(ОдинСимвол, Шрифт2, Цвет_Рис, Прямоугольник);
         }
         Шрифт1.Dispose(); Шрифт2.Dispose();
      }

      private string СлучайнаяСтрока()
      {  // Эта функция возвращает строку с шестью случайно выбранными
         // символами из набора "СтрокаСимволов":
         string СтрокаСимволов = "23456789ERYUPADFHKZXVBNM";
         // В этом наборе нет ни нуля, ни буквы "O"("Q"), ни единицы, ни
         // буквы "I", чтобы их не путать
         char[] МассивСимволов = СтрокаСимволов.ToCharArray();
         // Генератор псевдослучайных чисел:
         Random Случай = new Random();
         // SS - строка, сформированная случайным образом:
         string SS = string.Empty;
         // Цикл от 0 до 5 заполняет строку SS шестью случайно выбранными
         // символами из массива "МассивСимволов":
         for (int i = 0; i <= 5; i++)
            SS = SS + МассивСимволов[Случай.Next(МассивСимволов.Length)];
         return SS;
      }

      protected void Button1_Click(object sender, EventArgs e)
      {  // Вспоминаем строку из шести случайно выбранных символов,
         // записанную в текущую сессию:
         string СтрокаШести = (Page.Session["Символы"]).ToString();
         // Удаление всех пробелов и невидимых управляющих символов:
         TextBox1.Text = TextBox1.Text.Trim();
         if (TextBox1.Text == СтрокаШести)
         { Label1.Text = "Верно"; Label1.ForeColor = Color.Black; }
         else
         { Label1.Text = "Попробуйте еще раз"; Label1.ForeColor = Color.Red; }
      }
   }
}
```

Как видно из программного кода, в процедуре загрузки страницы `Page_Load` происходит вызов ключевой функции `ГенерироватьРисунок` только при первичной загрузке страницы, т. е. когда `IsPostBack = false`. Эта функция возвращает растровый рисунок в формате Bitmap, который мы сохраняем (`Save`) на сервере. Далее для элемента управления `Image1` указываем виртуальный путь к сохраненному рисунку.

Функция `ГенерироватьРисунок` создает объект класса `Graphics` из точечного рисунка формата Bitmap, с помощью функции `ВыборЦвета` выбирает случайным образом из двух вариантов цвет фона и цвет рисования. Затем рисует заполненный цветом фона прямоугольник, который мы искажаем сеткой линий и случайно разбросанными точками, и выводит в этот прямоугольник шесть случайно выбранных символов, которые подлежат распознаванию пользователем.

Как видно, программа имеет иерархическую структуру с большой глубиной вложенности. Заслуживает обсуждения функция `СлучайнаяСтрока`. В ней мы задаем набор символов в строковой переменной `СтрокаСимволов`. В этом наборе отсутствуют цифра "1", буквы "I", "J", "T", поскольку их начертание похоже, а это может затруднять пользователю их распознавание. По этой же причине в наборе отсутствуют цифра "0", буквы "O", "Q", "G", "C". Данную строку символов с помощью функции `ToCharArray` преобразуем в массив символов. Пользуясь этим массивом, в цикле легко заполнить строку шестью случайно выбранными символами.

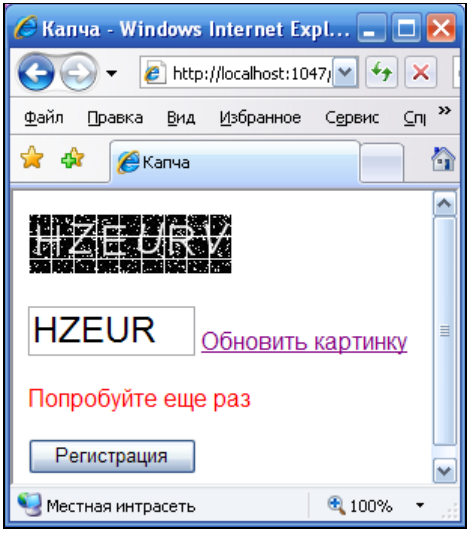


Рис. 14.16. Тестирование с помощью капча

Теперь нам требуется запомнить строку с шестью выбранными символами, чтобы ее сравнить со строкой тестового поля `TextBox1`, которое заполнит пользователь. Воспользоваться внешними переменными мы не можем, поскольку они стираются из оперативной памяти при перезагрузке страницы. Для этой цели воспользуемся объектом `Page.Session`. В технологии ASP.NET каждое Web-приложение при обращении к нему пользователя (клиента) создает объект `Page.Session`, этот объект называют *сессией* или *сеансом пользователя*. Объект `Session` содержит в себе методы для создания коллекции данных. Причем созданная коллекция данных на одной странице Web-приложения оказывается доступной на любой другой странице данного приложения. Даже если пользователь перешел на другое Web-приложение (на другой Web-узел), а затем вернулся в текущее приложение, то исходная коллекция данных сессии будет сохранена. Она будет сохранена до тех пор, пока пользователь не закрыл свой Web-браузер. Однако время жизни (лимит времени) сессии все же ограничено параметром `Timeout`. По умолчанию время жизни равно 20 минутам, его можно регули-

ровать в файле программной поддержки свойством объекта `Session.Timeout` или прямо в тексте программы. Таким образом, в процедуре `РисованиеСимволов` запоминаем строку с шестью выбранными символами, используя объект `Page.Session`.

При щелчке пользователем на кнопке **Регтистрация** читаем строку с шестью выбранными символами из объекта `Page.Session`. Теперь мы можем сравнить ее со строкой символов, полученной от пользователя в текстовом поле `TextBox1`.

Фрагмент работы капча представлен на рис. 14.16.

Убедиться в работоспособности данной программы можно, открыв соответствующее решение в папке Капча.

Пример 109. Отображение времени в Web-форме с использованием технологии AJAX

Мы уже обсуждали вопрос отображения времени в Windows-приложении (см. пример 86 в *главе 12*). Тогда для того чтобы время обновлялось каждую секунду, нами был использован элемент управления **Timer**. При обработке события `Tick` (события, когда прошел заданный интервал времени `Interval`, равный 1000 миллисекунд, т. е. 1 с) элемента управления **Timer** в метку **Label** копировалось новое значение времени, при этом изображение формы обновлялось (перерисовывалось). Когда речь идет о Web-форме, технология ASP.NET предполагает операции обратной отправки данных. Когда пользователь щелкает на кнопке или делает выбор в окне списка с включенной функцией `AutoPostBack`, серверу отсылается соответствующий запрос, после чего с него обратно клиенту передается целая страница. Если такое обращение к серверу будет происходить каждую секунду, то возрастет объем сетевого трафика и вследствие инерции Сети пользователь будет ощущать дискомфорт. Кроме того, если на этой странице предусмотрено заполнение пользователем полей, то серверу будет отсылаться страница с частично заполненными полями.

Решение этой проблемы состоит в использовании технологии AJAX, которая обеспечивает возможность выполнения частичного визуального обновления страницы посредством поддерживаемой ASP.NET AJAX-операции обратной отправки. Продемонстрируем возможность технологии AJAX на примере, когда мы имеем метку **Label** в Web-форме. На эту метку каждую секунду копируем новое время, но обновляем при этом не всю форму, а только метку с помощью технологии AJAX.

Решая эту задачу, запустим Visual Studio 2010 и закажем новый проект из шаблона **Empty ASP.NET Web Application**, укажем имя **Name** — AjaxTimer. К текущему проекту добавим Web-форму. Для этого в пункте меню **Project** выберем команду **Add New Item** и в появившемся окне дважды щелкнем по шаблону **Web Form**.

Далее, попав на вкладку конструктора Web-формы, перетащим в форму из раздела **AJAX Extensions** панели **ToolBox** элементы управления **ScriptManager**, **UpdatePanel**. Затем на элемент **UpdatePanel** поместим **Timer** и метку **Label**. Теперь

на вкладке файла программной поддержки **WebForm1.aspx.cs** напишем программный код, приведенный в листинге 14.11.

Листинг 14.11. Отображение времени в Web-форме

```
// Web-страница демонстрирует время на текстовой метке Label1. На эту метку
// каждую секунду копируем новое время, но обновляем при этом не всю форму,
// а только метку с помощью технологии AJAX
using System;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace AjaxTimer
{
   public partial class WebForm1 : System.Web.UI.Page
   {
      protected void Page_Load(object sender, EventArgs e)
      {
         Timer1.Interval = 1000; // 1000 миллисекунд = 1 секунда
         Label1.Text = "Текущее время: " + DateTime.Now.ToLongTimeString();
      }
      protected void Timer1_Tick(object sender, EventArgs e)
      {  // Обновление содержимого метки каждую секунду:
         Label1.Text = "Текущее время: " + DateTime.Now.ToLongTimeString();
      }
   }
}
```

Как видно из программного кода, при загрузке страницы задаем интервал времени, равный одной секунде. При обработке события `Tick` — события, когда прошел заданный интервал времени, в метку `Label1` копируется новое значение времени. При этом благодаря использованию технологии AJAX обновляется не вся Web-форма, а только те элементы, которые расположены на элементе **UpdatePanel**.

Фрагмент работы программы показан на рис. 14.17.

Убедиться в работоспособности Web-страницы можно, открыв решение AjaxTimer.sln в папке AjaxTimer.

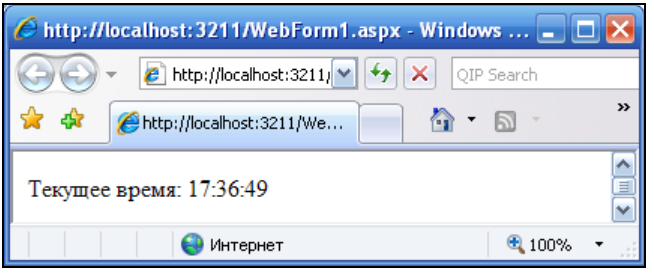


Рис. 14.17. В форме обновляется только AJAX-элемент **UpdatePanel**

Глава 15

Создание Web-служб и их клиентов

О Web-службах

Web-служба (от англ. *Web service*) — это программная система (более конкретно — откомпилированная библиотека динамической компоновки, т. е. файл формата DLL в папке bin приложения), расположенная на удаленном сервере (компьютере), к которой можно обращаться (потреблять сервис Web-службы) из своего клиентского компьютера. При этом возможности удаленной Web-службы будут реализованы в вашем Windows- или Web-приложении, а пользователь при достаточной скорости трафика может даже не заметить обращения вашей программы к удаленному серверу. На удаленном компьютере могут находиться и программный код, и данные. Для передачи по Интернету вызовов методов Web-служб и результатов их выполнения используется протокол SOAP (Simple Object Access Protocol). Протокол SOAP применяет формат сообщений, основанный на XML. Web-служба и протокол SOAP не зависят от конкретной платформы. Поэтому различные разработчики могут использовать Web-службы друг друга, не беспокоясь о совместимости операционных систем, технологий или языков программирования. Другое название Web-службы — Web-сервис.

Web-службы решают очень разные задачи. Это прогноз погоды, гороскоп, переводчик слов на различные иностранные языки, курсы валют, котировки ценных бумаг, проверка на корректность введенного пользователем почтового адреса, реестр улиц города, статистика правонарушений, информация о наличии свободных мест на авиарейс или в кинотеатр и т. д. Причем любой из подобных сервисов вы можете легко встроить в свое Windows- или Web-приложение путем создания клиентского приложения, потребляющего такой уже существующий сервис Web-службы.

Для поиска Web-служб в Интернете существует несколько сайтов, в которых Web-службы регистрируются их провайдерами для того, чтобы их могли найти заинтересованные в них разработчики. Таким сайтом является, например: **http://uddi.xml.org/uddi-org**. Регистрация в каталоге UDDI бесплатна. Здесь сегодня можно найти несколько десятков более или менее полезных Web-служб. В Сети можно найти и другие списки общедоступных Web-служб различных производителей, например: **http://www.xmethods.org/ve2/index.po**. Список русскоязычных Web-служб можно найти по адресу: **http://ivbeg.bestpersons.ru/feed/post3279396/**.

Упрощенно говоря, Web-служба в среде .NET состоит из двух компонентов: ASMX-файла и файла программной поддержки ASMX.cs. ASMX-файл содержит в себе информацию о методах Web-службы, о способах их тестирования, это можно посмотреть в любом Web-браузере. Файл программной поддержки написан на языке Visual C#, может быть скомпилирован для получения библиотеки DLL. Именно файл с расширением dll обеспечивает реализацию методов Web-службы. Например, на рис. 15.1 показано отображение в браузере Internet Explorer методов Web-службы, представленной в Сети по адресу:

http://webservice.webserviceshare.com/currencyconverter/rates.asmx.

Рис. 15.1. Пять методов Web-сервиса по операциям с курсами валют

Здесь перечислены пять методов Web-службы по операциям с текущими курсами валют. В создаваемом разработчиком клиентском приложении необходимо сделать ссылку на данный Web-сервер и таким образом включить в текущий проект удаленный класс. А затем в своем программном коде объявить создание нового экземпляра этого класса, и теперь можно уже обращаться ко всем методам класса.

Пример 110. Клиентское Web-приложение, потребляющее сервис Web-службы "Прогноз погоды"

В данном разделе создадим клиентское Web-приложение, потребляющее сервис Web-службы сайта **http://www.webservicex.net/globalweather.asmx** и продемонстрируем тем самым, что это — просто! Данная Web-служба возвращает прогноз погоды в городе, который пользователь задает в запросе к службе. Эта Web-

служба поддерживает два метода (функции): `GetCitiesByCountry` и `GetWeather` (рис. 15.2). На вход первой функции `GetCitiesByCountry` подают название страны, где хотят получить прогноз погоды, а на выходе функции получают перечисление городов этой страны, для которых Web-служба готова сделать прогноз погоды.

Рис. 15.2. Два метода Web-службы прогноза погоды

На вход второй функции `GetWeather` подают названия города и страны, а на выходе функции получают XML-строку, содержащую прогнозируемые параметры погоды (листинг 15.1).

Листинг 15.1. XML-код, полученный из функции `GetWeather`

```
<?xml version="1.0" encoding="utf-16"?>
<CurrentWeather>
  <Location>Moscow / Vnukovo , Russia (UUWW) 55-39N 037-16E</Location>
  <Time>Dec 09, 2010 - 03:30 AM EST / 2010.12.09 0830 UTC</Time>
  <Wind> from the E (090 degrees) at 11 MPH (10 KT):0</Wind>
  <Visibility> 1 mile(s):0</Visibility>
  <SkyConditions> overcast</SkyConditions>
  <Temperature> 23 F (-5 C)</Temperature>
  <Wind>Windchill: 12 F (-11 C):1</Wind>
  <DewPoint> 23 F (-5 C)</DewPoint>
  <RelativeHumidity> 100%</RelativeHumidity>
  <Pressure> 29.68 in. Hg (1005 hPa)</Pressure>
  <Status>Success</Status>
</CurrentWeather>
```

Как видно из листинга, параметры погоды указываются в XML-элементах: температура (`Temperature`), относительная влажность (`RelativeHumidity`), давление (`Pressure`) и проч. в Москве 9 декабря 2010 года на 8:30 по Гринвичу.

Напишем Web-приложение (также легко можно написать и Windows-приложение), обращающееся к функции `GetWeather` данной удаленной Web-службы. Причем программный код должен быть минимальным и демонстрировать лишь принцип подключения и доступ к нужному элементу (например, температуре) XML-документа.

Для этой цели после запуска Visual Studio 2010 выберем проект шаблона **Empty ASP.NET Web Application**, укажем имя **Name** — WebКлиентПогода. К текущему проекту добавим Web-форму. Для этого в пункте меню **Project** выберем команду **Add New Item** и в появившемся окне дважды щелкнем на шаблоне **Web Form**. Далее, попав в конструктор Web-формы, из панели элементов **Toolbox** перетащим кнопку **Button**, текстовое поле **TextBox** для вывода строки с данными XML и метку **Label** для вывода на нее значения температуры.

Теперь наступает ключевой момент, а именно — подключение ссылки на удаленную Web-службу, содержащую класс с необходимыми методами. Для этого в пункте меню **Project** выберем команду **Add Web Reference**. Этого пункта меню может не быть в меню **Project**, в этом случае выбираем **Add Service Reference | Advanced | Add Web Reference**. Также эту команду можно выбрать в контекстном меню окна **Solution Explorer**. В результате получим интерфейс, показанный на рис. 15.3. Здесь в поле **URL** введем адрес Web-службы: **http://www.webservicex.net/globalweather.asmx** и щелкнем на кнопке **Go**.

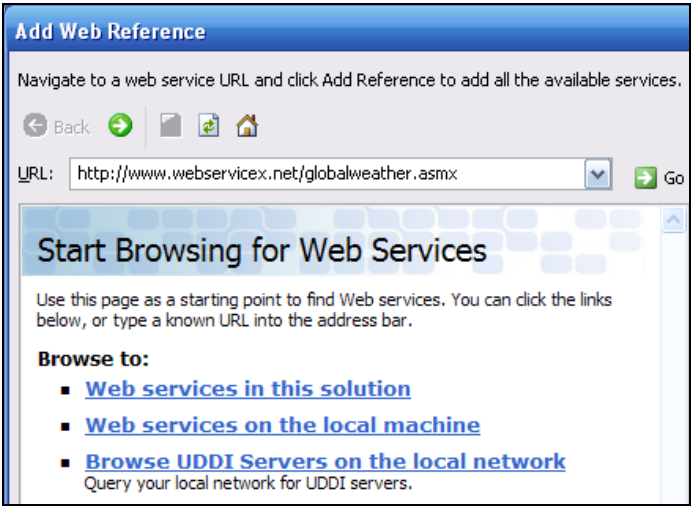


Рис. 15.3. Поиск нужной Web-службы в Сети

Теперь в окне под полем **URL** мы увидим оба метода данного сервиса в том виде, как это было на рис. 15.2 (после нажатия на кнопку **Go**). При этом станет доступной кнопка **Add Reference**. Щелкнем на этой кнопке, после этого в окне **Solution Explorer** появится новый узел **Web References** со ссылкой на удаленный класс **net.webservicex.www** (рис. 15.4).

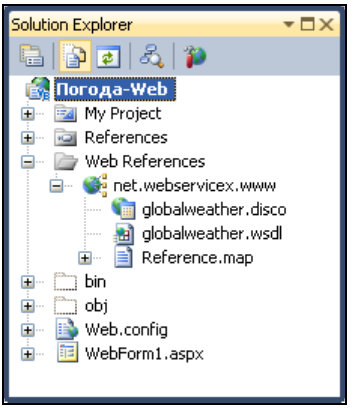


Рис. 15.4. В окне **Solution Explorer** появилась ссылка на удаленный класс

Теперь этот класс, содержащий необходимую нам удаленную Web-службу, мы можем использовать в своем программном коде (листинг 15.2).

Листинг 15.2. Web-приложение, обращающееся к сервису удаленной Web-службы прогноза погоды

```
// Web-приложение, потребляющее сервис удаленной Web-службы прогноза погоды.
// Приложение в текстовом поле TextBox демонстрирует XML-строку с параметрами
// погоды для города, указанного во входных параметрах при обращении
// к Web-службе. Также выводит в текстовую метку значение температуры
// в этом городе
using System;
using System.Web.UI.WebControls;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace Погода_Web
{
   public partial class WebForm1 : System.Web.UI.Page
   {
      protected void Page_Load(object sender, EventArgs e)
      {
         Button1.Text = "Выяснить погоду"; Label1.Text = string.Empty;
         TextBox1.TextMode = TextBoxMode.MultiLine;
         Button1.Focus();
      }

      protected void Button1_Click(object sender, EventArgs e)
      {
         // Создаем клиентское приложение Web-службы:
```

```
        // http://www.webservicex.net/globalweather.asmx.
        // Эта Web-служба часто бывает перегужена и поэтому может выдать
        // сообщение: "Server is too busy".
        // Создание экземпляра прокси-класса:
        var ПОГОДА = new net.webservicex.www.GlobalWeather();
        // Эти три строчки - для отладки, чтобы лишний раз
        // не "дергать" сервер:
        // var Reader = new System.IO.StreamReader("C:\\Погода.xml");
        // string Строка_XML = Reader.ReadToEnd();
        // Reader.Close();
        // Функция GetWeather запрашивает строковые параметры с названием
        // города и страны и возвращает строку с XML-документом:
        string Строка_XML = ПОГОДА.GetWeather("Moscow", "Russia");
        // Какая погода в Киеве:
        // string Строка_XML = ПОГОДА.GetWeather("Kyiv", "Ukraine");
        TextBox1.Text = Строка_XML;
        var Документ = new System.Xml.XmlDocument();
        // Загрузка строки XML в XML-документ
        Документ.LoadXml(Строка_XML);
        var Читатель = new System.Xml.XmlNodeReader(Документ);
        string Имя = string.Empty; string Значение = string.Empty;
        while (Читатель.Read() == true)
        {  // Читаем последовательно каждый узел, выясняя тип узла:
           if (Читатель.NodeType == System.Xml.XmlNodeType.Element)
                                          Имя = Читатель.Name;
           // Каждый раз запоминаем имя узла
           if (Читатель.NodeType != System.Xml.XmlNodeType.Text) continue;
           if (Имя == "Temperature") { Значение = Читатель.Value; break; }
        } // Выход из цикла, когда прочитали данные узла "Temperature"
        Label1.Text = "Температура воздуха в Москве: " + Значение;
     }
  }
}
```

Как видно, в программном коде при обработке события "щелчок на кнопке" **Выяснить погоду** создается экземпляр класса удаленной Web-службы. Далее происходит непосредственное обращение к методу класса `GetWeather` с входными параметрами `город` и `страна`, и метод `GetWeather` возвращает строку с XML-документом. Данную строку выводим в текстовое поле `TextBox1`. Для "расшифровки" XML-строки загружаем эту строку в XML-документ. Значение температуры находим в содержимом элемента `Temperature`, затем выводим значение температуры на метку `Label1`.

Фрагмент работы программы показан на рис. 15.5.

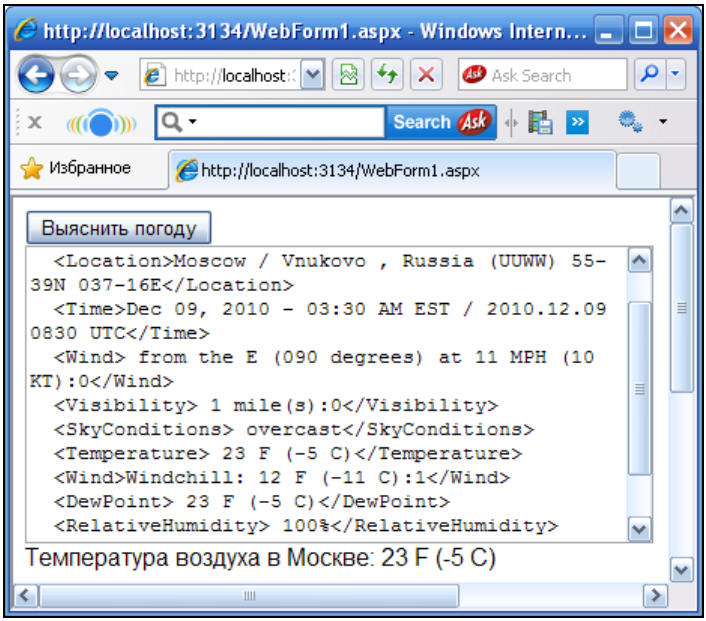


Рис. 15.5. Работа клиента Web-службы "Прогноз погоды"

Можно было бы вывести содержимое XML-документа на сетку данных **GridView**, тогда мы могли бы увидеть все параметры погоды в удобной таблице. Мы не стали этого делать, чтобы не загромождать программу и тем самым не "затушевать" главную идею, а именно обращение к удаленному классу, предоставленному Web-службой.

Таким образом, мы продемонстрировали создание клиентского Web-приложения, потребляющего сервис Web-службы. Замечу, что аналогичным образом можно создавать и Windows-приложения (настольные приложения), которые также, будучи клиентами какой-либо Web-службы, могут получать, например, справочную информацию в онлайновом режиме.

Убедиться в работоспособности данной программы можно, открыв соответствующий файл решения в папке WebКлиентПогода.

Пример 111. Клиентское Windows-приложение, использующее Web-службу "Прогноз погоды"

Теперь создадим клиентское Windows-приложение, потребляющее сервис той же Web-службы "Прогноз погоды", и продемонстрируем тем самым, что обращаться к удаленному классу можно не только из Web-приложения, но и из Windows-приложения (т. е. из настольного приложения).

Для решения этой задачи запустим Visual Studio 2010, выберем шаблон **Windows Forms Application C#**. В поле **Name** зададим имя WindowsКлиентПогода. В конструкторе формы добавим кнопку **Button**, текстовое поле **TextBox** и метку **Label**.

Следующим этапом разработки клиентского приложения является добавление ссылки на удаленный класс искомой Web-службы. Для этого выберем в пункте меню **Project** команду **Add Service Reference**, это приведет к появлению диалогового окна **Add Service Reference**. В этом окне щелкнем кнопку **Advanced**, а затем кнопку **Add Web Reference**. В появившемся диалоговом окне Add Web Reference в поле URL введем адрес Web-службы: **http://www.webservicex.net/globalweather.asmx** и нажмем кнопку **Go**. В окне ниже увидим список доступных методов данного сервиса, после этого щелкнем на кнопке **Add Reference**. При этом в окне **Solution Explorer** появится ссылка на удаленный класс: **net.webservicex.www**. Теперь мы можем использовать эту ссылку в программном коде (листинг 15.3).

Листинг 15.3. Windows-приложение, использующее сервис удаленной Web-службы прогноза погоды

```
// Windows-приложение, потребляющее сервис удаленной Web-службы прогноза
// погоды. Приложение в текстовом поле TextBox демонстрирует XML-строку
// с параметрами погоды для города, указанного во входных параметрах
// при обращении к Web-службе.
// Также выводит в текстовую метку значение температуры в этом городе
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace WebКлиентПогода
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
         // Добавляем внешнюю ссылку на Web-службу таким образом:
         // Проект - Добавить ссылку на службу - Дополнительно -
         // Добавить веб-ссылку http://www.webservicex.net/globalweather.asmx
         button1.Text = "Выяснить погоду"; label1.Text = string.Empty;
         textBox1.Multiline = true;
         button1.Focus();
      }
```

```
      private void button1_Click(object sender, EventArgs e)
      {  // Создание экземпляра прокси-класса:
         var ПОГОДА = new net.webservicex.www.GlobalWeather();
         // Эти три строчки - для отладки, чтобы лишний раз
         // не "дергать" сервер:
         //var Reader = new System.IO.StreamReader("C:\\Погода.xml");
         //string Строка_XML = Reader.ReadToEnd();
         //Reader.Close();
         // Функция GetWeather запрашивает строковые параметры с названием
         // города и страны и возвращает строку с XML-документом:
         string Строка_XML = ПОГОДА.GetWeather("Moscow", "Russia");
         // Какая погода в Киеве:
         // string Строка_XML = ПОГОДА.GetWeather("Kyiv", "Ukraine");
         textBox1.Text = Строка_XML;
         // Здесь считывание значения узла из XML-строки выполено более
         // эффективно:
         var XML_элемент = System.Xml.Linq.XElement.Parse(Строка_XML);
         string Значение = XML_элемент.Element("Temperature").Value;
         label1.Text = "Температура воздуха в Москве: " + Значение;
      }
   }
}
```

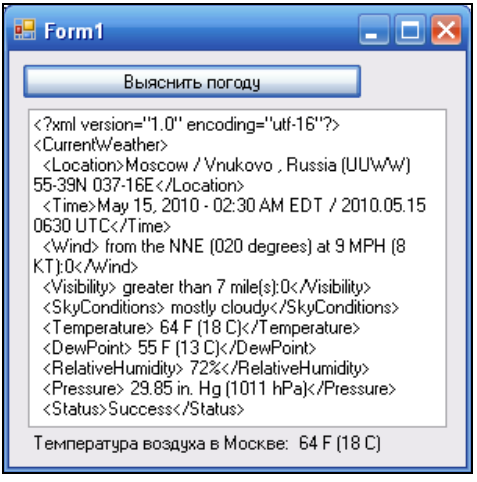


Рис. 15.6. Работа Windows-клиента Web-службы "Прогноз погоды"

В программном коде при обработке события "щелчок на кнопке" `button1` создаем экземпляр удаленного класса `ПОГОДА`, а затем обращаемся к методу класса `GetWeather`, подавая на вход метода страну и город (`Russia` и `Moscow`), для которого мы хотим вяснить прогноз погоды. Этот метод возвращает строку XML. Эту XML-

строку мы выводим в текстовое поле `textBox1`, так же, как мы это делали в Web-приложении, однако извлекаем из узла `Temperature` прогнозируемое значение температуры более эффективным образом, используя методы технологии Linq.

Фрагмент работы программы представлен на рис. 15.6.

Убедиться в работоспособности программы можно, открыв решение в папке WindowsКлиентПогода.

Пример 112. Создание простейшей Web-службы

А теперь, когда мы убедились, что создать клиентское приложение, потребляющее сервис Web-службы, не сложно, поставим задачу создать самую простую Web-службу, чтобы убедиться, что это тоже простая задача. Автор понимает, что сейчас наступил очень деликатный момент: если сейчас будет приведен какой-либо сложный пример, то можно навсегда отбить интерес у читателя разобраться с принципами работы Web-служб. Поэтому задача, решаемая данной Web-службой, будет максимально простой. Например, пользователю предлагается ввести два числа, а Web-служба берет на себя функцию сложения этих двух чисел и вывода (возврата) суммы. При этом необходимо произвести диагностику вводимых данных. От такой Web-службы нет особенной пользы, но на этой маленькой задачке нам будет удобно продемонстрировать все возможности. Здесь, так же как и при отладке активных Web-страниц из предыдущей главы, при отладке данной Web-службы в качестве удаленного серверного компьютера и клиентского компьютера мы будем использовать один ваш локальный компьютер.

Для решения этой задачи запустим Visual Studio 2010, выберем шаблон **Empty ASP.NET Web Application**, зададим имя, например, WebСлужбаСумма. Затем в меню **Project** выберем команду **Add New Item**, в появившемся окне укажем шаблон **Web Service** и щелкнем на кнопке **Add**. Сразу после этого попадаем на вкладку **WebService1.asmx.cs** готовой Web-службы "Hello World", и уже ее можно тестировать. Содержимое этой вкладки приведено в листинге 15.4.

Листинг 15.4. Файл программной поддержки Web-службы "Hello World"

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Services;
namespace WebСлужбаСумма
{
   /// <summary>
```

```
    /// Summary description for WebService1
    /// </summary>
    [WebService(Namespace = "http://tempuri.org/")]
    [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
    [System.ComponentModel.ToolboxItem(false)]
    // To allow this Web Service to be called from script, using ASP.NET AJAX,
    // uncomment the following line.
    // [System.Web.Script.Services.ScriptService]
    public class WebService1 : System.Web.Services.WebService
    {
        [WebMethod]
        public string HelloWorld()
        {
            return "Hello World";
        }
    }
}
```

Однако "HelloWorld — это другая задача, разработчики Visual Studio 2010 включили ее для облегчения програамистам освоения данной технологии. Нас интересует именно наша задача, поэтому изменим файл программной поддержки, как показано в листинге 15.5.

Листинг 15.5. Простейшая Web-служба

```
// На входе данной Web-службы предлагается ввести два числа, а Web-служба
// берет на себя функцию сложения этих двух чисел и вывода (возврата) суммы.
// При этом Web-служба производит диагностику вводимых данных
using System;
using System.Web.Services;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace WebApplication1
{
    /// <summary>
    /// Summary description for WebService1
    /// </summary>
    [WebService(Namespace = "http://tempuri.org/")]
    [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
    [System.ComponentModel.ToolboxItem(false)]
    // To allow this Web Service to be called from script, using ASP.NET AJAX,
    // uncomment the following line.
    // [System.Web.Script.Services.ScriptService]
```

```
   public class WebService1 : System.Web.Services.WebService
   {
      [WebMethod]
      public string Сумма(string Число1, string Число2)
      {  // Входные параметры объявляем типа string, чтобы принимать
         // от пользователя любые символы, анализировать их и при
         // "плохом вводе" сообщать по-русски.
         Single X, Y;
         bool Число_ли = Single.TryParse(
                 Число1, System.Globalization.NumberStyles.Number,
                 System.Globalization.NumberFormatInfo.CurrentInfo, out X);
         if (Число_ли == false) return "В первом поле должно быть число";

         Число_ли = Single.TryParse(
                 Число2, System.Globalization.NumberStyles.Number,
                 System.Globalization.NumberFormatInfo.CurrentInfo, out Y);
         if (Число_ли == false) return "Во втором поле должно быть число";

         Single Z = X + Y;
         return "Сумма = " + Z.ToString();
      }
   }
}
```

Как видно из текста программы, мы не программировали никакой пользовательский интерфейс. У нас здесь нет ни формы, ни кнопок, ни текстовых полей. Уже отсюда понятно, что Web-служба — это удаленный класс, к которому можно подключиться через Интернет из своего, например, Windows- или Web-приложения. На вход функции `Сумма` пользователь Web-службы подает два числа, которые принимаются функцией как две строки. Функция `Single.TryParse` проверяет, числовые ли символы содержатся во введенных строках, и если это так, то происходит преобразование строковых переменных в переменные типа `Single`.

Теперь можем протестировать данную Web-службу, для этого нажмем клавишу <F5>. При этом Web-браузер откроет ASMX-файл. Здесь мы увидим единственный метод этой Web-службы — `Сумма`. Щелчок указателем мыши на этой гиперссылке обеспечит возможность тестирования нашей Web-службы (рис. 15.7).

Мы можем вводить в данные поля нечисловые символы, числа с десятичной точкой или запятой и тем самым тестировать поведение нашей программы. Заметьте, что мы не программировали эти поля, система сама предоставила нам такую возможность для тестирования программируемой нами Web-службы.

Убедиться в работоспособности данной Web-службы можно, открыв в папке WebСлужбаСумма соответствующий файл решения.

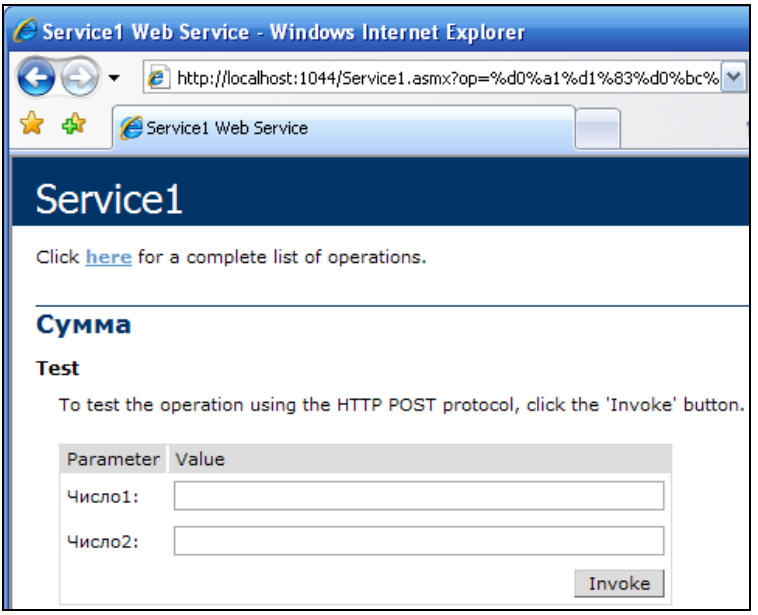


Рис. 15.7. Тестирование созданной Web-службы

Пример 113. Создание Windows-приложения — потребителя сервиса Web-службы

Теперь напишем Windows-приложение (т. е. создадим exe-файл), который будет обращаться к Web-службе, написанной в предыдущем разделе. Такое приложение можно называть *клиентским*. Для этого запустим Visual Studio 2010, выберем шаблон **Windows Forms Application C#** (можно даже **Console Application**), укажем имя проекта WebКлиентСумма. Из панели **Toolbox** перенесем в форму командную кнопку **Button**. Запланируем, что вся работа с удаленной Web-службой будет происходить при обработке события "щелчок на кнопке" **Пуск.**

Чтобы сделать Windows-приложение потребителем сервиса Web-службы, необходимо в его проекте создать Web-ссылку на удаленный класс. Для этого выберем в пункте меню **Project** (или в контекстном меню окна **Solution Explorer**) команду **Add Service Reference**, это приведет к появлению диалогового окна **Add Service Reference**. В этом окне нажмем кнопку **Advanced**, а затем в следующем окне кнопку **Add Web Reference.**

Поскольку мы отлаживаем и Web-службу, и его клиентское приложение на одном локальном компьютере, в поле **URL** пишем виртуальный адрес ASMX-файла. Этот адрес мы можем получить, запустив нашу Web-службу из предыдущего раз-

дела, а затем скопировав в буфер обмена его URL-адрес. Для локального компьютера, на котором автор отлаживал данный пример, URL-адрес был таким: **http://localhost:1611/Service1.asmx**.

Затем нажмем кнопку **Go** и тем самым получим доступ к кнопке **Add Reference**, после щелчка на которой в окне **Solution Explorer** добавится значок ссылки на класс `localhost`, обеспечивающий доступ к Web-службе (рис. 15.8).

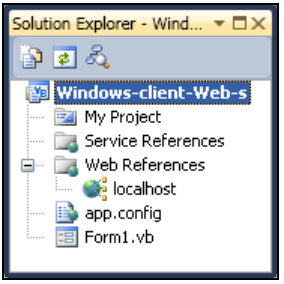


Рис. 15.8. Добавление ссылки на класс `localhost`

Если посмотреть на этот класс через браузер объектов **View in Object Browser** (используя контекстное меню в окне **Solution Explorer**), то можно увидеть все свойства, методы и события, которые содержатся в этом классе. Теперь, когда в наш проект добавлена ссылка на удаленный класс, мы можем вызывать методы этого класса в нашей программе (листинг 15.6).

Листинг 15.6. Windows-приложение, потребляющее сервис Web-службы

```
// Клиентское Windows-приложение, потребляющее сервис Web-службы
// предыдущего примера WebСлужбаСумма
using System;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace WebКлиентСумма
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
         button1.Text = "Пуск";
      }

      private void button1_Click(object sender, EventArgs e)
      {  // Чтобы добавить Web-службу к обычному Windows-приложению:
         // Project | Add Service Reference | Advanced | Add Web Reference,
```

```
            // затем в поле URL пишем виртуальный адрес Web-службы
            // http://localhost:1611/Service1.asmx.
            // Создаем экземпляр удаленного класса:
            var Удаленный = new localhost.WebService1();
            string Sum = Удаленный.Сумма("23,5", "11,4");
            MessageBox.Show(Sum);
        }
    }
}
```

Как видно из программного кода, при обработке события "щелчок мышью" на кнопке **Пуск** создаем экземпляр удаленного класса и обращаемся к его функции (методу) `Сумма`. При этом при вводе чисел специально одно из них пишем через десятичную точку, а другое — через запятую для тестирования возможности ввода чисел обоими способами.

Результат работы программы представлен на рис. 15.9.

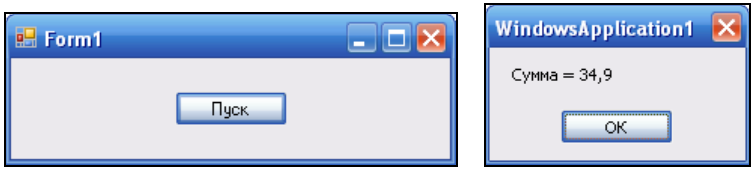


Рис. 15.9. Обращение Windows-приложения к Web-службе

Убедиться в работоспособности программы можно, открыв соответствующий sln-файл решения в папке WebКлиентСумма.

Пример 114. Web-служба "Торговая рекомендация на рынке Forex"

Создадим Web-службу, обеспечивающую торговой рекомендацией участников международного валютного рынка Forex. Как известно, любой желающий через Интернет, используя соответствующее программное обеспечение, может подключиться к этому рынку и совершить либо покупку какой-либо валютной пары (например, евро-доллар, EUR/USD), либо ее продажу. И в зависимости от правильно выбранного направления движения цены либо получить прибыль, либо убыток. Существует множество сайтов, где выкладываются рекомендуемые торговые стратегии работы на рынке Forex на текущую европейскую или американскую сессию. Например, мы доверяем какому-нибудь из таких сайтов, скажем, сайту **http://www.forex-rdc.ru/subscribers.php?action=prognoz**, и решаем руководствоваться его рекомендациями. Кроме того, мы создаем Web-службу, которая с помо-

щью синтаксического разбора указанной выше Web-страницы будет извлекать торговую стратегию, рекомендованную данным сайтом на текущий день. Поскольку в нашей книге мы должны привести очень краткий и выразительный пример, то ограничимся торговой рекомендацией только для одной валютной пары EUR/USD.

Для решения этой задачи запутим Visual Studio 2010, выберем шаблон **ASP.NET Empty Web Application C#**, зададим имя — WebСлужбаForex. Далее в пункте меню **Project** выберем пункт **Add New Item** и в появившемся окне дважды щелкнем на шаблоне **Web Service**. Мы получим простейшую Web-службу "Hello World". Однако мы ее изменим согласно условию нашей задачи. Содержимое файла программной поддержки приведено в листинге 15.7.

Листинг 15.7. Web-служба "Торговая рекомендация на рынке Forex"

```
// Web-служба, которая с помощью синтаксического разбора Web-страницы
// http://www.forex-rdc.ru/subscribers.php?action=prognoz
// извлекает торговую рекомендацию на рынке Forex для валютной пары EUR/USD,
// предлагаемую данным сайтом на текущий день, и выводит ее потребителю
// сервиса Web-службы в виде строки
using System;
using System.Web.Services;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace WebСлужбаForex
{
   /// <summary>
   /// Summary description for WebService1
   /// </summary>
   [WebService(Namespace = "http://tempuri.org/")]
   [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
   [System.ComponentModel.ToolboxItem(false)]
   // To allow this Web Service to be called from script, using ASP.NET AJAX,
   // uncomment the following line.
   // [System.Web.Script.Services.ScriptService]
   public class WebService1 : System.Web.Services.WebService
   {
      [WebMethod]
      public string Рекомендация()
      {  // Создаем объект для чтения Web-страницы:
         var КЛИЕНТ = new System.Net.WebClient();
         // Чтобы русские буквы читались корректно, объявляем объект Кодировка:
         var Кодировка = System.Text.Encoding.GetEncoding(1251);
         System.IO.Stream ПОТОК;
         string СТРОКА;
```

```
        try
        {  // Попытка открытия Web-ресурса:
           ПОТОК = КЛИЕНТ.OpenRead(
            "http://www.forex-rdc.ru/subscribers.php?action=prognoz");
        }
        catch (Exception Ситуация)
        {
           СТРОКА = string.Format(
                     "Ошибка открытия www.forex-rdc.ru\r\n{0}", Ситуация);
           return СТРОКА;
        }
        // Чтение HTML-разметки Web-страницы:
        var Читатель = new System.IO.StreamReader(ПОТОК, Кодировка);
        // Копируем HTML-разметку в строковую переменную:
        СТРОКА = Читатель.ReadToEnd();
        // Ищем в разметке страницы фрагмент с указанной строкой:
        int i = СТРОКА.IndexOf("Торговая стратегия:");
        // Ищем стратегию только для EUR/USD:
        СТРОКА = СТРОКА.Substring(i, 120);
        // Удаляем HTML-разметку:
        i = СТРОКА.IndexOf("</p>");
        СТРОКА = СТРОКА.Remove(i);
        СТРОКА = СТРОКА.Replace("</b>", "");
        // Вставляем текущую дату:
        СТРОКА = СТРОКА.Replace("стратегия:", "стратегия для EUR/USD на " +
                           DateTime.Now.ToLongDateString() + ":\r\n");
        ПОТОК.Close();
        return СТРОКА;
     }
  }
}
```

Как видно из программного кода, данная Web-служба содержит в себе один метод `Рекомендация()`, который не имеет входных параметров. В начале метода создаем экземпляр класса `WebClient` для чтения и последующего синтаксического разбора Web-страницы. Вообще говоря, Web-страница может быть записана на машинном носителе в какой-либо кодировке. Чаще всего это Unicode (UTF-8), русскоязычные сайты часто имеют кодировку Windows 1251. Чтобы выяснить, в какой кодировке записана Web-страница, следует в браузере в контекстном меню навести указатель мыши на пункт меню **Кодировка**.

При чтении Web-страницы создаем объект `ПОТОК`, используя URL-адрес страницы, и объект `Кодировка` для корректного отображения кириллицы. После копирования HTML-разметки страницы в строковую переменную `СТРОКА` ищем в раз-

метке страницы раздел с торговой стратегией для EUR/USD с помощью строковых операций.

Теперь запустим созданную Web-службу, нажав клавишу <F5>. Если формальных ошибок нет, мы получим отображение ASMX-файла в браузере (рис. 15.10).

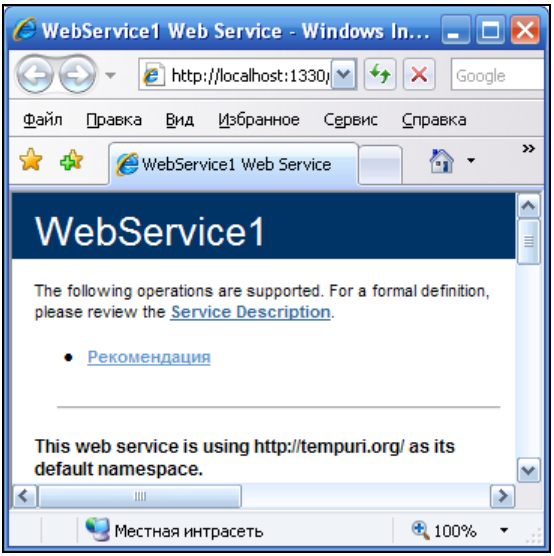


Рис. 15.10. Отображение ASMX-файла в браузере

Щелкнув в этом окне на изображении единственного метода `Рекомендация` данной Web-службы, мы попадаем на другую страницу, где получаем возможность его тестировать. Для этого щелкнем на кнопке **Invoke**. В результате тестирования получим в браузере результат работы данного метода Web-службы в формате XML-сообщения (рис. 15.11).

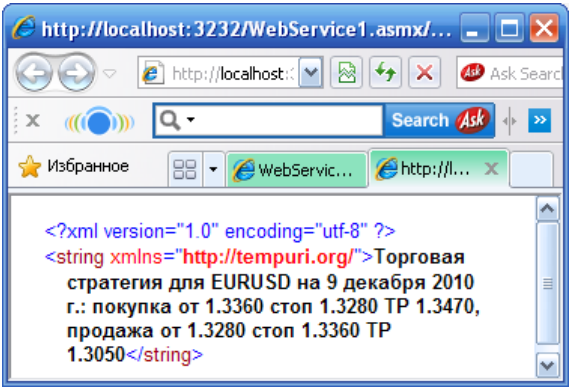


Рис. 15.11. XML-сообщение, как результат работы Web-сервиса

Убедиться в работоспособности программы можно, запустив соответствующее решение в папке WebСлужбаForex.

Пример 115. Клиентское приложение, потребляющее сервис Web-службы "Торговая рекомендация на рынке Forex"

Теперь напишем клиентское Windows-приложение, использующее Web-службу, созданную нами в предыдущем примере. Для этого запустим Visual Studio 2010, выберем шаблон **Windows Forms Application C#** и в поле **Name** зададим имя WebКлиентForex. Из панели элементов **Toolbox** перенесем в форму метку **Label**, куда будем помещать строку торговой рекомендации, полученную от Web-службы "Торговая стратегия на рынке Forex".

Далее необходимо в проекте создать Web-ссылку на удаленный класс Web-службы. Для этого выберем в пункте меню **Project** команду **Add Service Reference**, это приведет к появлению диалогового окна **Add Service Reference**. В этом окне нажмем кнопку **Advanced**, а затем в следующем окне кнопку **Add Web Reference**.

Теперь в поле **URL** пишем виртуальный адрес ASMX-файла используемой Web-службы. Этот адрес мы можем получить, запустив нашу Web-службу из предыдущего раздела, а затем скопировав в буфер обмена его URL-адрес. На вкладке программного кода наберем простейшую программу (листинг 15.6).

Листинг 15.8. Клиентское приложение, потребляющее сервис Web-службы "Торговая рекомендация на рынке Forex"

```
// Получаем прогноз рынка Forex на текущий день. Клиентское Windows-
приложение,
// потребляющее сервис Web-службы предыдущего примера WebСлужбаForex
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace WebКлиентForex
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
         // Создаем клиентское приложение Web-службы:
         // http://localhost:1330/WebService1.asmx
         // Создаем экземпляр удаленного класса:
         var Forex = new localhost.WebService1();
```

```
            this.Text = "Рынок Forex";
            label1.Text = Forex.Рекомендация();
        }
    }
}
```

Как видно из программного кода, при обработке события загрузки формы создаем экземпляр удаленного класса Web-службы, затем обращаемся к его методу `Рекомендация` и копируем получаемую строку в метку `label1`. Фрагмент работы программы приведен на рис. 15.12.

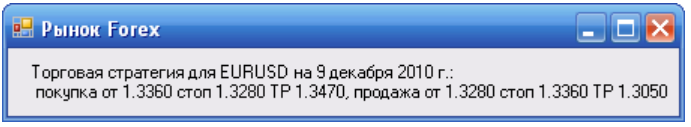


Рис. 15.12. Результат работы клиентского приложения

Убедиться в работоспособности программы можно, запустив соответствующее решение в папке WebКлиентForex.

Пример 116. Клиентское Web-приложение, потребляющее сервис Web-службы "Морфер"

Создадим клиентское Web-приложение не для Web-службы, написанной нами, а для удаленного Web-сервиса. Различных готовых Web-служб много, их можно посмотреть, протестировать на сайтах **http://uddi.xml.org/**, **http://www.xmethods.org/**, **http://www.webservicelist.com/** и др. Выберем одну из русскоязычных Web-служб, а именно Web-службу склонения существительных "Морфер", ее адрес в Сети следующий: **http://www.morpher.ru/WebServices/Morpher.asmx**. Наберем этот URL в адресной строке браузера и получим (рис. 15.13) операции (методы), поддерживаемые Web-службой Morpher.

Эта Web-служба предназначена для автоматизированной обработки текстов на русском языке. В частности метод `GetAll5` Web-службы (см. рис. 15.13) обеспечивает склонение слов и словосочетаний на русском языке. На вход функции (метода) `GetAll5` данной Web-службы подаем слово в именительном падеже, функция возвращает массив строк, в которых записано это слово в различных пяти падежах: родительном, дательном, винительном, творительном и предложном.

Для испытания этой Web-службы запустим Visual Studio 2010, выберем шаблон **Empty ASP.NET Web Application C#**. В поле **Name** укажем имя проекта WebКлиентMorpher. К текущему проекту добавим Web-форму. Для этого в пункте меню **Project** выберем команду **Add New Item** и в появившемся окне дважды щелк-

нем на шаблоне **Web Form**. В конструкторе формы из панели элементов **Toolbox** перенесем текстовое поле **TextBox** и кнопку **Button**. Далее, чтобы добавить ссылку на удаленный класс, выберем в пункте меню **Project** (или в контекстном меню окна **Solution Explorer**) команду **Add Service Reference**, это приведет к появлению одноименного диалогового окна. В этом окне нажмем кнопку **Advanced**, а в следующем окне кнопку **Add Web Reference**. Далее в поле **URL** введем ссылку на Web-службу: **http://www.morpher.ru/WebServices/Morpher.asmx**.

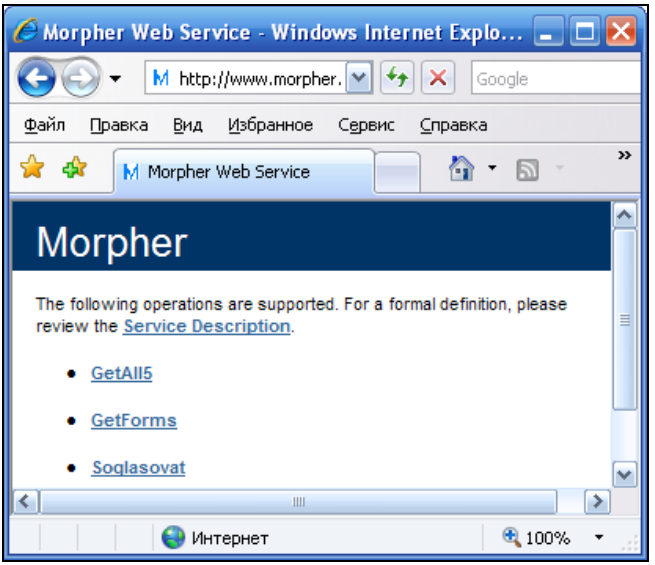


Рис. 15.13. Методы, поддерживаемые Web-службой Morpher

При этом кнопка **Add Reference** станет доступной, а после щелчка на ней в окне **Solution Explorer** появится ссылка на удаленный класс: **ru.morpher.www**. Теперь мы можем использовать эту ссылку в нашем программном коде (листинг 15.9).

Листинг 15.9. Клиентское Web-приложение, потребляющее сервис Web-службы "Морфер"

```
// Клиентское Web-приложение, потребляющее сервис Web-службы склонения
// существительных "Морфер". На вход метода Web-службы подаем слово на
// русском языке, на выходе получаем это слово в различных пяти падежах
using System;
using System.Web.UI.WebControls;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace WebКлиентMorpher
{
   public partial class WebForm1 : System.Web.UI.Page
```

```
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            Button1.Text = "Просклонять";
            TextBox1.TextMode = TextBoxMode.MultiLine;
        }
        protected void Button1_Click(object sender, EventArgs e)
        {  // Создаем клиентское приложение Web-службы:
            //         http://www.morpher.ru/WebServices/Morpher.asmx
            // Создаем экземпляр удаленного класса:
            var Склонение = new ru.morpher.www.Morpher();
            string[] Падежи = Склонение.GetAll5("Зиборов Виктор Владимирович");
            Падежи[4] = "О " + Падежи[4];
            // Перевод каретки vbCrLf в конце каждого склонения:
            foreach (string C in Падежи)
                TextBox1.Text = TextBox1.Text + C + "\r\n";
        }
    }
}
```

Как видно из текста программы, при обработке события "щелчок на кнопке" **Просклонять** создаем экземпляр удаленного класса `Склонение` и далее используем метод `GetAll5` класса для получения возможных склонений. Метод `GetAll5` возвращает массив строк с пятью склонениями. Цикл `foreach` обеспечивает символ конца строки `"\r\n"` в конце каждой строки текстового поля.

Фрагмент работы программы представлен на рис. 15.14.

Убедиться в работоспособности программы можно, запустив соответствующий файл решения в папке WebКлиентMorpher.

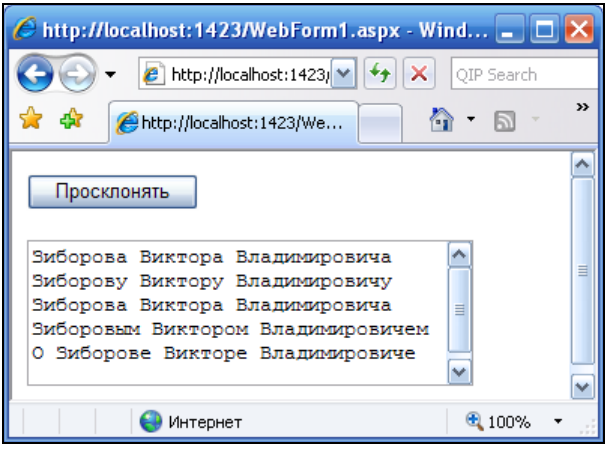


Рис. 15.14. Склонение слов обеспечивает Web-сервис

Пример 117. Получение данных от Web-службы Центрального банка РФ Web-приложением

Центральный банк Российской Федерации (ЦБРФ) предоставляет Web-службу для получения ежедневных экономических данных (курсы валют, учетные цены драгоценных металлов и проч.). Сервис данной Web-службы можно получать по адресу: **http://www.cbr.ru/scripts/Root.asp?Prtid=DWS**. Поставим задачу создания клиентского Web-приложения для получения ежедневных курсов валют. Такой сервис предлагается на сайте Центрального банка Российской Федерации по адресу: **http://www.cbr.ru/DailyInfoWebServ/DailyInfo.asmx.**

Мы воспользуемся функцией (методом) `GetSeldCursOnDate` данной Web-службы, на вход которой подают дату, а на выходе функции получают ежедневные курсы валют в виде `DataSet` (табличные данные).

Для решения этой задачи запустим Visual Studio 2010, выберем шаблон **Empty ASP.NET Web Application C#**. В поле **Name** зададим имя WebКлиентРоссия. К текущему проекту добавим Web-форму. Для этого в пункте меню **Project** выберем команду **Add New Item** и в появившемся окне дважды щелкнем на шаблоне **Web Form**. В конструкторе формы перетащим в форму командную кнопку **Button** и элемент сетки данных **GridView**, поскольку намереваемся с его помощью отображать табличные данные типа `DataSet`. Далее в пункте меню **Project** выберем команду **Add Web Reference** и в появившемся окне в поле **URL** введем адрес Web-сервиса:

http://www.cbr.ru/DailyInfoWebServ/DailyInfo.asmx

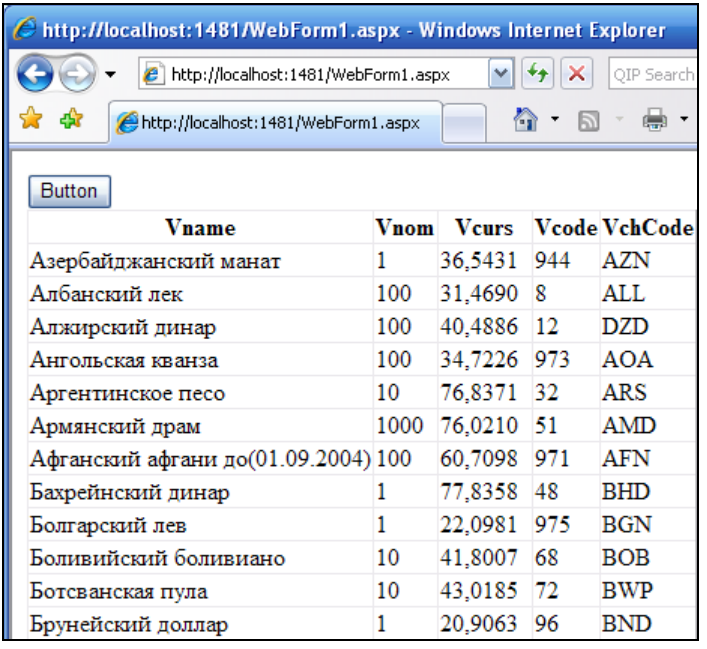
При этом станет доступной кнопка **Add Reference**. Щелкнем на этой кнопке, после этого в окне **Solution Explorer** появится новая папка **Web References** со ссылкой на удаленный класс: **ru.cbr.www**. Теперь в нашем программном коде мы сможем воспользоваться этим удаленным классом (листинг 15.8).

Листинг 15.10. Получение данных от Web-службы Центрального банка РФ Web-приложением

```
// Клиентское Web-приложение, потребляющее сервис Web-службы Центрального
// банка России для получения ежедневных курсов валют. На выходе приложения
// получаем таблицу курсов валют
using System;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace WebКлиентРоссия
{
   public partial class WebForm1 : System.Web.UI.Page
   {
```

```
    protected void Page_Load(object sender, EventArgs e)
    {
       Button1.Focus();
    }

    protected void Button1_Click(object sender, EventArgs e)
    {  // Создаем клиентское приложение Web-службы:
       //         http://www.cbr.ru/DailyInfoWebServ/DailyInfo.asmx
       // Создаем экземпляр удаленного класса:
       var Валюта = new ru.cbr.www.DailyInfo();
       System.DateTime Дата = System.DateTime.Now;
       // Получение ежедневных курсов валют:
       System.Data.DataSet НаборДанных = Валюта.GetSeldCursOnDate(Дата);
       // Содержимое DataSet в виде строки XML для отладки:
       string СтрокаXML = НаборДанных.GetXml();
       // Указываем источник данных для сетки данных:
       GridView1.DataSource = НаборДанных;
       GridView1.DataBind();
    }
  }
}
```



Vname	Vnom	Vcurs	Vcode	VchCode
Азербайджанский манат	1	36,5431	944	AZN
Албанский лек	100	31,4690	8	ALL
Алжирский динар	100	40,4886	12	DZD
Ангольская кванза	100	34,7226	973	AOA
Аргентинское песо	10	76,8371	32	ARS
Армянский драм	1000	76,0210	51	AMD
Афганский афгани до(01.09.2004)	100	60,7098	971	AFN
Бахрейнский динар	1	77,8358	48	BHD
Болгарский лев	1	22,0981	975	BGN
Боливийский боливиано	10	41,8007	68	BOB
Ботсванская пула	10	43,0185	72	BWP
Брунейский доллар	1	20,9063	96	BND

Рис. 15.15. Ежедневные курсы валют Web-сервиса ЦБРФ

Как и при создании предыдущих клиентов Web-служб, при обработке события "щелчок на кнопке" создаем экземпляр удаленного класса, в данном случае мы назвали его `Валюта`. На вход его метода `GetSeldCursOnDate` подаем текущую дату. Этот метод возвращает данные типа `DataSet`, которые мы указываем в качестве источника данных `DataSource` для элемента управления "сетка данных" `GridView1`.

Фрагмент работы программы показан на рис. 15.15.

Убедиться в работоспособности программы можно, открыв решение в папке WebКлиентРоссия. В заключение отмечу, что подобную Web-службу имеют также банки других стран, например Web-служба Национального банка Республики Беларусь расположена по адресу: **http://www.nbrb.by/Services/ExRates.asmx**.

Пример 118. Получение данных от Web-службы Национального банка Республики Беларусь Windows-приложением

Web-служба, предоставляющая текущие курсы валют, есть и у Национального банка Республики Беларусь. На примере потребления сервиса Web-службы этого банка создадим Windows-приложение (настольное приложение), получающее справочную информацию о текущих курсах валют в онлайновом режиме.

Для решения этой задачи запустим Visual Studio 2010, выберем шаблон **Windows Forms Application C#**. В поле **Name** зададим имя WebКлиентБеларусь. В конструкторе формы добавим кнопку **Button** и сетку данных **DataGridView**. Мы намереваемся в элемент управления **DataGridView** вывести данные с курсами валют. Элемент управления **DataGridView** в Windows-приложении играет ту же роль, что и **GridView** в Web-приложении. Методы этих классов похожи, хотя есть и отличия.

Следующим этапом разработки клиентского приложения является добавление ссылки на удаленный класс искомой Web-службы. Для этого выберем в пункте меню **Project** команду **Add Service Reference**, это приведет к появлению одноименного диалогового окна. В этом окне нажмем кнопку **Advanced**, а затем кнопку **Add Web Reference**. В появившемся диалоговом окне **Add Web Reference** в поле **URL** введем адрес Web-службы: **http://www.nbrb.by/Services/ExRates.asmx** и нажмем кнопку **Go**. В окне ниже увидим список доступных методов данного сервиса, после этого щелкнем на кнопке **Add Reference**. При этом в окне **Solution Explorer** появится ссылка на удаленный класс: **by.nbrb.www**. Теперь мы можем использовать эту ссылку в программном коде (листинг 15.11).

Листинг 15.11. Получение данных от Web-службы Национального банка Республики Беларусь Windows-приложением

```
// Клиентское Windows-приложение, потребляющее сервис Web-службы Национального
// банка Республики Беларусь для получения ежедневных курсов валют. На выходе
// приложения получаем таблицу курсов валют
```

```
using System;
using System.Data;
using System.Windows.Forms;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace WebКлиентБеларусь
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
      }
      private void button1_Click(object sender, EventArgs e)
      {  // Создаем клиентское приложение Web-службы:
         // http://www.nbrb.by/Services/ExRates.asmx.
         // Создаем экземпляр удаленного класса:
         var Валюта = new by.nbrb.www.ExRates();
         // А этот адрес я нашел на сайте:
         // http://ivbeg.bestpersons.ru/feed/post3279396/
         // Здесь есть ссылки на другие русскоязычные сервисы
         System.DateTime Дата = System.DateTime.Now;
         // На вход метода ExRatesDaily подаем текущую дату:
         DataSet НаборДанных = Валюта.ExRatesDaily(Дата);
         // Метод ExRatesDaily возвращает курсы валют в виде DataSet.
         // Содержимое DataSet в виде строки XML для отладки:
         string СтрокаXML = НаборДанных.GetXml();
         // Указываем источник данных для сетки данных:
         dataGridView1.DataSource = НаборДанных;
         // Указываем имя таблицы в наборе данных:
         dataGridView1.DataMember = "DailyExRatesOnDate";
      }
   }
}
```

В программном коде выполняем обычные уже в данной главе действия. При обработке события "щелчок на кнопке" `button1` создаем экземпляр удаленного класса, а затем обращаемся к методу класса `ExRates`, подавая на вход метода текущую дату. Этот метод возвращает таблицу курсов валют в формате `DataSet`. Эту таблицу, называемую `DailyExRatesOnDate`, указываем в качестве источника данных `DataSource` для сетки данных `dataGridView1`, которую мы используем для визуализации таблицы курсов валют.

Фрагмент работы программы представлен на рис. 15.16.

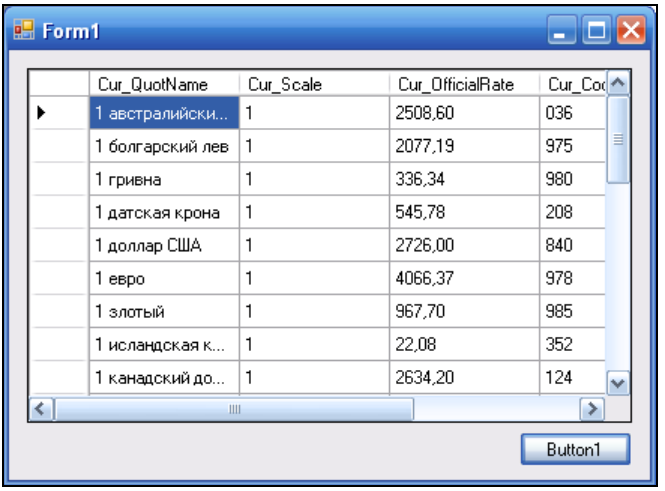


Рис. 15.16. Ежедневные курсы валют Web-службы НБРБ

Убедиться в работоспособности программы можно, открыв решение в папке WebКлиентБеларусь. Такое же Web-приложение, решающее ту же задачу вывода на сетку данных курсов валют, можно тестировать, запустив соответствующий файл решения в папке WebКлиентБеларусьWeb.

Глава 16

Использование технологии WPF

Что может нам дать WPF?

До сих пор мы обсуждали возможности программирования на C# либо настольных Windows-приложений (частный случай — консольные приложения), которые пользователи запускают непосредственно (например, exe-файлы), либо Web-приложений, выполняющихся внутри браузера. Оба этих приложения имеют свои положительные качества и недостатки. Например, настольные (автономные) приложения обладают большей гибкостью и реактивностью, выражающейся, например, практически в отсутствии задержки при выполнении команд. Даже если вы решаете систему уравнений большой размерности на современном компьютере в Windows-приложении, вы практически не ощущаете "запаздывания" с получением решения системы. В Web-приложении любая отправка данных серверу порождает задержку с ответом. Так, обычный запрос имени учетной записи пользователя (логин) и пароля в процедуре аутентификации пользователя к удаленной базе данных из Web-приложения может затянуться на продолжительное время с учетом скорости пропуска сетевого трафика. С другой стороны, к Web-приложениям могут иметь доступ одновременно многие пользователи.

Однако последнее время мы наблюдаем "сближение" возможностей Windows- и Web-приложений. Так, например, благодаря появлению Web-служб Windows-приложения (также как и Web-приложения) получили возможность обрабатывать не только данные, которые имеются непосредственно на винчестере данного настольного компьютера, но и данные, предоставленные той или иной Web-службой. А помощь (справка — <F1>) в использовании того или иного Windows-приложения часто осуществляется посредством обращения к Интернету. Все большую гибкость Web-приложения получают вследствие применения технологий JavaScript (выполнение сценариев на стороне браузера, а не на стороне сервера, что обеспечивает уменьшение "задержек" в работе приложения), благодаря AJAX полное обновление Web-страницы выполняется реже, Flash-технологии обеспечивают яркую и скоростную графику и т. д. Тем самым Web-приложения все больше становятся похожими на настольные приложения.

Технология Windows Presentation Foundation (WPF, графическая презентационная подсистема) задумана компанией Microsoft для того, чтобы еще более уменьшить этот разрыв. Приложение, написанное на основе WPF-технологии (WPF-приложение), способно функционировать как в виде автономного настольного Windows-приложения, так и в виде Web-приложения, выполняющегося в браузере. При этом разработчик WPF-приложения получает возможности мощного графического ядра и программного интерфейса для отрисовки различных графических элементов. Приведем маленький пример по поводу графических возможностей. При разработке Windows-приложений свойство `Opacity` (уровень непрозрачности) имеет только вся экранная форма, между тем в WPF-приложении это свойство имеют практически все элементы управления. Создается впечатление, что Microsoft по графическим возможностям в WPF-технологии стремится создать альтернативу Adobe Flash и Java-апплетам.

Еще одна очень важная особенность WPF-приложений — это возможность их работы в различных операционных средах. То есть разработанное вами на основе WPF-приложения автономное настольное приложение или приложение, выполняемое в браузере (так называемое XAML Browser Applications, XBAP-приложение), будет работать как в операционной системе Windows, так и в операционных системах семейства UNIX/Linux, операционных системах Solaris и Mac OS X. В результате этого практическая польза для Web-разработчиков состоит в следующем. Мы с вами говорили о том, что Web-хостинг, т. е. услуга по размещению в сети Web-страниц, существует в основном под уравлением операционных систем семейства UNIX/Linux. Хостингов на основе ASP.NET на Windows-платформе сравнительно немного. А это ограничивает написание Web-страниц на языках среды Visual Studio. Однако если мы будем писать Web-страницы на основе WPF-технологии, то проблема поиска необходимого хостинга отпадает, поскольку Web-страница, написанная на основе WPF-технологии, будет благополучно работать как на сервере под управлением UNIX/Linux, так и на Windows-сервере.

Пример 119. Создание простейшего WPF-приложения. Компоновка элементов управления с помощью сетки *Grid*

Поставим следующую задачу: требуется создать интерактивную Web-страницу, содержащую в себе текстовое поле, метку и кнопку. Данное приложение вычисляет значение квадратного корня из числа, введенного пользователем. После щелчка на кнопке приложение производит диагностику введенных символов, и если пользователь действительно ввел число, то на текстовую метку выводим результат извлечения корня. Это совсем маленькая задача, но на ее основе мы сможем рассмотреть все этапы создания WPF-приложения.

Итак, запустим Visual Studio 2010, закажем новый проект шаблона **WPF Browser Application C#** (из инсталлированных шаблонов **Windows**), назовем новый проект WpfXbapSqrt. После щелчка на кнопке **OK** попадаем на вкладку **Page1.xaml**. В WPF-приложении, так же как и в ASPX-приложении, имеем файлы разметки, здесь они написаны на языке XAML. Язык XAML (Extensible Application Markup Language) — это язык разметки для декларативного программирования WPF-приложений. Так же как и в ASP-технолонии кроме xaml-файлов разметки имеем соответствующие xaml.cs-файлы программной поддержки. На вкладке **Page1.xaml** размещают элементы управления, которые перетаскивают из панели элементов **Toolbox** на проектируемую Web-страницу. То есть создание WPF-приложений практически не отличается от создания приложений на основе Windows Forms.

Однако, перед тем как размещать и компоновать необходимые элементы, откроем вкладку **Page1.xaml** в режиме XAML (листинг 16.1). Обратите внимание на тег `<Grid>`, его назначение — определять сетку с рядами `Row` и столбцами `Column` для размещения в ячейках этой сетки (таблицы) элементов управления. Элемент управления **Grid** иногда называют контейнером макета страницы. Линии сетки (Grid-линии) делают невидимыми `ShowGridLines = false`, это позволяет компоновать элементы управления в нужном порядке.

Листинг 16.1. Проектируемая Web-страница на языке XAML

```
<Page x:Class="WpfXbapSqrt.Page1"
     xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
     xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
     xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
     mc:Ignorable="d"
     d:DesignHeight="300" d:DesignWidth="300"
     Title="Page1">
   <Grid>

   </Grid>
</Page>
```

Мы намереваемся управлять этой сеткой из соответствующего файла программной разметки и обращаться к обсуждаемой сетке, как к программному объекту. Однако на данный момент у этой сетки нет имени, поэтому в файле разметки для тега `<Grid>` зададим имя сетки таким образом: `<Grid Name="grid1">`. Мы будем использовать это имя позже, а сейчас перенесем на форму из панели элементов **Toolbox** (эту панель можно вызвать, как обычно, комбинацией клавиш <Ctrl>+ +<Alt>+<X>) необходимые нам текстовое поле **TextBox**, метку **Label** и командную кнопку **Button**. Можно было бы, просто последовательно дважды щелкая по упомянутым элементам в панели **Toolbox**, добавлять их на Web-страницу, в этом слу-

чае они расположились бы один под другим, как это нам необходимо. Однако хотелось бы иметь гибкую методику размещения элементов управления в любом месте Web-страницы. Эля этого как раз мы и упоминали сетку **Grid**. Вообще говоря, для целей размещения элементов на Web-странице имеем ряд компонентов, входящих в состав WPF:

- **Canvas** — дочерние компоненты могут сами управлять своим расположением;
- **DockPanel** — дочерние компоненты выравниваются по краям панели;
- **Grid** — дочерние компоненты располагаются в рядах и колонках;
- **StackPanel** — дочерние компоненты располагаются в одну строку вертикально или горизонтально;
- **VirtualizingStackPanel** — дочерние компоненты выравниваются по одной линии — горизонтальной или вертикальной;
- **WrapPanel** — дочерние компоненты располагаются слева направо и "заворачиваются" на следующую строку, когда заканчивается место на текущей строке.

Прежде чем перейти на вкладку программного кода, зададим, что сетка, в ячейках которой мы собираемся размещать эти три элемента управления, имеет одну колонку и три ряда. Для этого через контексное меню дизайнера Web-страницы попадаем на вкладку **Properties** (Свойства). И для сетки `grid1` находим свойство `ColumnDefinitions`. Щелкая на многоточии, добавляем (**Add**) одну колонку. Аналогично в строке свойства `RowDefinitions` добавляем три ряда. Вообще говоря, можно было задать колонку и три ряда в программном коде, но сделать это в свойствах сетки **Grid** совсем просто.

Теперь переходим на вкладку программного кода **Page1.xaml.cs** и вводим текст программы, представленный в листинге 16.2.

Листинг 16.2. Содержание файла программной поддержки простейшего WPF-приложения

```
// Данное WPF-приложение вычисляет значение квадратного корня из числа,
// введенного пользователем в текстовое поле. После щелчка на кнопке
// приложение производит диагностику введенных символов, и если пользователь
// действительно ввел число, то в текстовую метку выводим результат
// извлечения корня
using System;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Media;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace WpfXbapSqrt
{
   /// <summary>
   /// Interaction logic for Page1.xaml
```

```
/// </summary>
public partial class Page1 : Page
{
   public Page1()
   {
      InitializeComponent();
      // Строка заголовка Web-страницы:
      this.WindowTitle = "Извлекаю корень";
      // Компонуем элементы управления с помощью сетки Grid:
      grid1.Width = 200; grid1.Height = 100;
      // Сетку grid1 размещаем в левом верхнем углу Web-страницы:
      grid1.HorizontalAlignment = System.Windows.HorizontalAlignment.Left;
      grid1.VerticalAlignment = System.Windows.VerticalAlignment.Top;
      // Удобно для отладки показать линии сетки:
      // grid1.ShowGridLines = true;
      // Расстояния от краев Web-страницы до сетки grid1:
      grid1.Margin = new Thickness(10, 10, 10, 10);
      // Цвет объекта grid1:
      grid1.Background = Brushes.LightGray;
      // Указываем, что текстовое поле расположить в нулевой колонке в
      // нулевом ряду:
      Grid.SetRow(textBox1, 0); Grid.SetColumn(textBox1, 0);
      Grid.SetRow(label1, 1);   Grid.SetColumn(label1, 0);
      Grid.SetRow(button1, 2);  Grid.SetColumn(button1, 0);
      // Текстовое поле располагаем в левом нижнем углу ячейки, сдвинув
      // его на 10 пикселов:
      textBox1.HorizontalAlignment =
         System.Windows.HorizontalAlignment.Left;
      textBox1.VerticalAlignment = System.Windows.VerticalAlignment.Bottom;
      textBox1.Margin = new Thickness(10, 0, 0, 0); textBox1.Focus();
      label1.HorizontalAlignment = System.Windows.HorizontalAlignment.Left;
      label1.VerticalAlignment = System.Windows.VerticalAlignment.Bottom;
      label1.Margin = new Thickness(10, 0, 0, 0);
      button1.HorizontalAlignment =
         System.Windows.HorizontalAlignment.Left;
      button1.VerticalAlignment = System.Windows.VerticalAlignment.Top;
      button1.Margin = new Thickness(10, 0, 0, 0);
      button1.Content = "Извлечь корень"; button1.Width = 120;
      label1.Content = "Введите число в текстовое поле";
      label1.Foreground = Brushes.Blue; // - синий цвет текста на метке
   }
```

```
    private void button1_Click(object sender, RoutedEventArgs e)
    {
        Single X; // - из этого числа будем извлекать корень
        // Преобразование из строковой переменной в Single:
        bool Число_ли = Single.TryParse(textBox1.Text,
            System.Globalization.NumberStyles.Number,
            System.Globalization.NumberFormatInfo.CurrentInfo, out X);
        // Второй параметр - это разрешенный стиль числа (Integer,
        // шестнадцатиричное число, экспоненциальный вид числа и прочее).
        // Третий параметр форматирует значения на основе текущего языка
        // и региональных параметров
        // из Панели управления | Язык и региональные стандарты
        // число допустимого формата.
        // Метод возвращает значение в переменную X.
        if (Число_ли == false)
        {  // Если пользователь ввел не число:
            label1.Content = "Следует вводить числа";
            label1.Foreground = Brushes.Red; // - красный цвет текста на метке
            return;                          // - выход из процедуры
        }
        Single Y = (Single)Math.Sqrt(X);    // - извлечение корня
        label1.Foreground = Brushes.Blue;   // - синий цвет текста на метке
        label1.Content = string.Format("Корень из {0} равен {1:F5}", X, Y);
    }
  }
}
```

Как видно из программного кода, задачу компоновки элементов управления в ячейках сетки `grid1` решаем сразу после выполнения процедуры `InitializeComponent`. При обработке события "щелчок на кнопке" `button1` проверяем, число ли ввел пользователь в текстовое поле, и выводим результат извлечения корня в метку `label1`.

Фрагмент работы данного WPF-приложения представлен на рис. 16.1.

Теперь очень интересно проверить, как это WPF-приложение будет работать при размещении его на каком-нибудь хостинге, работающем под управлением ОС UNIX (таких хостингов подавляющее большинство, а хостингов из-под Windows значительно меньше). Мы не случайно в самом начале назвали данный проект WpfXbarSqrt, использовав при этом только английские буквы, поскольку имена файлов в ОС UNIX не должны содержать кириллицы. Файлы, которые следует размещать на хостинге, расположены в папке bin\Debug, этих файлов — четыре, они имеют следующие типы: exe, pdb, xbap и manifest. Это исполняемый файл, он содержит скомпилированный код, манифест приложения содержит метаданные, связанные с приложением, и непосредственно xbap-файл, который открывает брау-

зер. В качестве браузера следует использовать Internet Explorer или Mozilla Firefox с соответствующими плагинами.

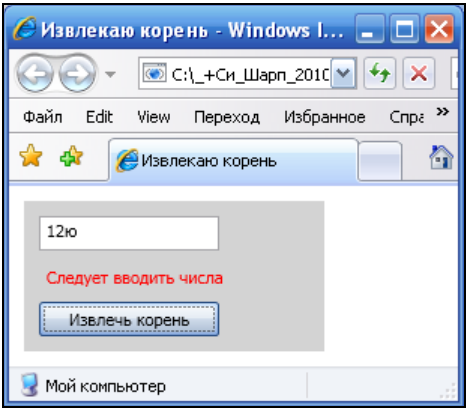


Рис. 16.1. Результат диагностики вводимых данных

Убедиться в работоспособности программы можно, открыв решение в папке WpfXbarSqrt.

Пример 120. Использование одного из эффектов анимации

Как уже упоминалось, Microsoft стремится по графическим возможностям в WPF-технологии приблизиться к возможностям Adobe Flash. Покажем на следующем примере возможность анимации (т. е. способности двигаться) кнопки и изображения при щелчке мыщью на них. При щелчке на кнопке она резко расширится, а затем плавно вернется в исходное состояние. Аналогично при щелчке на изображении его размер увеличится с последующим плавным уменьшением до первоначального. Здесь использована наиболее распространенная техника анимации — это анимация интерполяцией, при которой свойство модифицируется плавно от начальной точки до конечной.

Для демонстрации этой возможности запустим Visual Studio 2010, выберем среди инсталлированных шаблонов Windows шаблон **WPF Browser Application** и назовем его WpfXbapАниме. Чтобы на вкладке **Page1.xaml** появились кнопка и изображение, на панели элементов **Toolbox** дважды щелкнем по элементам управления **Button** и **Image**.

Скопируем в папку проекта изображение poryv.png, а затем добавим это изображение в проект командой **Project | Add Existing Item**. Теперь перейдем на вкладку программного кода **Page1.xaml.cs** и введем текст программы, представленный в листинге 16.3.

Листинг 16.3. WPF-приложение демонстрирует одну из возможностей анимации

```
// WPF-приложение выводит на Web-страницу командную кнопку и изображение. При
// щелчке на кнопке и на изображении демонстрируются возможности анимации:
// кнопка расширяется, а затем медленно уменьшается до исходных размеров;
// аналогично поведение изображения
using System;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Input;
using System.Windows.Media.Imaging;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace WpfXbapАниме
{
   /// <summary>
   /// Interaction logic for Page1.xaml
   /// </summary>
   public partial class Page1 : Page
   {
      public Page1()
      {
         InitializeComponent();
         this.WindowTitle = "Анимационный эффект";
         button1.Width = 100;
         button1.Content = "Привет!";   button1.Focus();
         image1.Source = new BitmapImage(new Uri("poryv.png",
                                          UriKind.Relative));
         image1.Width = 100; image1.Height = 100;
      }

      private void button1_Click(object sender, RoutedEventArgs e)
      {
         var Аниме = new System.Windows.Media.Animation.DoubleAnimation();
         // Изменить размер от 160 до 100 пикселов:
         Аниме.From = 160; Аниме.To = 100;
         // Продолжительность анимационного эффекта 5 секунд:
         Аниме.Duration = TimeSpan.FromSeconds(5);
         // Начать анимацию:
         button1.BeginAnimation(Button.WidthProperty, Аниме);
      }
```

```
      private void image1_MouseLeftButtonUp(object sender,
                                            MouseButtonEventArgs e)
      {  // Щелчок на изображении:
         var Аниме = new System.Windows.Media.Animation.DoubleAnimation();
         Аниме.From = 160; Аниме.To = 100;
         Аниме.Duration = TimeSpan.FromSeconds(5);
         image1.BeginAnimation(Image.WidthProperty, Аниме);
         image1.BeginAnimation(Image.HeightProperty, Аниме);
      }
   }
}
```

Сразу после инициализации компонентов страницы `InitializeComponent` устанавливаем ширину кнопки и размеры изображения равными 100 пикселам. Размеры изображения автор взял из его свойств (контекстное меню изображения **Свойства | Сводка**). При щелчке на кнопке создаваемого WPF-приложения инициализируем новый экземпляр класса `DoubleAnimation`. Далее указываем, в каких пределах изменять ширину кнопки и продолжительность процесса анимации, и, наконец, командой `BeginAnimation` начинаем анимационный эффект с командной кнопкой `button1`. Аналогичные установки и команды выполняем при щелчке мышью на изображении, только анимационным преобразованиям подвергаем одновременно ширину и высоту (`Height`) изображения.

Фрагмент работы программы показан на рис. 16.2.

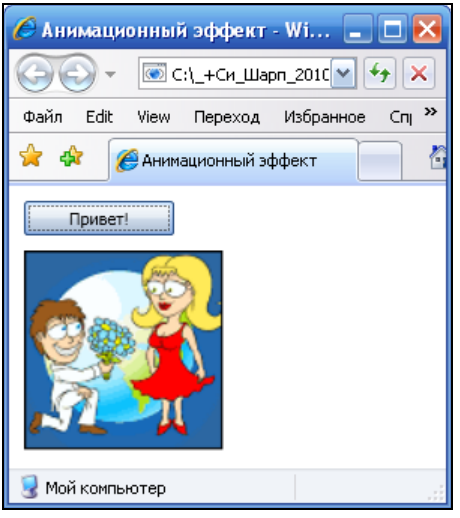


Рис. 16.2. Изображение плавно уменьшается

Убедиться в работоспособности программы можно, открыв решение в папке WpfXbarАниме.

Пример 121. Эффект постепенной замены (прорисовки) одного изображения другим

Создадим WPF-приложение, выполняемое в браузере, содержащее на Web-странице два изображения. В качестве изображений мы выбрали презентационные рисунки автомобилей Chevrolet Cavalier и BMW M3. Поскольку расположение их на странице является одинаковым, а также их размеры совпадают, пользователь видит только одно "верхнее" изображение. Таким образом, при загрузке страницы мы видим лишь автомобиль BMW M3. При щелчке мышью на видимом изображении оно становится все более прозрачным, постепенно "проявляя" "нижнее" изображение. Анимационный эффект длится пять секунд, в конце которого пользователь видит уже автомобиль Chevrolet Cavalier. Повторный щелчок мышью на изображении Chevrolet постепенно "проявляет" опять "верхнее" изображение, и мы вновь имеем честь видеть автомобиль BMW.

Для решения поставленной задачи запустим Visual Studio 2010, выберем среди инсталлированных шаблонов Windows шаблон **WPF Browser Application** и назовем его WpfXbapДваИзо. Далее, перейдя на вкладку дизайна **Page1.xaml**, на панели элементов **Toolbox** дважды щелкнем по элементу **Image**. Поскольку нам требуются два таких элемента, сделаем это еще раз. Мы можем совершенно не волноваться, как будут расположены эти элементы на Web-странице, и какие их размеры будут заданы, поскольку об этом мы позаботимся в программном коде. Теперь скопируем в папку проекта изображения c1.bmp и c2.bmp (рисунки автомобилей Chevrolet Cavalier и BMW M3), а затем добавим эти изображения в проект командой **Project | Add Existing Item**. В окне **Solution Explorer** появятся значки этих двух файлов. Теперь перейдем на вкладку программного кода **Page1.xaml.cs** и введем текст программы, представленный в листинге 16.4.

Листинг 16.4. Постепенная замена одного изображения другим

```
// WPF-приложение содержит на Web-странице два изображения. Поскольку
// месторасположение обоих изображений задано одинаково, а также совпадают
// размеры изображений, пользователь будет видеть только второе "верхнее"
// изображение. После щелчка на изображении оно становится все более
// прозрачным, постепенно "проявляя" тем самым "нижнее" изображение.
// После исчезновения "верхнего" изображения мы будем видеть только "нижнее"
// изображение. При повторном щелчке на изображении, наоборот, прозрачность
// верхнего изображения постепенно снижается, и в конце анимационного
// эффекта мы опять видим только "верхнее" изображение
using System;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Input;
```

```
using System.Windows.Media.Imaging;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace WpfXbapДваИзо
{
   /// <summary>
   /// Логика взаимодействия для Page1.xaml
   /// </summary>
   public partial class Page1 : Page
   {
      bool Флажок = false;
      public Page1()
      {
         InitializeComponent();
         this.WindowTitle = "Щелкни на изображении";
         // Изображение автомобиля Chevrolet Cavalier:
         image1.Source = new BitmapImage(new Uri("c1.bmp", UriKind.Relative));
         // Изображение автомобиля BMW M3:
         image2.Source = new BitmapImage(new Uri("c2.bmp", UriKind.Relative));
         // Размеры изображений:
         image1.Width = 591; image1.Height = 258;
         image2.Width = 591; image2.Height = 258;
         // Расстояния от краев Web-страницы до сетки Grid:
         image1.Margin = new Thickness(10, 10, 0, 0);
         image2.Margin = new Thickness(10, 10, 0, 0);
         // Присоединяем один обработчик двух событий:
         image1.MouseDown += new MouseButtonEventHandler(image_MouseDown);
         image2.MouseDown += new MouseButtonEventHandler(image_MouseDown);
         // image2.Opacity = 1; Свойство Opacity задает уровень непрозрачности
      }
      private void image_MouseDown(object sender, RoutedEventArgs e)
      {  // Процедура обработки события "щелчок" на любом из изображений.
         // Изменяем состояние флажка на противоположное:
         Флажок = !Флажок;
         // Создаем объект анимации:
         var Аниме = new System.Windows.Media.Animation.DoubleAnimation();
         // Устанавливаем пределы изменения степени непрозрачности:
         if (Флажок == true)
           { Аниме.From = 1; Аниме.To = 0; }
         else
           { Аниме.From = 0; Аниме.To = 1; }
         // Продолжительность анимационного эффекта задаем равной 5 секундам:
```

```
            Аниме.Duration = TimeSpan.FromSeconds(5);
            // Запускаем анимацию для "верхнего" изображения:
            image2.BeginAnimation(Image.OpacityProperty, Аниме);
        }
    }
}
```

Как видно из программного кода, вначале задаем булеву переменную `Флажок` так, чтобы она была "видна" из обеих процедур класса `Page1`. Исходя из состояний этой переменной `false` или `true`, будем менять первое изображение на второе или наоборот. Сразу после вызова процедуры `InitializeComponent` задаем принадлежность рисунков c1.bmp и c2.bmp изображениям `image1` и `image2`. Также задаем их размер и месторасположение на странице относительно сетки **Grid**. Присоединяем одну процедуру обработки двух событий, а именно, щелчок любой кнопкой мыши на одном изображении и на другом. Ниже пишем непосредственно процедуру обработки упомянутых событий `image_MouseDown`. В этой процедуре в зависимости от состояния флажка задаем пределы изменения степени непрозрачности либо от 1 до 0, либо от 0 до 1. Затем запускаем анимацию для "верхнего" изображения.

На рис. 16.3 показан фрагмент работы программы.

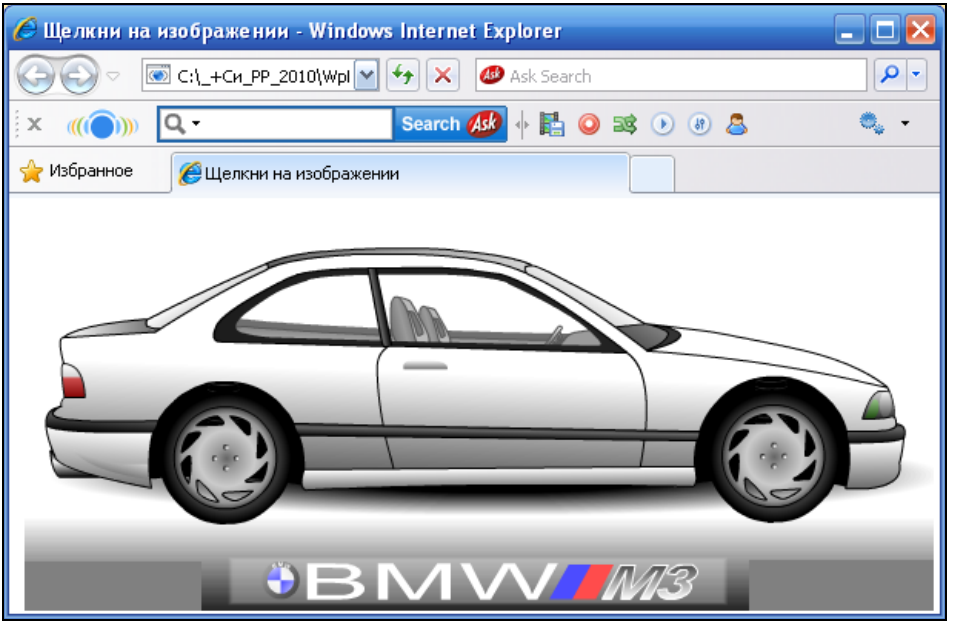


Рис. 16.3. После щелчка на изображении BMW M3 постепенно прорисовывается автомобиль Chevrolet

Убедиться в работоспособности программы можно, открыв решение в папке WpfXbarДваИзо.

Пример 122. Закрашивание области текста горизонтальным линейным градиентом

Создадим автономное настольное WPF-приложение, включающее в себе текстовый блок, содержащий некоторый текст. Чтобы подчеркнуть значимость написанного, закрасим область текста с применением градиента. Градиент — это вид заливки в компьютерной графике, в которой необходимо задать цвета ключевых точек, а цвета остальных точек рассчитываются линейной интерполяцией. Таким образом, можно получать плавные переходы из одного цвета в другой, задав координаты и цвет начальной и конечной точек.

Для решения этой задачи запустим Visual Studio 2010, выберем среди инсталлированных шаблонов Windows шаблон **WPF Application**, т. е. автономное WPF-приложение, и назовем его WpfGradientText. Далее, перейдя на вкладку дизайна **MainWindows.xaml**, на панели элементов **Toolbox** дважды щелкнем по элементу **TextBlock**. Его размеры будем задавать в программном коде, поэтому сразу переходим на вкладку кода **MainWindows.xaml.cs** (листинг 16.5).

Листинг 16.5. Плавный переход от одного цвета к другому в области текстового блока

```
// Автономное WPF-приложение содержит текстовый блок. Цвет текста в этом блоке
// закрашен с применением градиента. Между начальной t=0.0 и конечной t=1.0
// точками области текста заданы две ключевые точки t=0.25 и t=0.75.
// Каждой точке ставим в соответствие цвета: желтый, красный, синий
// и зеленый. Между этими цветами задаем плавный переход от одного цвета
// к другому с помощью градиента
using System.Windows;
using System.Windows.Media;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace WpfGradientText
{
   /// <summary>
   /// Interaction logic for MainWindow.xaml
   /// </summary>
   public partial class MainWindow : Window
   {
      public MainWindow()
      {
         InitializeComponent();
         this.Title = "Игорь Губерман";
         textBlock1.Text = "Какие дамы, и не раз, шептали: \"Дорогой,\n" +
              "Конечно, да, но не сейчас, не здесь, и не с тобой!\"";
         textBlock1.Width = 470; textBlock1.Height = 50;
```

```
            textBlock1.FontSize = 20;
            // textBlock1.Background = Brushes.Gray;
            // Создаем горизонтальный линейный градиент
            LinearGradientBrush Градиент = new LinearGradientBrush();
            // Задаем область для закрашивания линейным градиентом:
            Градиент.StartPoint = new Point(0, 0.5);
            Градиент.EndPoint = new Point(1, 0.5);
            // Четыре точки t=0.0; t=0.25; t=0.75 и t=1.0
            // образуют три подобласти с переходом цвета
            // от желтого к красному, далее синему и затем зеленому
            Градиент.GradientStops.Add(new GradientStop(Colors.Yellow, 0.0));
            Градиент.GradientStops.Add(new GradientStop(Colors.Red, 0.25));
            Градиент.GradientStops.Add(new GradientStop(Colors.Blue, 0.75));
            Градиент.GradientStops.Add(new GradientStop(Colors.LimeGreen, 1.0));
            // Закрашиваем текст горизонтальным линейным градиентом:
            textBlock1.Foreground = Градиент;
        }
    }
}
```

Как видно из программного кода, сразу после вызова процедуры `InitializeConponent` задаем шуточный афоризм Игоря Губермана для свойства текстового блока `TextBlock1.Text`. Далее создаем горизонтальный линейный градиент между четырех точек, соответственно желтого, красного, синего и зеленого цветов. Градиент обеспечивает постепенный переход от одного цвета к другому (рис. 16.4).

Убедиться в работоспособности программы можно, открыв решение в папке WpfGradientText.

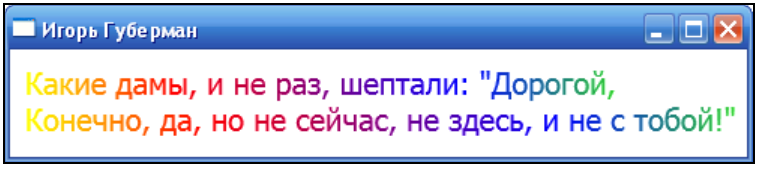


Рис. 16.4. Закрашивание текстового блока горизональным линейным градинентом

Пример 123. Проверка орфографии в элементе управления редактирования текста

Если читатель помнит, в примере 54 *главы 9* мы рассматривали возможность проверки орфографии (правописания) в текстовом поле и при этом использовали

объектную библиотеку MS Word. Платформа .NET Framework 4 WPF позволяет обойтись без обращения к библиотеке MS Word, поскольку содержит в себе возможность проверки орфографии в редактируемом элементе управления, таком как **TextBox** или **RichTextBox**. При этом варианты проверки орфографии отображаются в виде контекстного меню. Например, когда пользователь щелкает правой кнопкой мыши по слову с ошибкой, он получает набор орфографических вариантов или вариант **Ignore All** (Пропустить все).

Создадим автономное настольное WPF-приложение, включающее в себе текстовое поле с возможностью проверки орфографии. Для этого запустим Visual Studio 2010, выберем среди шаблонов Windows шаблон **WPF Application**, т. е. автономное WPF-приложение, и назовем его WpfTextBoxОрфография. Далее, перейдя на вкладку дизайна **MainWindows.xaml**, на панели элементов **Toolbox** дважды щелкнем по элементу **TextBox**. Его расположение, размеры и другие свойства будем задавать в программном коде, поэтому сразу перейдем на вкладку кода **MainWindows.xaml.cs** (листинг 16.6).

Листинг 16.6. Включение проверки орфографии в элементе управления редактирования текста *TextBox*

```
// Автономное WPF-приложение содержит элемент управления TextBox с включенной
// проверкой правописания англоязычного текста.
// Технология .NET Framework 4 WPF обеспечивает только английский,
// французский, немецкий и испанский словари.
// Чтобы появилась возможность проверять русскоязычный текст, следует
// в коллекцию CustomDictionaries добавить пользовательский словарь
// русскоязычной лексики *.lex
using System;
using System.Windows;
using System.Windows.Controls;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace WpfTextBoxОрфография
{
   /// <summary>
   /// Interaction logic for MainWindow.xaml
   /// </summary>
   public partial class MainWindow : Window
   {
      public MainWindow()
      {
         InitializeComponent();
         this.Title = "Проверка орфографии";
         // Размеры окна:
```

```
            this.Width = 300; this.Height = 150;
            // Расстояния от краев текстового поля до краев окна:
            textBox1.Margin = new Thickness(10, 10, 0, 0);
            // Размеры текстового поля:
            textBox1.Width = 270; textBox1.Height = 95; textBox1.Focus();
            // Включить проверку орфографии:
            textBox1.SpellCheck.IsEnabled = true;
            // Можно включить проверку орфографии также таким образом:
            // SpellCheck.SetIsEnabled(textBox1, true);
            // Разрешить перенос слов на другую строку:
            textBox1.TextWrapping = TextWrapping.Wrap;
            // При нажатии клавиши <Enter> разрешить переход на следующую строку:
            textBox1.AcceptsReturn = true;
            // Коллекция словарей для проверки орфографии:
            System.Collections.IList Словари = SpellCheck.
                                        GetCustomDictionaries(textBox1);
            // Добавить в коллекцию словарей словарь, созданный нами:
            Словари.Add(new Uri(@"C://dic.lex"));
            // Software is like sex, it's better when it's free
        }
    }
}
```

Как видно из текста программы, чтобы включить средство проверки правописания, свойству `SpellCheck.IsEnabled` присвоим значение `true` в элементе управления редактирования текста. При включении этого средства проверки орфографии слова с ошибками подчеркиваются красной волнистой линией. Технология .NET Framework 4 WPF обеспечивает только английский, французский, немецкий и испанский словари.

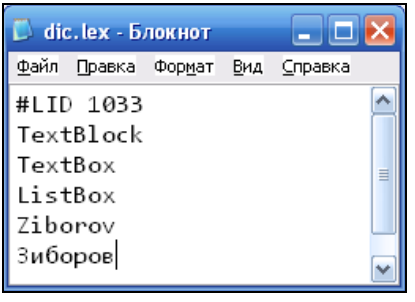


Рис. 16.5. Пользовательский словарь разрешенной лексики

Чтобы появилась возможность проверять русскоязычный текст, следует в коллекцию CustomDictionaries добавить пользовательский словарь русскоязычной лек-

сики. Пользовательские словари используют файлы лексики, которые представляют собой текстовые файлы с расширением lex. Автор не нашел такого готового файла, возможно, читатель найдет его самостоятельно. Текстовый lex-файл лексики имеет следующую структуру (рис. 16.5).

В этом файле показаны первые несколько строк файла лексики. Каждая строка файла лексики содержит отдельное слово, утвержденное для проверки орфографии. Первая строка файла может указывать код языка и региональных параметров (LCID), к которым применяется словарь. Если языковой стандарт не указан, словарь применяется для всех языков. Как видно, файл dic.lex содержит фамилию автора на английском языке, поэтому при проверке орфографии слово "Ziborov" не было подчеркнуто красной волнистой линией (рис. 16.6), а английское слово "free" написано с ошибкой, контекстное меню приводит варианты проверки орфографии.

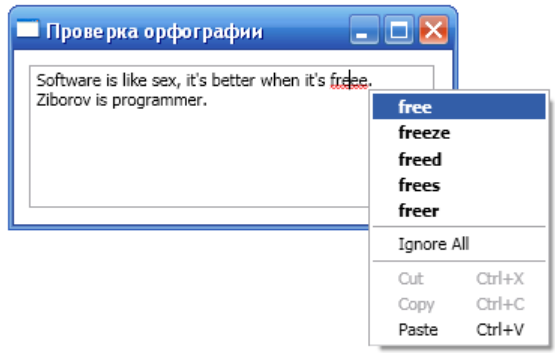


Рис. 16.6. Слова с ошибками подчеркиваются красной волнистой линией

Убедиться в работоспособности программы можно, открыв решение в папке WpfTextBoxОрфография.

Пример 124. Программирование WPF-проигрывателя. Компоновка элементов управления с помощью панели *StackPanel*

Среди элементов управления WPF на панели **Toolbox** можно найти элемент воспроизведения файлов мультимедиа **MediaElement**. На основе этого элемента в данном примере создадим автономное настольное WPF-приложение, реализующее проигрыватель мультимедиа с возможностями включения паузы, остановки и регулировки уровня громкости.

Для этой цели запустим среду Visual C# 2010, выберем среди шаблонов Windows шаблон **WPF Application**, т. е. автономное WPF-приложение, и назовем

его WpfПроигрыватель. Далее перейдем на вкладку дизайна **MainWindows.xaml**. В этом примере для компоновки элементов управления мы будем использовать элемент **StackPanel**. Элемент **StackPanel** используется для расположения элементов по вертикали (по умолчинию) или горизонтали. Эти элементы, перенесенные на вкладку дизайна после "стек-панели", называют дочерними по отношению к элементу **StackPanel**. Поэтому переключим вкладку **MainWindows.xaml** в состояние **Design**, а затем на панели элементов **Toolbox** (эту панель можно вызвать, как обычно, комбинацией клавиш <Ctrl>+<Alt>+<X>) дважды щелкнем на элементе **StackPanel**. Далее на пенели **Toolbox** дважды щелкнем последовательно на элементах **MediaElement** и **StackPanel**, т. е. вторая "стек-панель" будет вложена в первую. Иными словами, первый элемент **StackPanel** будет содержать в себе (будет иметь в качестве дочерних) элементы **MediaElement** и вторую **StackPanel**. Эти дочерние элементы будут располагаться по вертикали. Теперь, чтобы для второй "стек-панели" перенести три кнопки (**Play**, **Pause** и **Stop**), текстовый блок и элемент выбора из диапазона значений **Slider** (так называемый "ползунок"), дважды щелкнем по ним последовательно на пенели **Toolbox**. Мы хотим расположить их горизонтально (см. рис. 16.7), но об этом и о многом другом мы позаботимся в программном коде. Хотя часто это делают на языке XAML (это оказывается более удобным), но не забываем, что мы пишем книгу о языке C#. Перед тем, как перейти на вкладку программного кода, посмотрим содержимое файла разметки **MainWindows.xaml** (листинг 16.7).

Листинг 16.7. Разметка дизайна интерфейса WPF-проигрывателя

```
<Window x:Class="WpfПроигрыватель.MainWindow"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      Title="MainWindow" Height="350" Width="525">
   <Grid>
      <StackPanel Height="100" HorizontalAlignment="Left" Margin="10,10,0,0"
                  Name="stackPanel1" VerticalAlignment="Top" Width="200">
         <MediaElement Height="120" Name="mediaElement1" Width="160" />
         <StackPanel Height="100" Name="stackPanel2" Width="200">
            <Button Content="Button" Height="23" Name="button1" Width="75" />
            <Button Content="Button" Height="23" Name="button2" Width="75" />
            <Button Content="Button" Height="23" Name="button3" Width="75" />
            <TextBlock Height="23" Name="textBlock1" Text="TextBlock" />
            <Slider Height="23" Name="slider1" Width="100" />
         </StackPanel>
      </StackPanel>
   </Grid>
</Window>
```

Из XAML-разметки видно, что внешний элемент **StackPanel** включает в себя элементы **MediaElement** и **StackPanel**, и эта вторая **StackPanel** в свою очередь содержит три кнопки **Button**, текстовый блок **TextBlock** и "ползунок" **Slider**.

Теперь перейдем на вкладку **MainWindows.xaml.cs** и напишем программный код, представленный в листинге 16.8. При этом не забываем, что для каждого программируемого события, например `button1_Click`, следует его создать, скажем, используя окно **Properties**. При этом эти события отобразяться в XAML-разметке.

Листинг 16.8. Использование элемента управления *MediaElement* при программировании WPF-проигрывателя

```
// WPF-проигрыватель, позволяющий воспроизводить мультимедиа, включать паузу,
// остановку, а также настраивать громкость с помощью "ползунка"
using System;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Media;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace WpfПроигрыватель
{
   /// <summary>
   /// Interaction logic for MainWindow.xaml
   /// </summary>
   public partial class MainWindow : Window
   {
      public MainWindow()
      {
         InitializeComponent();
         // Строка заголовка Web-страницы:
         this.Title = "Мадиа проигрыватель";
         this.Width = 430; this.Height = 430;
         // Компонуем элементы управления с помощью панели StackPanel:
         stackPanel1.Width = 410; stackPanel1.Height = 400;
         stackPanel2.Width = 410; stackPanel2.Height = 50;
         // При отладке удобно задать фоновый цвет панели:
         // stackPanel1.Background = Brushes.LightGray;
         stackPanel1.Orientation = Orientation.Vertical;
         stackPanel2.Orientation = Orientation.Horizontal;
         button1.Focus(); button1.Content = "Play";
         button2.Content = "Pause"; button3.Content = "Stop";
         // Расстояния от краев текстового блока до краев окна:
         textBlock1.Margin = new Thickness(10, 13, 10, 0);
```

```
    textBlock1.Text = "Громкость:"; textBlock1.FontSize = 12;
    // Элемент выбора из диапазона значений "Ползунок" (Slider):
    slider1.Minimum = 0; slider1.Maximum = 1; slider1.Value = 0.5;
    // MediaElement представляет элемент, содержащий аудио и/или видео:
    mediaElement1.Source = new Uri(@"C://ВысшийПилотаж.mp4");
    // или mediaElement1.Source = new Uri(@"C:\ВысшийПилотаж.mp4");
    mediaElement1.Width = 400; mediaElement1.Height = 340;
    mediaElement1.HorizontalAlignment = System.
                              Windows.HorizontalAlignment.Left;
    // Manual - состояние, используемое для управления элементом
    // MediaElement вручную, при этом можно использовать интерактивные
    // методы, такие как Play и Pause:
    mediaElement1.LoadedBehavior = MediaState.Manual;
    mediaElement1.UnloadedBehavior = MediaState.Stop;
    mediaElement1.Stretch = Stretch.Fill;
}

private void button1_Click(object sender, RoutedEventArgs e)
{  // Чтобы начать просмотр ролика, следует подать команду Play.
   // Если MediaElement пребывает в состоянии Pause, то команда Play
   // продолжит воспроизведение ролика с текущего положения.
   // Метод Play начинает воспроизведение файла мультимедиа:
   mediaElement1.Play();
   // Задаем громкость воспроизведения файла мультимедиа в
   // зависимости от состояния элемента "ползунок":
   mediaElement1.Volume = (double)slider1.Value;
}

private void mediaElement1_MediaEnded(object sender, RoutedEventArgs e)
{  // При завершении воспроизведения мультимедийного файла переводим
   // элемент MediaElement в состояние Stop:
   mediaElement1.Stop();
}

private void slider1_ValueChanged(object sender,
                      RoutedPropertyChangedEventArgs<double> e)
{  // Если значение свойства Value элемента "ползунок" изменяется,
   // то изменяем громкость вопроизведения мультимедиа Volume:
   mediaElement1.Volume = (double)slider1.Value;
}

private void button2_Click(object sender, RoutedEventArgs e)
{  // Приостанавливаем воспроизведение файла мультимедиа:
```

```
            mediaElement1.Pause();
        }

        private void button3_Click(object sender, RoutedEventArgs e)
        {  // Останавливаем и возвращаем воспроизведение мультмедийного
            // файла на начало:
            mediaElement1.Stop();
        }
    }
}
```

Как видно из программного кода, сразу после завершения процедуры `InitializeComponent` мы задаем размеры окна и размеры обоих элементов **StackPanel** и указываем, как будут ориентированы дочерние к ним элементы. Далее после очевидных установок переводим элемент **MediaElement** в "ручное" управление `Manual`, что позволит использовать интерактивные методы `Play`, `Pause` и `Stop`. Затем обрабатываем каждое из событий нажатия кнопок, регулировки звука и завершения воспроизведения.

Фрагмент работы программы представлен на рис. 16.7.

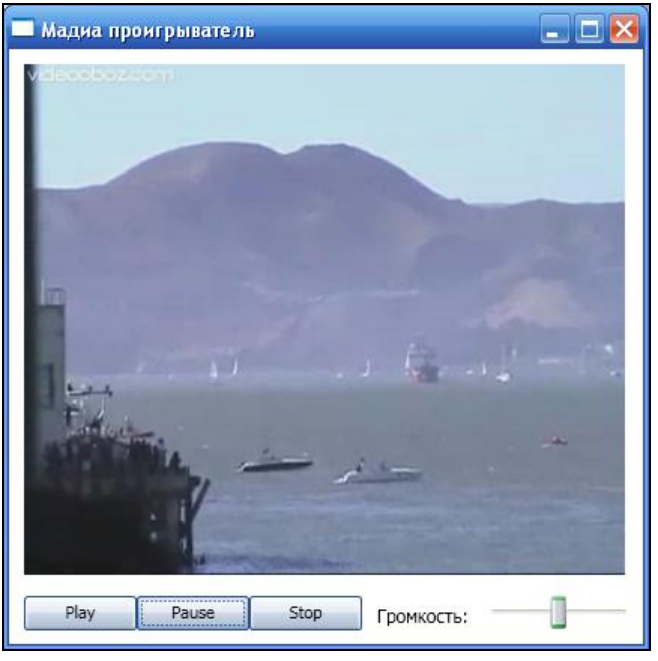


Рис. 16.7. Интерфейс созданного WPF-проигрывателя

Убедиться в работоспособности программы можно, открыв решение в папке WpfПроигрыватель.

Пример 125. Наложение текста на видео

Вспомним видеоролик из предыдущего примера. Тихая морская гавань, размеренная, неторопливая летняя атмосфера. И вдруг в эту умиротворенность врывается рев самолета. Мы видим очень быстро летящий военный истребитель, неведомо как залетевший в эту мирную спящую гавань. Его крылья вот-вот коснутся воды. Будем считать, что этот ролик — начало большого фильма. Наша задача — наложить на это видео одно слово огромными буквами на весь экран: "ПОЛЕТ".

Для реализации поставленной задачи запустим Visual C# 2010, выберем среди шаблонов Windows шаблон **WPF Application**, т. е. автономное WPF-приложение, и назовем его WpfТекстНаВидео. Далее перейдем на вкладку дизайна **MainWindows.xaml**. Здесь в панели **Toolbox** найдем элемент управления **MediaElement** и двойным щелчком перенесем его на вкладку дизайна проектируемого окна. Размеры элемента и прочие параметры будем устанавливать в программном коде. Таким образом, перейдем на вкладку **MainWindows.xaml.cs** и введем текст, представленный в листинге 16.7.

Листинг 16.9. Использование элемента управления *MediaElement* при программировании WPF-проигрывателя

```
// Программа воспроизводит видеоролик с помощью элемента MediaElement
// и накладывает на него форматированный текст "ПОЛЕТ"
using System;
using System.Windows;
using System.Windows.Media;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace WpfТекстНаВидео
{
   /// <summary>
   /// Interaction logic for MainWindow.xaml
   /// </summary>
   public partial class MainWindow : Window
   {
      public MainWindow()
      {
         InitializeComponent();
         this.Title = "Высший пилотаж";
         // Размеры окна:
         this.Width = 420; this.Height = 355;
         mediaElement1.Width = 400; mediaElement1.Height = 300;
         mediaElement1.Source = new Uri(@"C://ВысшийПилотаж.mp4");
         // или mediaElement1.Source = new Uri(@"C:\ВысшийПилотаж.mp4");
         // Создаем форматированный текст:
```

```
            FormattedText ФорматированныйТекст = new FormattedText("ПОЛЕТ",
               System.Globalization.CultureInfo.CurrentCulture,
               FlowDirection.LeftToRight,
               new Typeface(this.FontFamily, FontStyles.Normal,
                  FontWeights.ExtraBold, FontStretches.Normal), 111,
                  Brushes.Red);
            // Создаем объект класса Geometry для отрисовки
            // форматированного текста
            Geometry ГеометрическаяФигура = new PathGeometry();
            // Строим геометрическую фигуру с отрисованным
            // форматированным текстом:
            ГеометрическаяФигура = ФорматированныйТекст.
                                        BuildGeometry(new Point(0, -28));
            // Масштабируем геометрическую фигуру, растягивая ее по вертикали:
            ScaleTransform Масштаб = new ScaleTransform();
            // Указываем коэффициент растяжения по вертикали:
            Масштаб.ScaleY = 3.55;
            ГеометрическаяФигура.Transform = Масштаб;
            // Задаем геометрию в пределах контура медиаэлемента:
            mediaElement1.Clip = ГеометрическаяФигура;
        }
    }
}
```

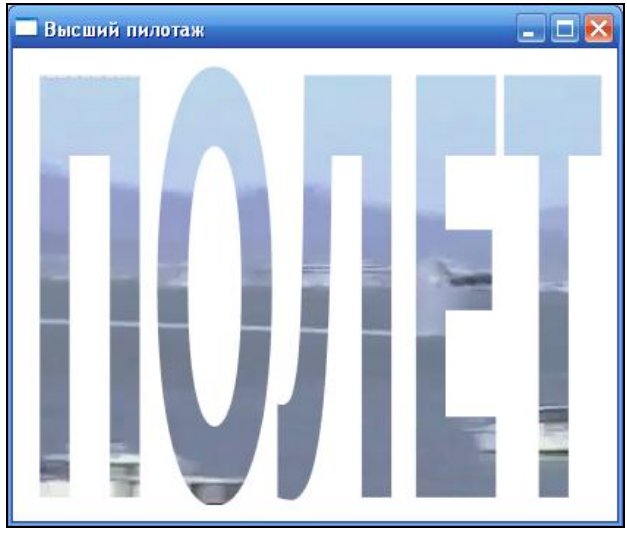


Рис. 16.8. Наложение текста на видео в WPF-приложении

Как видно из текста программы, после выполнения процедуры `InitializeComponent` задаем очевидные параметры для элемента воспроизведения

файлов мультимедиа: его размеры и путь к медиафайлу. Далее создаем форматированный текст класса `FormattedText`. Объекты этого класса предназначены для рисования многострочного текста, в котором каждый символ можно форматировать отдельно. Объект `FormattedText` считается низкоуровневым, поскольку обрабатывает текст как графические элементы. Технология WPF применяет визуализацию на субпиксельном уровне, которая позволяет отображать текст с повышенной точностью формы, выравнивая символы до дробной части пиксела. Далее создаем объект класса `Geometry` для отрисовки форматированного текста, затем, масштабируя его, определяем место (Clip) объекта в пределах контура медиаэлемента. Кстати, свойство `Clip` часто используется для отсечения ("обрезания") от построенной геометрической фигуры некоторой области, предназначенной для визуализации.

Фрагмент работы программы показан на рис. 16.8.

Убедиться в работоспособности программы можно, открыв решение в папке WpfТекстНаВидео.

Пример 126. Переходы в WPF-приложениях

Приложения WPF часто состоят из набора страниц. По мере работы пользователя с приложением он переходит со страницы на страницу, как в Web-приложении. Windows Presentation Foundation поддерживает переходы в стиле браузера, которые могут использоваться в приложениях обоих типов: автономных приложениях и XBAP-приложениях Web-браузера. Для демонстрации возможности перехода от страницы к странице создадим WPF-приложние, выполняемое в браузере. Это приложение будет содержать в себе две Web-страницы. Первая Web-страница имеет кнопку для перехода на вторую Web-страницу и гиперссылку для перехода на почтовый сервер **www.ukr.net**. Вторая Web-страница содержит только гиперссылку для перехода назад на первую Web-страницу. Таким образом, мы продемонстрируем возможность перехода с одной страницы на другую с помощью объекта **NavigationService** и гиперссылки **Hyperlink**.

Итак, запустим Visual Studio 2010, закажем новый проект шаблона **WPF Browser Application C#** (из инсталлированных шаблонов Windows), назовем новый проект WpfXpabПереходы. После щелчка на кнопке **OK** попадем на вкладку **Page1.xaml**. Вызвав панель элементов **Toolbox** (например, с помощью комбинации клавиш <Ctrl>+<Alt>+<X>), дважды щелкнем на элементе **Button**, а затем на элементе **TextBlock**. Таким образом, на вкладке дизайна **Page1.xaml** появятся один элемент над другим: командная кнопка и текстовый блок. На вкладке программного кода **Page1.xaml.cs** введем программный код, представленный в листинге 16.10.

Листинг 16.10. Организация переходов на двугие ресурсы, страница 1

```
// Данное WPF-приложение содержит в себе две Web-страницы: Page1.xaml
// и Page2.xaml. На первой мы разместили командную кнопку и текстовый блок.
```

```
// В программном коде первой страницы мы создали объект Hyperlink,
// чтобы обеспечить переход на почтовый сервер www.ukr.net. Щелчок мышью
// на кнопке реализует переход на вторую Web-страницу Page2.xaml.
// Возврат со второй страницы на первую организован также
// с помощью гиперссылки Hyperlink
using System;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace WpfXpabПереходы
{
   /// <summary>
   /// Interaction logic for Page1.xaml
   /// </summary>
   public partial class Page1 : Page
   {
      public Page1()
      {
         InitializeComponent();
         this.WindowTitle = "Первая WPF-страница";
         button1.Width = 170; button1.Focus();
         button1.Content = "Перейти на вторую страницу";
         textBlock1.Text = "Перейти ";
         // Создаем объект класса Hyperlink для размещения гиперссылки:
         Hyperlink Ссылка = new Hyperlink();
         // Задаем ссылку для перехода на нужный ресурс:
         Ссылка.NavigateUri = new Uri("http://www.ukr.net");
         // Задаем текст, отображаемый в элементе управления Hyperlink:
         Ссылка.Inlines.Add("на почтовый сервер www.ukr.net");
         // Добавляем ссылку в текстовый блок:
         textBlock1.Inlines.Add(Ссылка);
      }

      private void button1_Click(object sender, RoutedEventArgs e)
      {  // Объект NavigationService обеспечивает навигацию на другой ресурс:
         this.NavigationService.Navigate(new Uri("Page2.xaml",
                                          UriKind.Relative));
      }
   }
}
```

Текст программы очевиден из его содержания и приведенных комментариев. Единственное, поясним `Hyperlink`. Такого элемента нет в панели **Toolbox**, поскольку он создается внутри других элементов, например **TextBlock**. Поэтому мы создали объект класса `Hyperlink` в программном коде, задали его некоторые параметры и добавили к элементу **TextBlock** командой `Inlines.Add`. Фрагмент работы Web-страницы Page1.xaml показан на рис. 16.9.

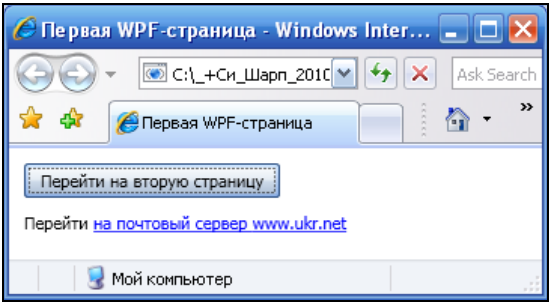


Рис. 16.9. Интерфейс первой Web-страницы WPF-приложения

Теперь создадим вторую Web-страницу, на которую будем переходить, щелкнув по командной кнопке. Для этого в пункте меню **Project** выберем команду **Add New Item** и в появившемся окне дважды щелкнем на шаблоне **Page(WPF)**. На вкладку **Page2.xaml** перенесем из панели элементов **Toolbox** метку **Label** и текстовый блок **TextBlock**. Теперь перейдем на вкладку программного кода **Page2.xaml.cs** и введем текст, представленный в листинге 16.11.

Листинг 16.11. Организация переходов на двугие ресурсы, страница 2

```
using System;
using System.Windows.Controls;
using System.Windows.Documents;
// Другие директивы using удалены, поскольку они не используются
// в данной программе
namespace WpfXpabПереходы
{
   /// <summary>
   /// Interaction logic for Page2.xaml
   /// </summary>
   public partial class Page2 : Page
   {
      public Page2()
      {
         InitializeComponent();
         this.WindowTitle = "Вторая WPF-страница";
```

```
            label1.Content = "Вы перешли на вторую страницу";
            textBlock1.Text = "Щелкните ";
            // Создаем объект класса Hyperlink для размещения гиперссылки:
            Hyperlink Ссылка = new Hyperlink();
            // Задаем ссылку для перехода на нужный ресурс:
            Ссылка.NavigateUri = new Uri("Page1.xaml", UriKind.Relative);
            // Задаем текст, отображаемый в элементе управления Hyperlink:
            Ссылка.Inlines.Add("здесь");
            // Добавляем ссылку в текстовый блок:
            textBlock1.Inlines.Add(Ссылка);
            // Добавляем текст в текстовый блок:
            textBlock1.Inlines.Add(" для перехода назад на первую страницу.");
        }
    }
}
```

Как видно из текста программы, использованный подход аналогичен программной поддержке для первой Web-страницы. Фрагмент работы второй Web-страницы Page2.xaml показан на рис. 16.10.

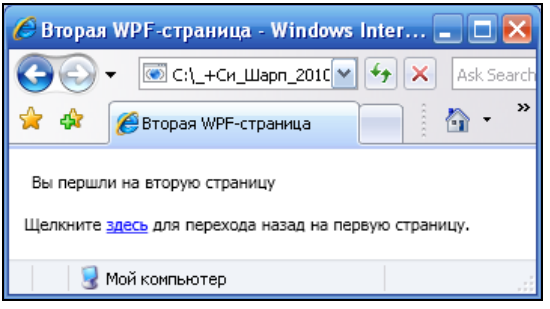


Рис. 16.10. Интерфейс второй Web-страницы WPF-приложения

Убедиться в работоспособности программы можно, открыв решение в папке WpfXpabПереходы.

Желаю вам, уважаемые читатели, получить не только удовольствие от процесса программирования на Visual C#, но и зарабатывать достойные вас деньги.

Извините, все.

Приложение

Описание компакт-диска

Таблица П1. *Содержимое компакт-диска*

Название папки	Описание программы	Номер примера
Hover	Простейшая программа с экранной формой, меткой, командной кнопкой и диалоговым окном, отслеживание события `MouseHove`	2
Корень	Программа вводит через текстовое поле число, при щелчке на командной кнопке извлекает из него квадратный корень и выводит результат на метку `label1`. В случае ввода не числа сообщает пользователю об этом предупреждением красного цвета также на метку `label1`	3
Passport	Программа для ввода пароля в текстовое поле, причем при вводе вместо вводимых символов некто, "находящийся за спиной пользователя", увидит только звездочки	4
CheckBox1	Программа управляет стилем шрифта текста, введенного на метку **Label** посредством флажка **CheckBox**	5
CheckBox2	Совершенствование предыдущей программы. Побитовый оператор `^` — исключающее ИЛИ	6
ВкладкиTabControl	Программа, позволяющая выбрать текст из двух вариантов, задать цвет и размер шрифта для этого текста на трех вкладках **TabControl** с использованием переключателей **RadioButton**	7
Visible	Программа пишет в метку **Label** некоторый текст, а пользователь с помощью командной кнопки делает этот текст либо видимым, либо невидимым. Здесь использовано свойство `Visible`. При зависании мыши над кнопкой появляется подсказка "Нажми меня" (свойство `ToolTip`)	8

Таблица П1 *(продолжение)*

Название папки	**Описание программы**	**Номер примера**
ComboBox_Calc	Программа, реализующая функции калькулятора. Здесь для отображения вариантов выбора арифметических действий используется комбинированный список **ComboBox**	9
Unico	Программа демонстрирует возможность вывода в текстовую метку, а также в диалоговое окно `MessageBox` греческих букв. Программа приглашает пользователя ввести радиус *R*, чтобы вычислить длину окружности	10
Сумма	Консольное приложение складывает два числа и выводит сумму на консоль	11
ТаблКорней	Консольное приложение задает цвета и заголовок консоли, а затем выводит таблицу извлечения квадратного корня от нуля до десяти	12
ConsoleMessageBox	Консольное приложение выводит в окно `MessageBox` текущую дату и время в различных форматах, используя `String.Format`	13
ССылкаНаVisualBasic	В данном консольном приложении Visual C# используем функции Visual Basic. Приложение приглашает пользователя ввести два числа, анализирует, числа ли ввел пользователь, и выводит результат суммирования на экран. При этом используются функции Visual Basic: `InputBox`, `IsNumeric` (для контроля, число ли ввел пользователь) и `MsgBox`	14
Monitoring	Программа отображает координаты курсора мыши относительно экрана и элемента управления. Программа содержит форму, список элементов **ListBox** и два текстовых поля. Программа заполняет список **ListBox** данными о местоположении и изменении положения курсора мыши. Кроме того, в текстовых полях отображаются координаты положения курсора мыши относительно экрана и элемента управления **ListBox**	15
NewButton	Программа создает командную кнопку в форме "программным" способом, т. е. с помощью написания непосредственно программного кода, не используя при этом панель элементов управления **Toolbox**. Программа задает свойства кнопки: ее видимость, размеры, положение, надпись на кнопке и подключает событие "щелчок на кнопке"	16

Таблица П1 *(продолжение)*

Название папки	**Описание программы**	**Номер примера**
ДваСобытияОднаПроц	В форме имеем две командные кнопки, и при нажатии указателем мыши любой из них получаем номер нажатой кнопки. При этом в программе предусмотрена только одна процедура обработки событий	17
Calc	Программа Калькулятор с кнопками цифр. Управление калькулятором возможно только мышью. Данный калькулятор выполняет лишь арифметические операции	18
ССылкиLinkLabel	Приложение Windows обеспечивает ссылку для посещения почтового сервера **www.mail.ru**, ссылку для просмотра папки C:\Windows\ и ссылку для запуска текстового редактора Блокнот с помощью элемента управления **LinkLabel**	19
Key	Обработка событий клавиатуры. Программа, которая информирует пользователя о тех клавишах и комбинациях клавиш, которые тот нажал	20
NumbersOnly	Программа анализирует каждый символ, вводимый пользователем в текстовое поле формы. Если символ не является числовым, то текстовое поле получает запрет на ввод такого символа. Таким образом, программа не дает возможность пользователю ввода не числовых данных	21
ТолькоЧисло+ТчкОрЗпт	Программа разрешает ввод в текстовое поле только цифровых символов, а также разделителя целой и дробной частей числа (т. е. точки или запятой)	22
TXT_Unicode	Программа для чтения/записи текстового файла в кодировке Unicode	23
TXT_1251	Программа для чтения/записи текстового файла в кодировке Windows 1251	24
ТекстовыйРедактор	Простой текстовый редактор. Открытие и сохранение файла в диалоге. Событие формы `Closing`	25
Тестирование	Программа тестирует студента по какому-либо предмету обучения	26
RTF_edit	Программа простейшего RTF-редактора	27
TXT_print	Печать текстового документа. Программа имеет такие возможности: открыть в стандартном диалоге 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	28

Таблица П1 *(продолжение)*

Название папки	**Описание программы**	**Номер примера**
Read_Write_bin	Программа для чтения/записи бинарных файлов с использованием потока данных	29
Simple_Image1, Simple_Image2 и Simple_Image3	Программа выводит в форму изображение растрового графического файла формата BMP, JPG, PNG или других форматов	30
БольшойРисунокСкроллинг	Программа выводит изображение из растрового файла в элемент управления **PictureBox**, размещенный на элементе управления **Panel**, с возможностью прокрутки изображения	31
РисМышью	Программа позволяет при нажатой левой или правой кнопке мыши рисовать в форме	32
РисФигур	Программа позволяет рисовать в форме графические примитивы: окружность, отрезок, прямоугольник, сектор, текст, эллипс и закрашенный сектор. Выбор того или иного графического примитива осуществляется с помощью элемента управления **ListBox** (Список)	33
ВыборЦвета1 и ВыборЦвета2	Программа меняет цвет фона формы `BackColor`, перебирая константы цвета, предусмотренные в Visual Studio 2010, с помощью элемента управления **ListBox**	34
ПечатьЭллипса	Печать графических примитивов. Программа выводит на печать (на принтер) изображение эллипса. Понятно, что таким же образом можно распечатывать и другие графические примитивы: прямоугольники, отрезки, дуги и т. д. (см. методы объекта `Graphics`)	35
ПечатьBMPфайла	Программа выводит на печать графический файл формата BMP	36
График	Программа рисует график продаж по месяцам. Понятно, что таким же образом можно построить любой график по точкам для других прикладных целей	37
БуферОбменаTXT	Программа для управления буфером обмена в текстовом формате. Программа имеет возможность записи какого-либо текста в буфер обмена, а затем извлечения этого текста из буфера обмена	38
БуферОбменаBitmap	Программа оперирует с буфером обмена, когда тот содержит изображение	39

Таблица П1 *(продолжение)*

Название папки	**Описание программы**	**Номер примера**
AltPrintScreen	Программная имитация нажатия клавиш <Alt>+<PrintScreen> с помощью функции Microsoft Windows API, а также методом `Send` класса `SendKeys`	40
БуферОбменаSaveBMP	Программа читает буфер обмена, и если данные в нем представлены в формате растровой графики, то записывает их в BMP-файл	41
ПростоТаймер	Демонстрация использования таймера **Timer**. После запуска программы показываются форма и элемент управления **ListBox**. Через две секунды в списке элементов появляется запись "Прошло две секунды", и через каждые последующие две секунды в список добавляется аналогичная запись	42
SaveСкриншотКаждые5сек	Программа после запуска каждые пять секунд делает снимок текущего состояния экрана и записывает эти снимки в файлы Pic1.BMP, Pic2.BMP и т. д. Количество таких записей в файл — пять	43
ТаблTxt	Программа формирует таблицу из двух строковых массивов в текстовом поле, используя функцию `String.Format`. Кроме того, в программе участвует элемент управления **MenuStrip** для организации выпадающего меню, с помощью которого пользователь выводит сформированную таблицу в Блокнот с целью последующего редактирования и вывода на печать	44
ТаблTxtPrint	Программа формирует таблицу на основании двух массивов переменных с двойной точностью. Данную таблицу программа демонстрирует пользователю в текстовом поле **TextBox**. Есть возможность распечатать таблицу на принтере	45
Табл_HTM	Вывод таблицы в Internet Explorer. Здесь реализован несколько необычный подход к выводу таблицы для ее просмотра и печати на принтере. Программа записывает таблицу в текстовый файл в формате HTML. Теперь у пользователя появляется возможность прочитать эту таблицу с помощью обозревателя Web-страниц Internet Explorer или другого браузера	46
ТаблGrid	Программа заполняет два строковых массива и выводит эти массивы на экран в виде таблицы, используя элемент управления **DataGridView** (Сетка данных). Элемент управления **DataGridView** предназначен для просмотра таблиц с возможностью их редактирования	47

Таблица П1 *(продолжение)*

Название папки	Описание программы	Номер примера
ТаблВвод	Программа предлагает пользователю заполнить таблицу телефонов его знакомых, сотрудников, родственников, любимых и т. д. После щелчка на кнопке **Запись** данная таблица записывается на диск в файл в формате XML. Для упрощения текста программы предусмотрена запись в один и тот же файл C:\tabl.xml. При последующих запусках данной программы таблица будет считываться из этого файла, и пользователь может продолжать редактирование таблицы	48
ГауссGrid	Программа для решения системы линейных уравнений. Ввод коэффициентов предусмотрен через **DataGridView**	49
ТаблWebHTM	В программе для отображения таблицы используется элемент управления **WebBrowser**. Таблица записана на языке HTML с помощью элементарных тегов `<tr>` (строка в таблице) и `<td>` (ячейка в таблице)	50
FlashWeb	Программа использует элемент управления **WebBrowser** для отображения Flash-файлов	51
Split	Эта программа использует элемент управления **WebBrowser** для отображения Web-страницы и ее HTML-кода	52
ЗаполнениеWeb_формы	Программа загружает в элемент WebBrowser начальную страницу поисковой системы **http://yahoo.com**. Далее, используя указатель на неуправляемый интерфейс, заполняет Web-форму поисковой системы, а затем отправляет заполненную форму на сервер. В итоге получаем в элементе **WebBrowser** результат работы поисковой системы, а именно множество ссылок на страницы, содержащие указанные ключевые слова	53
Орфография	Программа позволяет пользователю ввести какие-либо слова, предложения в текстовое поле и после нажатия соответствующей кнопки проверить орфографию введенного текста. Для непосредственной проверки орфографии воспользуемся функцией `CheckSpelling` объектной библиотеки MS Word	54
ТаблицаWord	Программа вывода таблицы средствами MS Word: запускается программа, пользователь наблюдает, как запускается редактор MS Word и автоматически происходит построение таблицы	55

Таблица П1 *(продолжение)*

Название папки	Описание программы	Номер примера
ExcelПи	Программа обращается к одной простой функции объектной библиотеки MS Excel для получения значения числа $\pi = 3{,}14$	56
ExcelПлт	Программа использует финансовую функцию `Pmt()` объектной библиотеки MS Excel для вычисления суммы периодического платежа на основе постоянства сумм платежей и постоянства процентной ставки	57
ExcelСЛАУ	Программа решает систему уравнений с помощью функций объектной библиотеки MS Excel	58
ExcelГрафик	Программа строит график (диаграмму) средствами объектов компонентной библиотеки MS Excel	59
ACADЧертеж	Программа строит средствами объектов библиотеки AutoCAD элементарный чертеж из отрезков и некоторого текста. Этот чертеж сохраняется в файле формата DWG. Конкретнее: эта программа запускает AutoCAD 2000i, рисует два отрезка, два текстовых объекта, сохраняет чертеж в файле C:\Чертеж.dwg и завершает работу AutoCAD	60
MatlabВызов	Программа, подготовив соответствующие команды для MATLAB, вызывает его на выполнение этих команд. В результате строится затухающая синусоида $y = \sin(x) \times e^{-x}$	61
MatlabСлау	Программа, подготовив команды для решения системы уравнений в среде MATLAB, вызывает ее на выполнение этих команд. В результате получаем решение, которое выводим его на экран с помощью `MessageBox`	62
БД_SQL_Server	Создание базы данных SQL Server	63
БД_SQL_Server2	Отображение таблицы базы данных SQL Server в экранной форме	64
БД_mdb	Отображение таблицы базы данных MS Access в экранной форме	66
БДDataReader1	Программа читает все записи из таблицы БД MS Access и выводит их на консоль с помощью объектов `Command` и `DataReader`	67
БДСоздание	Программа создает базу данных MS Access, т. е. файл new_BD.mdb. Эта база данных будет пустой, т. е. не будет содержать ни одной таблицы. Наполнять базу данных таблицами можно будет впоследствии как из программного кода C# 2010, так и используя MS Access. В этом примере технология ADO.NET не использована	68

Таблица П1 *(продолжение)*

Название папки	Описание программы	Номер примера
БдСоздТаблицы	Программа записывает структуру таблицы в пустую базу данных MS Access. Программная реализация подключения к БД. В этой БД может еще не быть ни одной таблицы, т. е. БД может быть пустой. Либо в БД могут уже быть таблицы, но название новой таблицы должно быть уникальным	69
БдДобавлЗаписи	Программа добавляет запись в таблицу базы данных MS Access. Для этого при создании экземпляра объекта `Command` задаем SQL-запрос на вставку (Insert) новой записи в таблицу базы данных	70
БдReaderGridView	Программа читает все записи из таблицы базы данных с помощью объектов классов `Command`, `DataReader` и элемента управления **DataGridView**	71
БдАдаптерGridView	Программа читает из БД таблицу в сетку данных **DataGridView** с использованием объектов классов `Command`, `Adapter` и `DataSet`	72
БдUpdate	Программа обновляет записи (Update) в таблице базы данных MS Access	73
БдУдаленЗаписи	Программа удаляет запись из таблицы БД с использованием SQL-запроса и объекта класса `Command`	74
Linq1	Решаем две задачи по выбору элементов из массива с помощью стандартных запросов технологии LINQ	75
Linq2	Решаем три различных задачи по выбору элементов (объектов) из списка с помощью стандартных запросов технологии LINQ	76
LinqЦеныНаПродукты	Программа формирует список некоторых продуктов питания. Первый LINQ-запрос группирует элементы списка по критерию цены: в первом списке оказываются продукты, цена за единицу которых меньше или равна 90 руб., а во втором, соответственно, больше 90 руб. Второй LINQ-запрос вычисляет среднюю цену продукта по каждой группе. Результаты запросов выводятся в текстовое поле	77
LinqDictionary	Задаем массив сотрудников учреждения. Из этого массива создаем словарь сотрудников, а в качестве ключа к этому словарю выбираем имя сотрудника. С помощью LINQ-запроса из массива сотрудников выбираем тех, чей возраст превышает 33 года.	78

Таблица П1 *(продолжение)*

Название папки	Описание программы	Номер примера
	При выводе результата запроса на печать учитываем, что мы говорим "47 лет", но "34 года". То есть если из возраста вычесть число, кратное 10, то при остатке меньше 5 говорят, например, "34 года", а при остатке больше или равном 5 говорят "47 лет"	
LinqСоздатьXML-документ	Программа создает типичный XML-документ. С ее помощью можно разобраться в структуре XML-документа. В комментариях приведена терминология содержимого XML-документа: корневой элемент, вложенные элементы, имя элемента и его значение, а также атрибуты элемента, их имена и значения. XML-документ представляет телефонную книгу, содержащую имя контакта, номер домашнего телефона, а также мобильного. Программа после создания XML-документа отображает его на консоли, а также записывает его в файл. Если этот файл открыть с помощью MS Excel, то мы получим таблицу из трех столбцов	79
Linq3 и Linq4	Дана строка XML, содержащая прогнозные метеорологические показатели для Москвы на заданную дату. Программа извлекает из корневого элемента XML-документа значение температуры элемента `Температура`	80
Linq5	Имеем XML-данные, в которых содержится таблица с именами и телефонами, причем имена в этой телефонной табличке повторяются. Задача состоит в том, чтобы в данной таблице телефонов (представленной в виде XML) найти все строчки с именем "Витя" с помощью LINQ-запроса	81
LinqГорода	В данной программе экранная форма содержит элемент управления для отображения и редактирования табличных данных **DataGridView**, две командные кнопки и текстовое поле. При старте программы, если есть соответствующий файл XML, программа отображает в **DataGridView** таблицу городов — название города и численность населения. При щелчке на кнопке **Сохранить** все изменения в таблице записываются в XML-файл. При щелчке на второй кнопке **Найти** выполняется LINQ-запрос к набору данных `DataSet` на поиск городов-миллионеров в искомой таблице. Результат запроса выводится в текстовое поле	82

Таблица П1 *(продолжение)*

Название папки	**Описание программы**	**Номер примера**
LinqToSqlГорода	Данное Windows-приложение состоит из экранной формы и элемента управления **DataGridView**. В программе организован LINQ-запрос к базе данных городов с помощью базового класса сущностей `DataContext`. Посредством этого класса в данной программе организован LINQ-запрос к базе данных на получение коллекции (списка) городов, численность населения в которых превышает миллион жителей. Результат запроса выведен на элемент управления **DataGridView**	83
ПроверкаФамилии	Проверка данных, вводимых пользователем, на достоверность. Программа осуществляет синтаксический разбор введенной пользователем текстовой строки на соответствие ее фамилии на русском языке	84
ПроверкаЧисла	Проверка данных, вводимых пользователем, на достоверность. Программа осуществляет синтаксический разбор введенной пользователем текстовой строки на соответствие ее положительному рациональному числу	84
Opacity	Программа демонстрирует стандартную форму. Щелчок мышью в пределах этой формы начинает постепенный процесс исчезновения формы: форма становится все более прозрачной, а затем исчезает вовсе. Далее она постепенно проявляется снова, и т. д. Еще один щелчок в пределах формы останавливает этот процесс, а следующий щелчок процесс возобновляет и т. д.	85
Гринвич	Программа в полупрозрачной экранной форме отображает текущее время по Гринвичу. Таким образом, программа демонстрирует текущее время по Гринвичу и при этом не заслоняет собой другие приложения	86
Значок_в_области_уведомлений	Эта программа сообщает пользователю время, прошедшее с момента старта операционной системы на данном компьютере. Доступ к этой информации реализован через контекстное меню значка в области уведомлений панели задач	87
ПеремещениеФормы	Нестандартная форма. Программа позволяет перемещать форму мышью, "зацепив" ее не только за заголовок, а в любом месте формы	88
Player	Программа реализует функции проигрывателя Windows Media Player 11	89

Таблица П1 *(продолжение)*

Название папки	Описание программы	Номер примера
Help	В программе предусмотрена экранная форма, которая в заголовке имеет только кнопку **Справка** (в виде вопросительного знака) и кнопку **Закрыть**. Здесь реализована контекстная помощь, когда после щелчка мыши на кнопке **Справка** можно получить контекстную всплывающую подсказку по тому или иному элементу управления, находящемуся в форме	90
Summa	Создание простейшей активной Web-страницы на Visual C# 2010. Web-страница демонстрирует способность складывать числа, введенные пользователем	91
Valid1	Проверка введенных пользователем числовых данных с помощью валидаторов. Выясняется, ввел ли пользователь хоть что-либо в текстовые поля, а также определяется тип введенных данных и выясяется, соответствуют ли они типу данных `Double`	92
Validations	Проверка достоверности ввода имени, адреса e-mail, URL-адреса и пароля. Web-страница проверяет ввод всех этих сведений, например при регистрации пользователя. Причем если страница (Web-форма) успешно прошла все этапы проверки, то направляем пользователя на другую, уже разрешенную для этого пользователя, Web-страницу	93
Login	Регистрация и аутентификация пользователя с помощью базы данных MS Access. Данный пример включает в себя три Web-формы: Registration.aspx, Login.aspx и Secret.aspx. Первая форма Registration.aspx приглашает пользователя ввести регистрационные данные, проверяет правильность ввода имени пользователя и пароля с использованием валидаторов, регистрирует пользователя в базе данных MS Access и перенаправляет пользователя на уже разрешенный после регистрации ресурс Secret.aspx. Вторая форма Login.aspx запрашивает имя пользователя и пароль, проверяет наличие пользователя с таким именем и паролем в базе данных. Если такого пользователя не оказалось, то форма отправляет пользователя на регистрацию Registration.aspx, а если есть, то он получает доступ к ресурсу Secret.aspx. К третьей странице имеют доступ только зарегистрированные пользователи. Поэтому здесь выясняется, с какй страницы пришел сюда пользователь	94

Таблица П1 *(продолжение)*

Название папки	Описание программы	Номер примера
tab	Таблица с переменным числом ячеек, управляемая двумя раскрывающимися списками. Web-страница позволяет с помощью двух раскрывающихся списков **DropDownList** заказать необходимое число рядов и столбцов в таблице, а затем построить заказанную таблицу	95
Menu	Web-страница демонстрирует, как можно организовать переход на разные страницы сайта (гиперссылки) с помощью раскрывающегося списка **DropDownList**	96
Ssylka	Передача данных между Web-страницами через параметры гиперссылки. В данном примере имеем две Web-страницы: Source.aspx и Target.aspx. На первой странице Source.aspx с помощью генератора случайных чисел `Random` выбираем одну из пар "имя—фамилия", затем кодируем их, чтобы они не были видны в адресной строке. Щелчок пользователя по гиперссылке вызывает переход на страницу Target.aspx, причем в гиперссылке указаны оба закодированных параметра	97
Передача	Web-страница получает имя пользователя и адрес его электронной почты от Web-формы и отображает эти данные на странице	98
Передача2	На начальной Web-странице имеем командную кнопку **ПЕРЕХОД** и текстовое поле, которое заполняет пользователь. После щелчка на кнопке происходит переход на другую Web-страницу. На новой странице отображается содержимое текстового поля и надпись на кнопке из предыдущей страницы	99
TabGrdWeb	Вывод табличных данных в Web-форму с помощью элемента управления **GridView**. В данной Web-странице организован вывод двух строковых массивов в таблицу Web-формы с помощью элемента управления **GridView** и объекта класса `DataTable`	100
Hash_Grid	Вывод в Web-форму хэш-таблицы, которая позволяет поставить в соответствие государства их столицам. То есть в качестве ключей имеем государства, а их столицы — в качестве значений. Далее, используя элемент управления **GridView**, программа выводит эту хэш-таблицу на Web-страницу	101

Таблица П1 *(продолжение)*

Название папки	**Описание программы**	**Номер примера**
RW_txt	Чтение/запись текстового файла Web-приложением. Web-приложение читает текстовый файл в текстовое поле, а пользователь имеет возможность редактировать текст и сохранять его в том же файле	102
Counter	Web-приложение, реализующее счетчик посещений сайта с использованием базы данных и объекта `Session`	103
Cookie	Чтение/запись cookie-файлов. Web-страница предлагает посетителю ввести данные о себе: имя и род занятий. При нажатии кнопки **Запись Cookie** введенные в текстовые поля сведения будут записаны в cookie-файл. Этот cookie-файл будет храниться на компьютере пользователя сутки. В течение этих суток, каждый раз вызывая данную страницу, в текстовых полях мы будем видеть введенные нами сведения, которые мы можем тут же исправлять и опять записывать в cookie	104
Web_ изображение	На странице имеем изображение — файл pogyv.png, при щелчке мышью на нем изображение увеличивается вдвое без перезагрузки Web-страницы. В этом же проекте решена также другая задача. На Web-странице имеем изображение, например мужчины, — файл m.jpg. Это изображение используем для ссылки на другую Web-страницу, например на WebForm1.aspx. Причем при наведении на него указателя мыши происходит смена изображения на изображение женщины — файл g.jpg	105
ТекстНаклWeb	Web-страница формирует файл изображения методами класса `Graphics`. На изображение выводится текстовая строка, наклоненная к горизонту на 356° (наклон вверх). Далее этот файл изображения демонстрируется в форме	106
ГостеваяКнига	Данная Web-страница приглашает посетителя оставить какие-либо записи, которые могут прочитать другие посетители страницы. Записи сохраняются в текстовом файле kniga.txt. Записи отображаются на Web-странице с помощью сетки данных **GridView**	107
Капча	Программа тестирует пользователя на "человечность", предлагает распознать искаженную строку символов	108

Таблица П1 *(продолжение)*

Название папки	Описание программы	Номер примера
AjaxTimer	Web-страница демонстрирует время на текстовой метке `Label1`. На эту метку каждую секунду копируем новое время, но обновляем при этом не всю форму, а только метку с помощью технологии AJAX	109
WebКлиентПогода	Web-приложение, потребляющее сервис удаленной Web-службы прогноза погоды. Приложение в текстовом поле **TextBox** демонстрирует XML-строку с параметрами погоды для города, указанного во входных параметрах при обращении к Web-службе. Также выводит в текстовую метку значение температуры в этом городе	110
WindowsКлиентПогода	Windows-приложение, потребляющее сервис удаленной Web-службы прогноза погоды. Приложение в текстовом поле **TextBox** демонстрирует XML-строку с параметрами погоды для города, указанного во входных параметрах при обращении к Web-службе. Также выводит в текстовую метку значение температуры в этом городе	111
WebСлужбаСумма	На входе данной Web-службы предлагается ввести два числа, а Web-служба берет на себя функцию сложения этих двух чисел и вывода (возврата) суммы. При этом Web-служба производит диагностику вводимых данных	112
WebКлиентСумма	Клиентское Windows-приложение, потребляющее сервис Web-службы предыдущего примера WebСлужбаСумма	113
WebСлужбаForex	Web-служба, которая с помощью синтаксического разбора Web-страницы **http://www.forex-rdc.ru/subscribers.php?action=prognoz** извлекает торговую рекомендацию на рынке Forex для валютной пары EURUSD, предлагаемую данным сайтом на текущий день, и выводит ее потребителю сервиса Web-службы в виде строки	114
WebКлиентForex	Получаем прогноз рынка Forex на текущий день. Клиентское Windows-приложение, потребляющее сервис Web-службы предыдущего примера WebСлужбаForex	115
WebКлиентMorpher	Клиентское Web-приложение, потребляющее сервис Web-службы склонения существительных "Морфер". На вход метода Web-службы подаем слово на русском языке, на выходе получаем это слово в различных пяти падежах	116

Таблица П1 *(продолжение)*

Название папки	Описание программы	Номер примера
WebКлиентРоссия	Клиентское Web-приложение, потребляющее сервис Web-службы Центрального банка России для получения ежедневных курсов валют. На выходе приложения получаем таблицу курсов валют	117
WebКлиентБеларусь и WebКлиентБеларусьWeb	Клиентское Windows-приложение и клиентское Web-приложение, потребляющие сервис Web-службы Национального банка Республики Беларусь для получения ежедневных курсов валют. На выходе приложений получаем таблицу курсов валют	118
WpfXbarSqrt	Данное WPF-приложение вычисляет значение квадратного корня из числа, введенного пользователем в текстовое поле. После щелчка на кнопке приложение производит диагностику введенных символов, и если пользователь действительно ввел число, то в текстовую метку выводим результат извлечения корня	119
WpfXbarАниме	WPF-приложение выводит на Web-страницу командную кнопку и изображение. При щелчке на кнопке и на изображении демонстрируются возможности анимации: кнопка расширяется, а затем медленно уменьшается до исходных размеров; аналогично поведение изображения	120
WpfXbarДваИзо	WPF-приложение содержит на Web-странице два изображения. Поскольку месторасположение обоих изображений задано одинаково, а также совпадают размеры изображений, пользователь будет видеть только второе "верхнее" изображение. После щелчка на изображении оно становится все более прозрачным, постепенно "проявляя" тем самым "нижнее" изображение. После исчезновения "верхнего" изображения мы будем видеть только "нижнее" изображение. При повторном щелчке на изображении, наоборот, прозрачность верхнего изображения постепенно снижается, и в конце анимационного эффекта мы опять видим лишь "верхнее" изображение	121
WpfGradientText	Автономное WPF-приложение содержит текстовый блок. Цвет текста в этом блоке закрашен с применением градиента. Между начальной `t=0.0` и конечной `t=1.0` точками области текста заданы две ключевые точки `t=0.25` и `t=0.75`. Каждой точке ставим в соответствие цвета: желтый, красный, синий и зеленый. Между этими цветами задаем плавный переход от одного цвета к другому с помощью градиента	122

Таблица П1 *(продолжение)*

Название папки	Описание программы	Номер примера
WpfTextBoxОрфография	Автономное WPF-приложение содержит элемент управления **TextBox** с включенной проверкой правописания англоязычного текста. Технология .NET Framework 4 WPF обеспечивает только английский, французский, немецкий и испанский словари. Чтобы появилась возможность проверять русскоязычный текст, следует в коллекцию `CustomDictionaries` добавить пользовательский словарь русскоязычной лексики — файл с расширением lex	123
WpfПроигрыватель	WPF-проигрыватель, позволяющий воспроизводить мультимедиа, включать паузу, остановку, а также настраивать громкость с помощью "ползунка"	124
WpfТекстНаВидео	Программа воспроизводит видеоролик с помощью элемента **MediaElement** и накладывает на него форматированный текст "ПОЛЕТ"	125
WpfXpabПереходы	Данное WPF-приложение содержит в себе две Web-страницы: Page1.xaml и Page2.xaml. На первой мы разместили командную кнопку и текстовый блок. В программном коде первой страницы мы создали объект `Hyperlink`, чтобы обеспечить переход на почтовый сервер **www.ukr.net**. Щелчок мышью на кнопке реализует переход на вторую Web-страницу Page2.xaml. Возврат со второй страницы на первую организован также с помощью гиперссылки `Hyperlink`	126

В папке "xx Дополнительные материалы" на компакт-диске также содержатся файлы картинок, баз данных, файл словаря и пример HTML-таблицы, используемые при разработке рассмотренных в книге примеров.

Предметный указатель

А

Б

В

Г

Д

И

К

М

Н

О

П

Р

С

Т

Ф

Х

Ц

Э